

Functional Review using SpringSoft Verdi Automatic Debug and NextOp BugScope Assertion Synthesis

Yunshan Zhu
President and CEO
NextOp Software

Thomas Li
Director of Product Marketing
SpringSoft

Functional Review Overview

It is well-accepted that code review is desirable to ensure the quality of RTL designs. However, from a management perspective, the lack of specific targets beyond coding guidelines often makes it difficult to gain uniform participation, and measure results. Furthermore, traditional code review is incomplete in that it only considers the static RTL, rather than examining it in the full context of how the code is exercised in the test environment.

A ‘functional review’ goes beyond a code review to combine an RTL design and functional properties based on its tests; in a functional review a designer can examine the RTL in the context of how it has been exercised. This paper will show how two technologies, assertion synthesis and debug automation, can be linked to allow designers to achieve an efficient functional review by providing a target for them to analyze and capture their design intent.

Assertion Synthesis and Automated Debug

Assertion synthesis that takes an RTL design and its simulation runs as input and generates assertion and coverage properties in standard assertion languages such as SVA and PSL. The properties are guaranteed to hold for all of the input simulation runs, and can capture the behaviors of RTL design signals across entire test suites.

Assertion synthesis enhances existing simulation, formal and emulation verification flows by helping design and verification engineers to uncover corner-case bugs, expose functional

coverage holes, and increase verification observability. Assertion synthesis allows design and verification teams to implement assertion-based verification in a timely and resource efficient manner, mitigating the risk of project delays and defective silicon.

An automatic debug system provides a platform with an integrated design knowledge database and design behavior visualization interface that speed up the process for designers to comprehend their design and verification results through automatic design correlating and tracing technologies. The underlying database that stores the verification results provides the crucial link between the verification tools and debug environment.

An automatic debug system allows a designer to efficiently traverse RTL and have a “local” view RTL signal values at any particular time in simulation. Assertion synthesis generates properties that capture a “global” view of the signals across all simulation cycles. Combined with assertion synthesis, an advanced automatic debug system offers designers both the global view and the local view of RTL signal behaviors and enables effective functional review and comprehension.

BugScope Assertion Synthesis and Verdi Automated Debug System Functional Review Methodology

The methodology below combines NextOp’s BugScope Assertion Synthesis with SpringSoft’s Verdi Automated Debug System to perform an effective functional review of the design.

1. Run BugScope to generate NextOp properties for RTL modules under review.

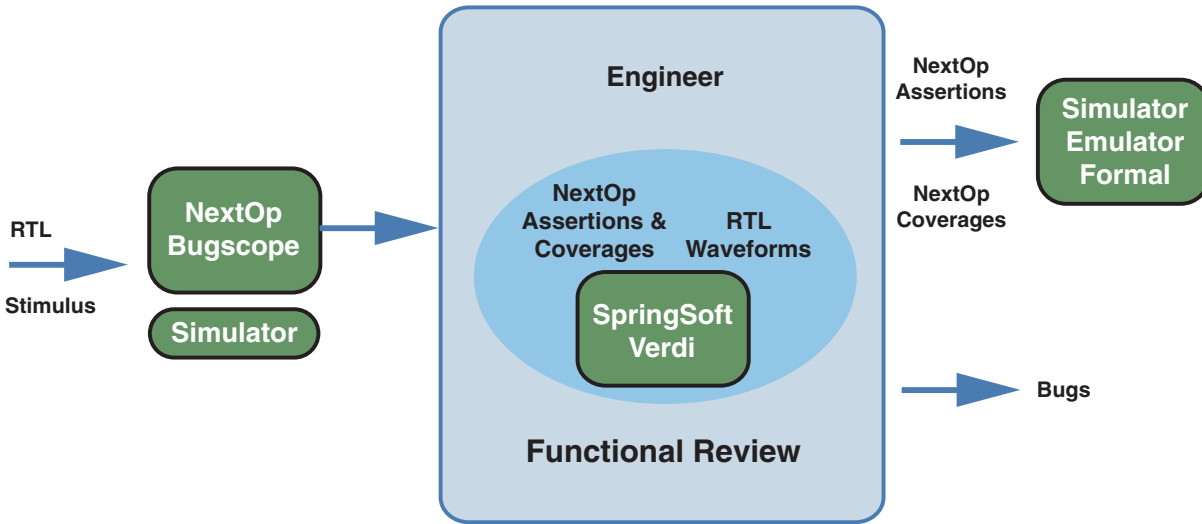


Fig. 1: Functional Review Flow diagram using BugScope and Verdi

2. Load NextOp properties, RTL, and waveforms into the Verdi environment.
3. Use NextOp properties as targets to drive functional review. Each NextOp property should be classified as an assertion or a functional coverage property. When reviewing each NextOp property, use the Verdi source window to cross link signals in a property with their corresponding RTL. Use the Verdi waveform window to examine the dynamic values of the signals if needed. A bug in the RTL or testbench is often exposed in the process.
4. Bind the classified assertions with RTL for additional verification; leverage coverage properties to guide additional test generation; and fix the bugs discovered during the review.

The above functional review process will produce three set of deliverables: assertions, coverage properties and bugs. The assertions capture designer's intent and the coverage properties capture deficiencies in the testbench that should be communicated to the verification engineers. The following section highlights various types of RTL bugs that have been found during functional review.

Case Studies: Bugs Identified from Assertion Synthesis property review

Example 1: RTL code contains incorrect logic that is difficult to see

Code sample:

```

always @(posedge clk..) begin
  if(load_cnt)
    cnt_incr <= 1'b1;
  else if(cnt == 71)
    cnt_incr <= 1'b0;
  if(cnt_incr) cnt <= cnt+1;
end
  
```

NextOp property:

```

cnt == id72 |-> @cnt != id73;
  
```

Discussion:

The property is an assertion. The designer's intent is that whenever cnt is 72, in the next cycle, cnt should be reset to zero and therefore should never be 73.

When reviewing the property along with its RTL, the designer realizes that cnt is incremented whenever load_cnt is asserted, and potential overflow could occur. There are two errors in the RTL:

- The priority to load_cnt was incorrect.
- The code to latch load_cnt when cnt == 71 was missing to handle the scenario correctly.

The following diagram illustrates the buggy scenario. The property offers the designer an orthogonal view of the RTL and enables him to recognize the bug.

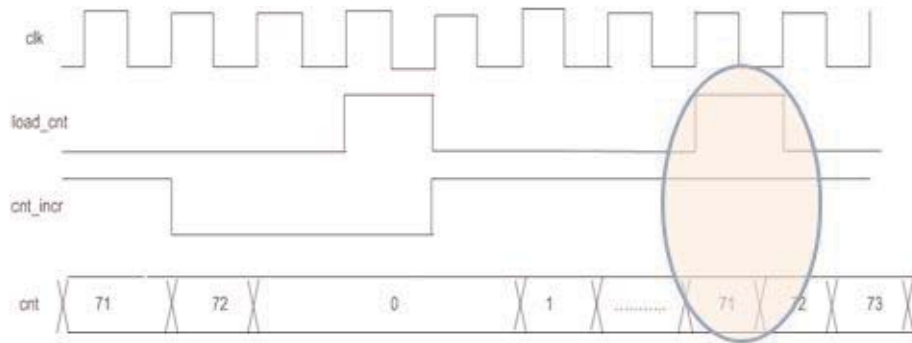


Fig. 2: Timing Diagram to illustrate bug scenario

than max_cell_size due to wrap-around bit. The property holds only for Mode 1. The other modes have not been tested.

When the designer traces the fanout of max_cell_size, a bug is found. The following RTL code shows that the RTL for delta is missing logic to handle the wrap-around bit of

Example 2: RTL code is missing logic for certain cases

NextOp property:

```
max_cell_size >= start_cell_size
```

Discussion:

When reviewing this property, the designer points out that the start_cell_size is encoded based on four different modes.

- Mode1 : start_cell_size[15:0] is the true cell size
- Mode2 : start_cell_size[7:0] is the true cell size while start_cell_size[8] denotes wrap-around
- Mode3 : start_cell_size[3:0] is the true cell size while start_cell_size[4] denotes wrap-around
- Mode4 : start_cell_size[1:0] is the true cell size while start_cell_size[2] denotes wrap-around

In Mode 2,3 and 4, start_cell_size can be bigger

start_cell_size. The buggy code would have caused the value of delta to underflow.

Code sample:

```
wire [15:0] delta = clear ? 16'h0 : (!sop ?
r_delta : max_cell_size ñ start_cell_size);
```

Example 3: RTL code is buggy due to the interaction of multiple modules

NextOp property:

```
start |-> r_state != RUN
```

Discussion:

In this example, BugScope helped the designer to catch a bug where two different modules have differing assumptions on the communication protocol.

The state machine included 4 states: IDLE-> RUN-> WAIT-> PAYLOAD. The signal start comes from another block, initiates the transaction, and could only be asserted at IDLE state according to the design assumption.

When reviewing the properties, the designer questions why a stronger assertion `tstate[2:0] == IDLE` is not generated. By cross linking with the waveform, the designer and verification engineer find out that the start signal was wrongly asserted by a neighboring block when `r_state == PAYLOAD`. The state machine has not been design to handle this case and start was ignored. As a consequence, certain transactions will be lost by the buggy design.

Key Elements of BugScope/Verdi Functional Review Process

The key features of BugScope and the Verdi system that enable an effective function review are outlined below.

1. NextOp properties are distinct from one another and provide an orthogonal view to the original RTL by incorporating test information in addition to the static RTL.
2. NextOp properties are concise and easy to understand. By using internal RTL signals, NextOp properties capture complex functionalities without having to use complex assertion language expressions. The functional review is therefore focused on comprehending user's RTL code and not on the syntax of generated properties.
3. The Verdi system allows designers to click on a NextOp property and immediately cross probe to relevant part of the RTL design. Designers can understand the

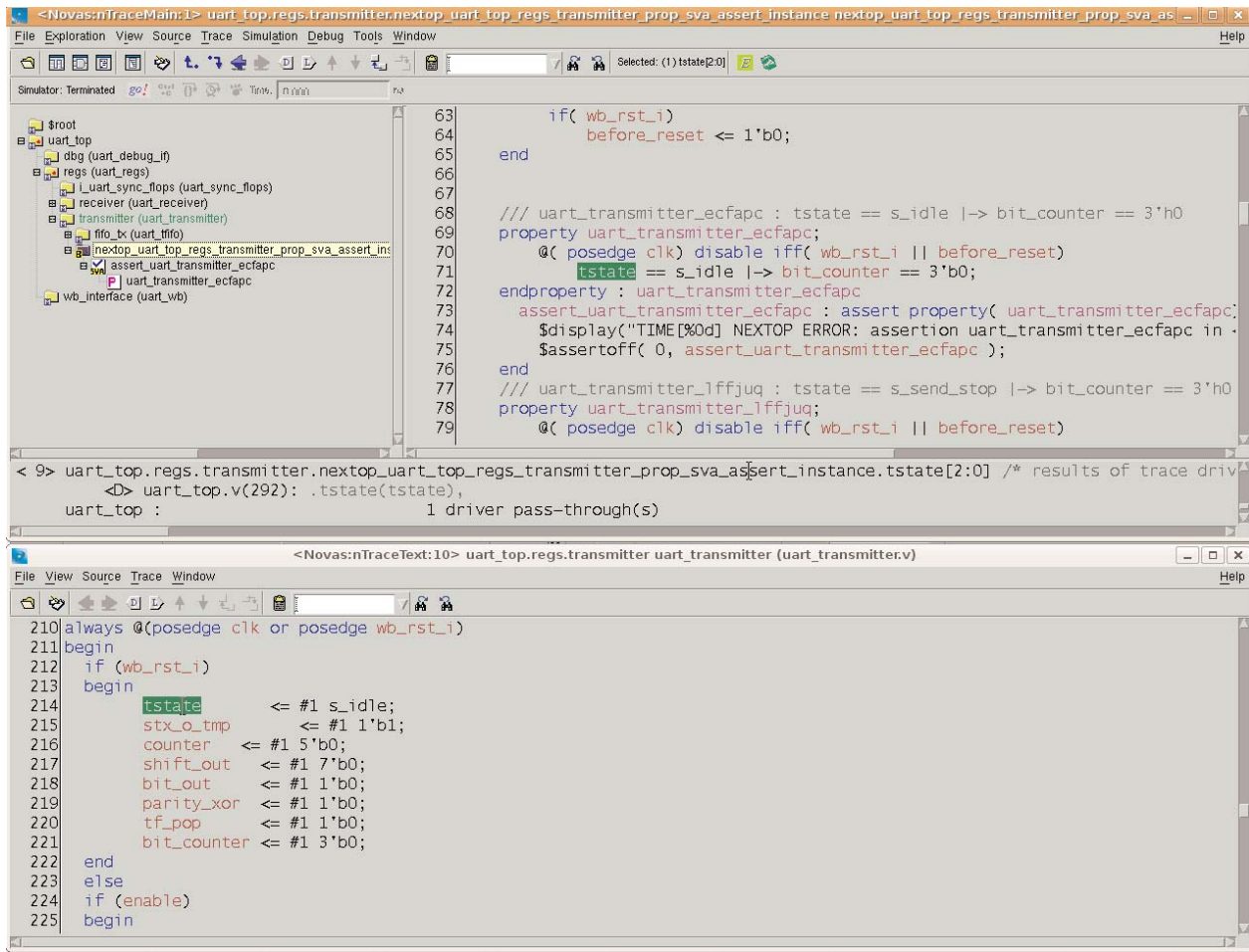


Fig. 3: BugScope and the Verdi System cross-couple assertion and coverage properties with RTL code.

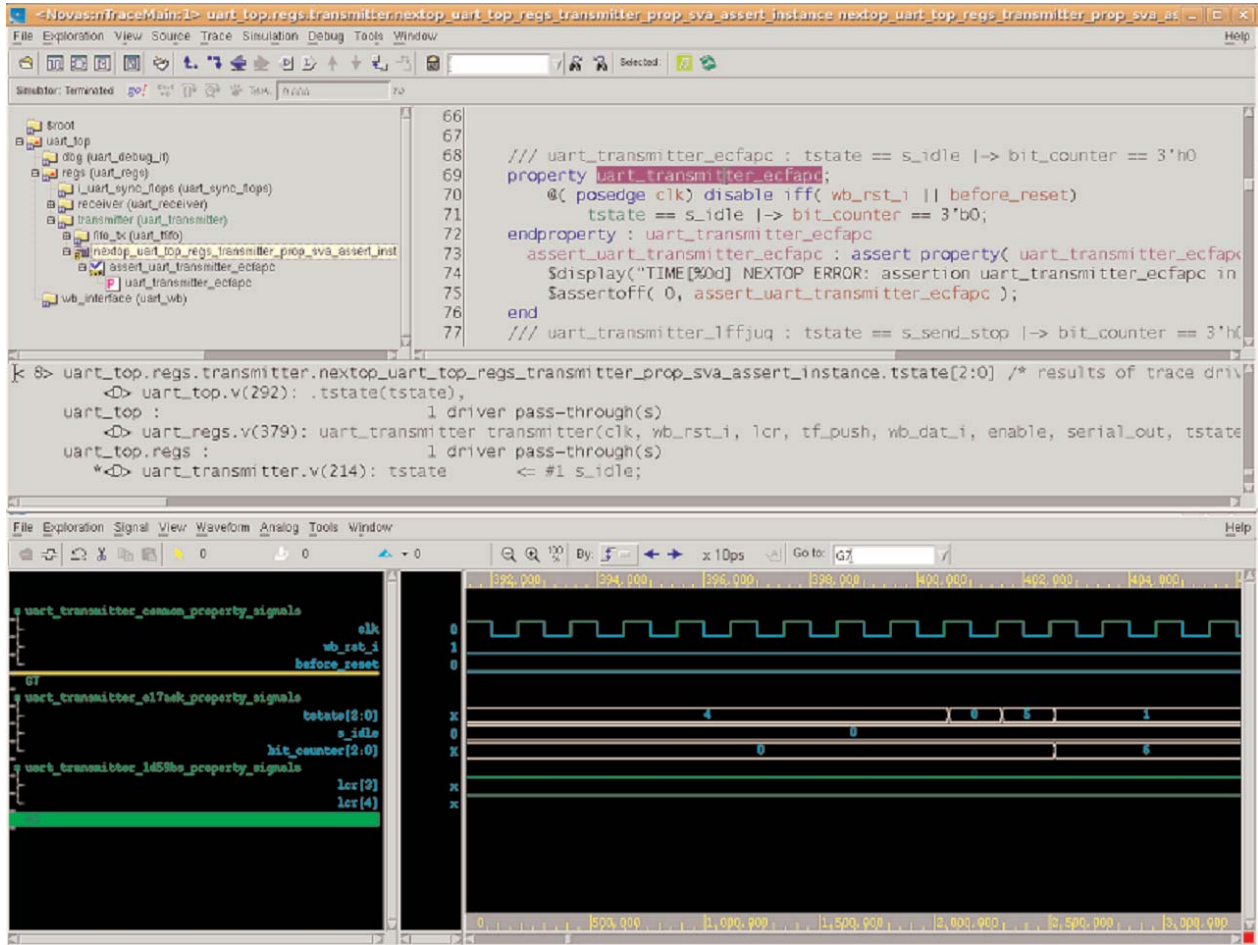


Fig. 4: BugScope & Verdi system cross-couple assertion and coverage properties with waveforms

properties easily by highlighting the BugScope property and automatically tracing back to the appropriate signal in the design to make their assessment. They can display a window with NextOp properties and drag the NextOp property to the Verdi RTL source code window. The Verdi system will automatically locate where the signal has been defined within the RTL.

4. As shown in Figure 2, BugScope and the Verdi system are linked to also provide the design and verification teams with cross-coupled views of the RTL code and waveform. All three elements, properties, RTL code and waveform can also be displayed in one window (not shown).

5. To reduce the learning curve associated with assertion languages such as System Verilog Assertions (SVA) and Property Specification Language (PSL), the simple, Verilog-like NextOp language is added as a comment to the SVA.
6. The Verdi system's assertion-based debug capabilities facilitate quick traversal from assertion failure to the related design activity. The Verdi system extracts, isolates, and displays pertinent logic in flexible design views, showing the inter-relationships between the design, assertions, and test-bench. The Verdi system's assertion analyzer gives the details of the assertion evaluation history for designers to quickly understand how the assertion has been exercised.

7. The Verdi system's various and tightly synchronized design views (including source code, schematic, waveform...etc.) provide intuitive design traversal capabilities for designers to quickly understand their design behavior.

In conclusion, NextOp BugScope Assertion Synthesis and SpringSoft's Verdi Automated Debug System, when used together, create a measurable, efficient target for design organizations to use for functional review. Designers utilizing BugScope assertion synthesis as part of their Assertion-Based Verification process can classify the properties as assertions and coverage holes during this same review, enabling them to simultaneously identify coverage holes and provide triggers for their simulation, formal or emulation verification process.

Authors:

Dr. Yunshan Zhu President and CEO NextOp Software

Yunshan Zhu co-founded NextOp Software and has led the company through the development, delivery and production implementation of its

flagship assertion synthesis product by multiple semiconductor companies. Prior to NextOp, Dr. Zhu was a member of the Advanced Technology Group at Synopsys. Dr. Zhu also worked as a visiting scientist and a post-doc at Carnegie Mellon University where he co-invented the bounded model checking algorithm. Dr. Zhu did his undergraduate study at University of Science and Technology of China and received his Ph.D. in Computer Science from University of North Carolina at Chapel Hill.

Thomas Li Director of Product Marketing, SpringSoft

Thomas Li is currently the Product Marketing Director for SpringSoft. He has over 15 years of EDA industry experience including the positions as an application engineer, design services consultant and technical marketing manager. Before joining SpringSoft, he worked with Synopsys and Mentor Graphics.