SOLIDIFICATION Whitepaper

Static Functional Verification with Solidify

A New Low-Risk Methodology for Faster Debug of ASICs and Programmable Parts

averant

Abstract

The growing complexity of ASICs and programmable parts means functional verification is the nightmare that keeps project managers up at night. Designers are creating ASICs that can't be completely verified in a reasonable time with the tools and computing resources they have available. As a result of the gap between what can be designed and what can be verified, achieving a functionally stable design is difficult and involves many iterations.

This paper presents ***Solidification***™, a new low-risk methodology for faster debug of ASICs and programmable parts that significantly reduces verification time and effort while increasing quality and robustness of designs.

Introduction

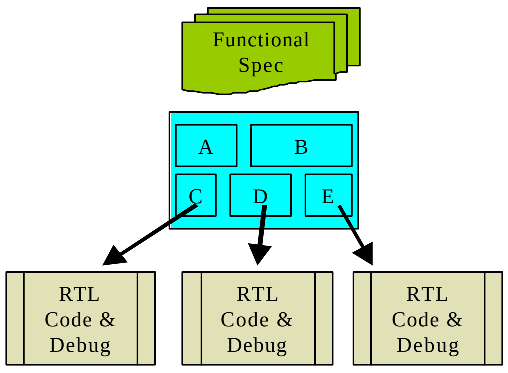
Experts in HDL design and verification formed Averant in 1997 to create innovative high-performance design automation software. Its first product, Solidify, is an interactive, easy-to-use design tool for static functional verification. RTL designers, architects, and validation engineers use Solidify to verify that the blocks in their design comply with their specifications. Solidify's static approach uses no vectors, and employs an exhaustive analysis method that is guaranteed 100% for any verified behavior. Besides RTL verification, Solidify detects inconsistencies in functional specifications, and includes static coverage technology for verification assurance. In one tool, Solidify offers designers an alternative to test vector generation, functional simulation, and code coverage.

Verification Gap and Functional Non-Convergence

> *Running test benches on today's large ASICs still "takes too long, too much memory, and is too painful a process."*
>
> *– Gary Smith, Chief EDA Analyst, Dataquest, Feb. 1999, ISD Magazine*

Complexity of ASICs and programmable parts is continuing to rise. The number of ASIC designs in the range of 100K-1M gates has jumped from 42% to 60% according to Dataquest (November 1998). Similarly, the average programmable part has a size of 60K gates and will be 100K in the year 2000. This increase in complexity is causing designers to spend more time struggling to verify that their designs meet specifications and are without bugs. Studies have shown that verification effort is now 50-80% of the total design effort and is continuing to be a major bottleneck in the design cycle.

In a typical design flow, the chip architect writes the functional specification as a text document that describes the partitioning of the design and the functionality of each block. The blocks are then assigned to individual designers for RTL coding and debug. For each block, interface specifications and functional test plans may be written. As chip sizes continue to grow, the individual blocks are more numerous and complex.



The traditional verification methodology is to create a set of vectors either manually, or by means of a program. These custom programs are written in C, Perl, or HDL, or a test vector generation language is used. After the vectors are created, simulation is used to observe the block's response to the vector set. To determine correct behavior, either reference models or rule-based checks (monitors) are used to analyze the simulation results.

Verification typically ends when code coverage determines that the test bench exercised all source lines. However this may not uncover all the corner cases. So random testing continues until the number of bugs found approaches zero.

There are a number of problems with the current approach:

Requires test-benches. Generation of testbenches is difficult and time-consuming and engineers would avoid it if they could. Testbench generation for basic functionality is often a tedious process such as trying to verify a finite-state-machine will never reach an illegal state or that a state-vector will always be one-hot. Depending on the number of inputs and internal state-holding elements, an exponential explosion occurs in the number of vectors needed. This is particularly true for covering corner conditions. The number of vectors must be large to increase the chance of covering all the corner cases.

Already designers have to know Verilog, Perl and C to script their verification suites. Requiring them to learn yet another complex language for testbench generation creates additional barriers to getting effective verification of their designs. Even when a testbench generation language is used, the methodology is still vector-based with all its associated difficulties.

The testbench and verification suites are typically not easy to maintain and involve a large collection of files that run to many megabytes. Any change to the RTL involves significant effort to update and re-verify the testbench and verification suite.

As the models and testbenches are becoming larger, simulation times and engineering effort are growing drastically. Faster simulators and workstations, and larger teams partially address these problems, but what is required is an improvement that is several orders of magnitude better.

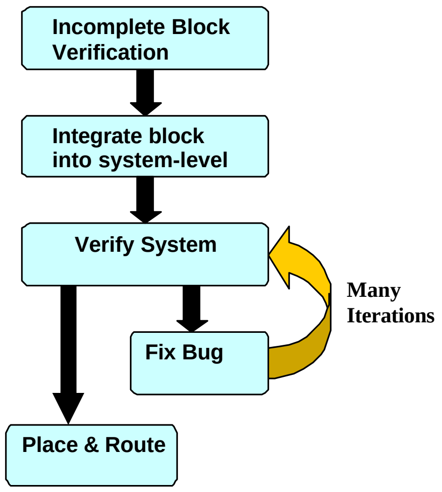
Coverage Effort. Traditional code coverage analysis gives a measure of the completeness of the verification, typically by ensuring all the lines of the HDL were executed. Unfortunately it slows the simulator which is already a bottleneck. Typically getting coverage to 90% is not too difficult. This is not sufficient however. The testbench verification suite should provide 100% coverage as measured by a coverage tool. Getting to 100% coverage often requires a huge effort to create and verify those vectors. Even when a vector set gives 100% coverage, unless all possible vector combinations are used, a corner case could still be missed.

Complex Verification Environment. Combining a simulator with a waveform viewer, code coverage tool, test bench generator, and simulation monitor creates a complex verification environment for the designer. Each of these individual tools must be mastered to be used effectively. Unfortunately each of these tools slows down the design and debug process because of the tool interactions and data translation that are required. Additionally the cost to the CAD department to maintain and support these tools is significant.

Functional Non-Convergence

When the various blocks in a system are combined, the verification challenge becomes much greater since many more lines of RTL code need to be simulated. Bugs found at this stage are typically due to incomplete verification of the component blocks, incorrect interface specifications, and miscommunication between members of the design team. Once found, edits at the block level are used to fix the problem. However instead of updating and re-running the block-level testbenches, the patched block is re-inserted into the system for further debugging. Often the effort to create block level testbenches is thrown away as maintaining block-level testbenches is not trivial. The danger in not re-running verification on the blocks is significant. Applying changes to code often can introduce additional bugs.

Once a change in the design has been introduced, chip or system level verification continues to find any remaining (or new) bugs. By avoiding complete verification at the block level their discovery is postponed until the more difficult and lengthy system simulation is attempted. Later discovery leads to more instability at the system-level and delays achieving confidence that most bugs have been found before committing the design to silicon. This functional non-convergence leads to a longer debug cycle, with less certainty that all design errors have been detected. In some projects, the design never becomes stable and causes it to be cancelled.


Incomplete Block Verification
Integrate block into system-level
Verify System
Fix Bug
Place & Route
Many Iterations

Incompletely verified blocks being brought together cause functional non convergence. Numerous bug appear at system integration. Many iteration are needed to edit and fix these problems This can produce furthe bugs unless exhaustive block verification is done.

Static Verification Revolution

Static verification technology eliminates the need for vectors and provides an exhaustive analysis that is guaranteed to be 100% correct. An additional benefit in not using vectors is much faster processing compared to vector simulation. This is a revolution in the analysis of electronic circuits. The static approach was first seen with timing analysis at the gate-level. This revolution has also been taken to the RTL level where formal verification tools also employ static verification technology. Formal tools can be divided into equivalence checking, model checking, and a new type called static functional verification. Solidify is a static functional verification tool.

Equivalence Checking. This compares two designs to see if they are equivalent. Pre- and Post synthesis designs can be compared, or whenever a gate-level netlist is edited or tweaked to resolve timing or other issues. Equivalence checking eliminates the need for gate-level simulation to confirm the RTL behavior is maintained. It does require a golden reference on which all equivalence checks are based, which is typically the RTL description.

Model Checking. This describes an algorithmic approach that an RTL or gate-level description satisfies a set of properties or behaviors. The designer writes properties that are based on a functional specification for the particular block being verified, e.g., does my FSM only transition to a legal state. Model checkers have existed for several years but suffer from a number of problems:

Capacity. Because of the model checking approach, typical capacity is only 1K-2K gates for complex properties, but today's designs require capacities at least 10X greater at the block-level.

Performance. Verification times are typically in minutes for each property, and with no estimate of how long each property will take to complete. A result may not be returned for hours or days. This prevents the model checker from being a tool the day-to-day designer can use to make quick progress in debugging the design.

Ease of Use. Writing properties requires learning a complex and arcane language. As a consequence, an "expert" is assigned to the use of the tool. This means the day-to-day designer does not have access to this technology.

User Interface. The user interface has been text based which does not make it easy to use.

Static Functional Verification

Static functional verification is similar to model checking in that it verifies that an RTL or gate-level description satisfies a set properties or behaviors. However it uses an entirely different algorithmic approach that provides a breakthrough in capacity, speed and ease-of-use. *Solidify* is the only static functional verification tool available today and has the following features:

Capacity. Blocks over 20K gates. Complex properties can be used for complete block verification

Performance. Verification times are typically in just a few seconds for a property.

Ease of Use. Easy Verilog-based property language that a designer can learn in a few hours.

User Interface. Full graphical user interface for use in day-to-day design and debug of RTL.

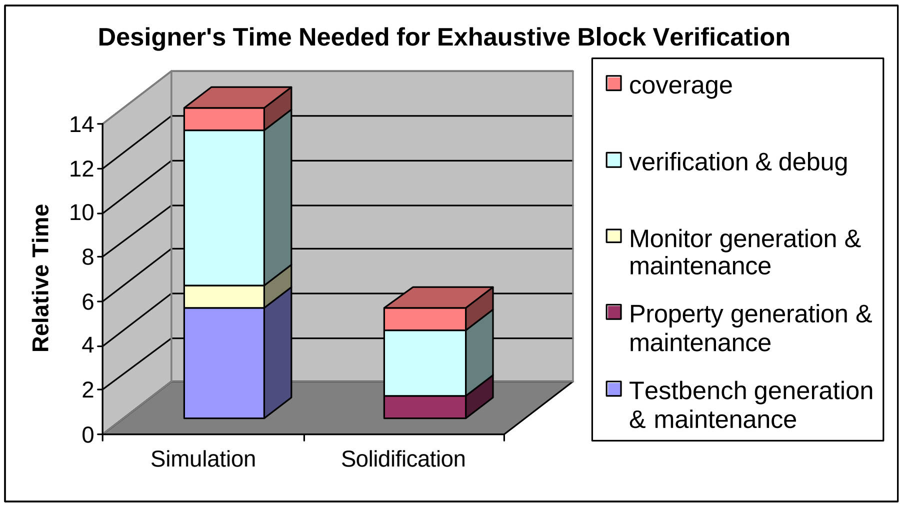
Coverage Analysis. Static coverage analysis reports the effectiveness of the properties ensuring that no part of the block was unverified.

Verifies Wide Class of Circuits. Verifies all kinds of circuits written in VHDL or Verilog including bus arbiters, cache controllers, sequencers, arithmetic blocks, datapaths, ALUs, reset circuits, bus interfaces, memories, instruction and address decoders, pipelines, transaction protocols, and state machines. Also handles dual phase clocks as well as blocks with multiple clocks.

Static verification technology makes verification faster and lowers effort since no vectors are used. Since its analysis is exhaustive and verified properties or behaviors are guaranteed 100% correct, static functional verification can be used to bridge the verification gap.

Functional Closure Achieved with Solidification

The key to achieving functional closure is to perform exhaustive block-level verification before integration, and re-verifying any block-level changes during integration-level testing. Solidification is a methodology that uses Solidify to satisfy both of these requirements. Solidify's speed, capacity and ease-of-use make exhaustive block-level verification practical and enables functional closure.



Solidification reduces the verification effort and the time it takes to achieve integration and improves quality:

Generation and maintenance of testbenches eliminated. This is typically a difficult chore that designers would gladly do without and is a large part of the verification effort.

Corner cases immediately revealed. One of the challenges of verification is coming up with all the corner cases that need to be verified. Solidify's static approach is exhaustive, and will uncover every possible exception to a property that is to be verified, including unusual corner cases.

Answers in seconds. Individual properties typically verify in just seconds so you can find more bugs in a day.

Low maintenance verification suite. No additional files, testbenches or scripts need to be maintained for verification. Re-running block verification is easy if any change is made to the HDL because of a system-level problem.

Tells you what you missed. Static coverage analysis assures that enough properties have been written to cover all of the design. It is an exhaustive analysis and avoids false coverage problems by uncovering conditions that might otherwise have been missed. It is also highly accurate and does not report false negatives, i.e., uncovered areas that are in fact covered. This is very important if designers are to be able to make quick progress with a tool. Generated coverage reports clearly point out areas of the design which require properties to be written. It also gives a metric of the progress made, so managers can judge the effort and progress to verify a block. Since coverage analysis is a separate step it has no impact on verification times, and an initial analysis runs in seconds.

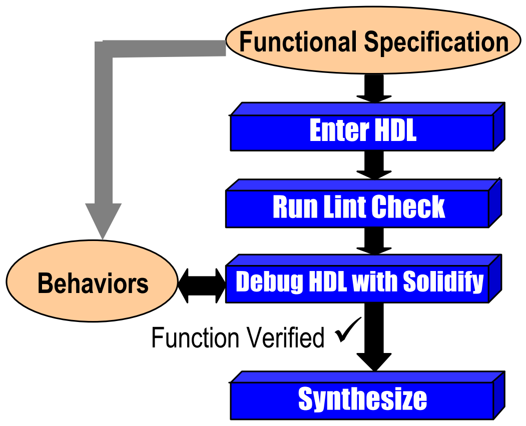
Captures design intent. The property language captures in a readable format the intent of the design as described in the functional specifications and interface behaviors. This becomes a kind of verifiable documentation that contributes to insight and understanding of the design. As the specifications are written as properties, then verified, inconsistencies become identified. This improves the quality of the design, and decreases design schedules as problems are identified early on.

Solidification enables functional closure at the chip or system-level by reducing the number of bugs before integration and supports easy re-verification of blocks if bug fixes are made. The reduction in verification effort and time is achieved without changing the style of the HDL that is written. Adopting Solidify is a low-risk strategy and designers quickly reap its benefits.

Solidification Flow

"Solidify's new static approach has demonstrated that it can shorten the time it takes to verify, and reduces our verification effort. The tool has been easy to adopt and we have not had to make any change to our synthesis flow. I feel Solidify will be an important addition to our verification toolset."

Jim Herndon, Staff Design Engineer, Compaq Computer



Using Solidify does not disturb the synthesis flow and requires no changes in the syntax and style of the HDL that is written. Designers should follow good design practice and use lint tools before debugging their blocks with Solidify. This removes obvious syntax and other design rule errors in the HDL. To confirm block behavior, Solidify applies properties entered by the designer to the block or sub-block being verified.

The properties verify the functional and test specifications. Properties are readable and have a Verilog-based syntax. The effort to write a property is similar to writing a monitor for a simulation test environment. Solidify supports quick interactive investigation and tuning of properties through its user interface, so designers can make quick progress.

After the designer has entered a set of properties, static coverage analysis is run to reveal what parts of the design are uncovered by the set of properties. Three levels of coverage are provided which range from a fast initial review of the design to an exhaustive analysis. Designers work with the first level until the block is completely covered, then move on to the second and third levels which take more computation time. This ensures the designer is working to improve the design and not waiting unnecessarily for the analysis to complete. Incremental coverage analysis furthers enhances designer productivity and reduces verification effort.

Once verification has been completed for a block, it is ready for synthesis and integration with the other blocks in the design.

Return on Investment (ROI)

Given that over half the cost of creating a design is in verification, and three-fourths of that cost is in the time of engineering staff, reducing verification effort and time is essential. Solidification lowers the cost of creating working IC designs and quickly gives a worthwhile return on investment. It shortens schedules, reduces verification costs, catches bugs earlier, supports IP reuse and lowers CAD support costs.

Solidify shortens schedules by reducing the number of bugs found and fixed at chip or system-level integration. Iterations due to functional non-convergence, discussed earlier, are reduced or eliminated using Solidify's exhaustive block-level verification. The schedule also contracts because testbench generation, simulation with code coverage, and iterations during integration are eliminated. This reduced verification schedule translates into an early entry into the marketplace, which can bring faster and larger financial rewards. Increased efficiency in the engineering staff is also realized since fewer resources are needed to verify a design.

Solidify reduces verification costs by reducing the time needed by engineering staff and by reducing the simulation budget. To get the number of simulation cycles equivalent to Solidify's static approach, a huge investment in CAD software would be needed. From the CAD manager's perspective, because simulation, testbench, and coverage tools are not needed for the block level, the need for additional licenses for these tools is reduced. From the design manager's perspective, the maintenance of complex verification suites with their individual scripts, programs, and custom testbenches is eliminated as well, reducing engineering effort.

The earlier a bug is found the lower the cost of repair for a project. This is particularly true for difficult bugs and corner cases. If a critical bug is missed until after place & route, what will be the cost? It will be expensive since most of the design cycle is repeated to verify any fix that is made. Accounting for the time and effort to re-verify, a conservative analysis would be over $50K for a typical ASIC. If a bug is found on a hardware emulator or physical prototype, it is the most expensive to fix. After tapeout, the cost is much higher. A widely publicized example is the Intel Pentium divider bug which cost Intel hundreds of millions of dollars.

Using Solidify for block verification promotes reuse in future designs with future development savings. Blocks are exhaustively verified and are more robust for distribution. The properties themselves make a readable documents which show what was verified and how. This means an IP block can be easily delivered to another design team which demonstrates the behavior and intent of the design with a minimum set of files. Finally both the property set and HDL can be re-used and re-verified easily, including any refinements needed. This is in contrast to simulation verification suites that have a higher maintenance burden.

Integrated Design Environment Makes it Easy

Solidify is designed to be an RTL Integrated Development Environment (RTL-IDE™) that organizes the creation, edit and debug of RTL code to be as efficient as possible. Similar in style to modern software development environments, Solidify allows engineers to make quick progress finding and fixing bugs. It integrates several different tools, which work seamlessly together. Compilation, verification and coverage analysis are combined in a single tool so that everything needed for development is provided. No file translations are required, and Solidify accepts both VHDL and Verilog designs.

The user interface (UI) is written in JAVA and provides a consistent look-and-feel across UNIX and NT workstations. The UI has a number of components or windows. The *results viewer* which exposes all signals that violate an expected behavior or property, provides just the data needed to understand the error. The *design browser* organizes all the HDL and design files so projects with many files and sub-blocks are handled smoothly. The *HDL editor* includes syntax color-coding to make edits easier.

Solidify supports remote computation on either NT or UNIX platforms across a network. Many engineers find NT platforms offer price vs. performance benefits, but still the majority of their design work remains under UNIX. Solidify conveniently supports remote verification on a low-cost NT platform for UNIX users, and delivers a high-level of verification performance.

Solidify on NT supports both remote UNIX verification, and remote invocation of RTL design tools such as Design Compiler from Synopsys. This means that NT-based designers can take advantage of other existing design tools on their UNIX network. Users benefit from improved utilization of existing CAD software and are not forced to buy unnecessary software licenses.

Summary

The growing complexity in ASICs and programmable parts means functional verification will continue to be the nightmare which keeps projects managers up at night unless a new methodology is used. Staying with the current simulation based approach will mean more schedule slips and re-spins after tapeout.

The key to bridge the verification gap and achieve functional closure is to perform Solidification—exhaustive block-level verification. Solidify, the CAE industry's first static functional verification tool, brings a new level of productivity to every-day designers, architects, and validation engineers. It enables companies to verify what they can design, have predictable schedules, and deliver high-quality products to their customers.

Solidify is a practical design tool with a low-risk of adoption. It does not disturb the synthesis design flow, and a one-day training is all that is needed to start using the tool. It speed, capacity, and ease-of-use quickly makes it an indispensable design tool.

To find out more please contact:

averant

1257-A Oakmead Parkway
Sunnyvale CA 94086
Office 408-774-0386
Fax 408-774-1121
Web www.averant.com