2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Efficient Formal strategies to verify the robustness of the design

Sakthivel Ramaiah
Cadence Design Systems (India) Pvt Ltd

***Abstract*- Any design architecture must be able to correctly handle the order of priority and race conditions which occur due to the asynchronous nature of input, which could be in terms of signals or packets. The standard may not specify all nuances of the race conditions and priority, and it is imperative to have a detailed implementation specification. There are possibilities in the standard, which are open for interpretation as the specification evolves. Formal verification is excellent at highlighting such situations provided the approach is systematic.**

This paper demonstrates the methodology to strategize SV assertions.

Keywords - Formal Env, AUX code, SV Assertions, Jasper™ FPV App

I. INTRODUCTION

This paper recommends two unique *"positive-negative" and "effect without cause"* strategies that were adopted for qualifying a block in Cadence CXL Controller IP. These strategies exhibit the art of categorizing formal check assertions and showcase the systematic approach required to qualify design comprehensively. These strategies also help bring clarity in the implementation specification to avoid inconsistencies.

II. PROPOSED STRATEGY

A. *Strategy 1: Positive-Negative*

For each feature, both positive and negative aspects are analyzed. For example, if specification states that:

"*When a Request packet has been sent, receipt of any packet other than a Status packet is considered an unexpected packet*."

The all-round analysis has forced us to ask the following questions and to develop additional checks for this feature:

- What happens if an unexpected packet is received after a request?
- What happens if a status packet is received without any request?

In general, let us consider a signal "x" and a packet "y":

- "x" should be asserted, or "y" should be sent for "p" number of reasons
- "x" should not be asserted, or "y" should not be sent for "n" number of reasons

Where "p" refers to positive checks and transforms to "p" number of assertions, and
where "n" refers to negative checks and transforms to "n" number of assertions.
The formal environment auxiliary (AUX) code is written to identify "p" reasons and used to disable "n" assertions.

B. *Strategy 2: Effect Without Cause*

In general, for cause-effect feature checks, inputs trigger an assertion to check an expected output. There is, however, a disadvantage in having only this check.

For example, what happens if there is an effect without any cause? In order to ensure there are no unexpected changes in outputs or state transitions, each "cause and effect" type check has an accompanying "when there's an effect, check there was a valid cause".

For example, “when active request packet is received in low-power state, DUT goes to retrain state.”

There are two kinds of checks applied for this condition in order to make the verification foolproof:

- *Assertion 1 (cause-effect check)*
  Pre-condition: DUT receiving Active Request packet in low-power state
  Check: DUT should reach retrain state
- *Assertion 2 (effect without cause check)*
  Pre-condition: DUT transitioning to retrain state from low-power state
  Check: DUT should have received an Active Request packet

Note: AUX code support would be required for “Assertion 2”

III. IMPLEMENTATION

The Formal Environment contains a top SV module where DUT and Checker modules are instantiated. The Checker module has Assumptions (Constraints) and Assertions (Checks) supported by AUX code which mimics DUT features in terms of FSM model.

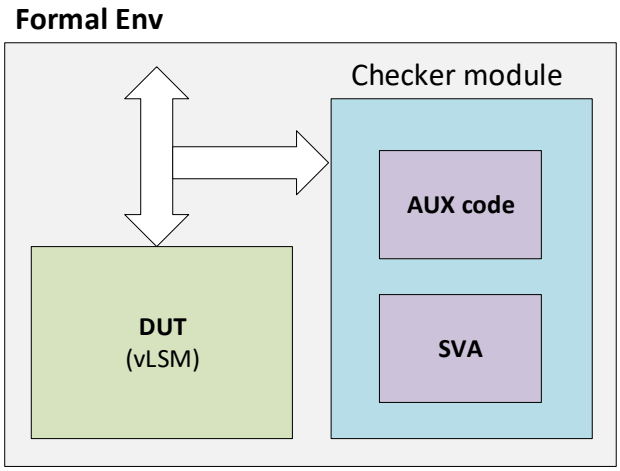

Figure 1. Formal Environment block level diagram

IV. RESULTS

Deploying an independent formal verification engineer to formally verify the Arb/Mux block of the CXL Controller identified several grey areas of the CXL specification that required clarification. At Cadence, these Arb/Mux discussions took place within a small group of engineers, namely the block designer, the formal engineer, the technical lead, and the CXL IP architect. The controllability and quick turnaround time at the Arb/Mux formal level allowed for rapid development and improvement of the Arb/Mux IP block, while the complex UVM top-level testbench was developed in parallel. Around 70+ bugs have been found in the CXL ArbMux Block using this approach. Figure 2 summarizes some key statistics from this exercise executed using Cadence® Jasper™ Formal Property Verification (FPV) App.

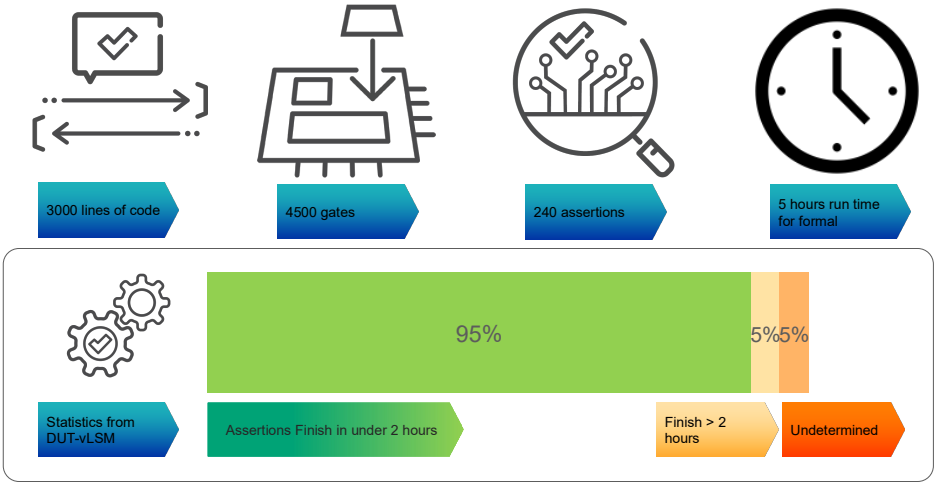

Figure 2. Highlights of Formal Verification of the CXL ArbMux-vLSM RTL Block

V. APPLICATIONS

Formal checks implemented using these strategies helped find bugs in Cadence CXL Controller IP that typical formal techniques would not. These techniques paved the way to highlight the grey areas in the standard specification as well as in the design implementation specification.

Adopting these strategies provided a structured thought process in analyzing design or implementation specification grey areas.

What happens if the DUT encounters any of these circumstances?

- Receives valid packets in the wrong state
- Receives valid packet but not applicable to a particular mode
- Receives back-to-back request packets or status packets - redundant
- Receives multiple requests at the same time across different interfaces
- Receives packet with reserved values in valid field or non-zero values in reserved fields

What happens if the DUT?

- sends valid packet but in the wrong state or wrong mode
- sends back-to-back request or status packets - redundant

These questions and checks paved the way to have deterministic and consistent behavior of the RTL-DUT for noisy inputs and hence making the product robust.

VI. SUMMARY

The adoption of formal verification for critical sub-modules of the IP is particularly important in newly released standards where the specification is raw and prone to different reader interpretations, coupled with the fact that the independently developed VIP is being developed in parallel to the Design IP and will take time to reach maturity for top-level testing. Approximately 10% of the total bugs found during PCIe and CXL verification were via formal verification. The comprehensive approach described above of having both positive and negative checks in the Formal Environment has yielded a considerable number of bugs to be reported.

REFERENCES

[1] https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/ip/design-ip/pcie-formal-verification-wp.pdf