2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Novel Use of Symbolic Map Based Constraints for Synchronization Block Verification Using Formal Methods

Vikas Vikram Singh, Sanjana Jain, Patnala Naga Sai, Ipshita Tripathi
Qualcomm India Pvt. Ltd.

***Abstract*- Formal Verification provides powerful ways of finding corner case scenarios in many design types. This is due to the inherent nature of formal verification where the randomness of the inputs to the DUT help uncover scenarios where the design can fail based on the checks written. While this strength is best utilized by lightly constraining the inputs to see what the formal tools can uncover, they can pose a challenge when the design under test expects the inputs to be ordered across interfaces in a strict order. In this paper we talk about how we tackled this challenge using a novel method of using symbolic variables in a multi-dimensional map structure.**

Keywords- formal verification, symbolic variables, formal bug-hunting, formal model performance

I. INTRODUCTION

Formal verification is increasingly used in the verification flow to uncover corner case bugs over the last decade and a half. Formal verification is often used due to its ability to exhaustively verify the design-under-test. Based on state space search algorithms formal verification tools can cover all state transitions and can exhaustively prove the correctness of designs by finding complex corner case scenarios which are often very difficult to cover with simulation [3]. These scenarios are often due to the temporal interplay of multiple input ports covering a large set of possible input scenarios. Mostly in formal models, inputs are lightly constrained to allow the randomness of inputs to interplay with each other and allows the tools to guide us to the scenarios under which the design will malfunction based on the checks we have written.

Lightly constrained setups enable us to expose corner scenarios easily [6] and thus most formal engineers try to only allow essential constraints and actively ensure that randomness is maintained across interfaces. However, in our verification effort which we discuss here, we are faced with the challenge that involves strict ordering among the interfaces to get the design-under-test to function properly. This poses a challenge in that now the formal testbench has to constrain the inter-interface traffic to ensure that ordering is maintained while also ensuring that the temporal randomness is held intact so that the corner-scenarios due to the temporal distribution of inputs do not get filtered out. This required us to come up with many possible ways of constraining such that this balance is maintained while also ensuring that the testbench performance is not impacted by the implementation.

Formal testbench performance and the convergence and depths achieved by the checks are always kept in mind while developing a formal model. As part of using the bounded proof methodology [7] we expect that the checks will cross the required proof depth in order to be able to sign off the design. For that to be seen, the formal model needs to be light in implementation as large logic structures will end up affecting the convergence of the checks and cause them to not cross the desired proof depth. In that case, even if we allow for randomness of the inputs the verification will not be holistic and thus this aspect of the performance also needs to be considered in planning stage only.

Thus in order to correctly apply the formal methods to effectively stress the design, we need to consider the ordering requests within pipe, temporal randomness, input type completeness as well as testbench implementation complexity to come up with a successful strategy which will enable us to cross the desired proof depth. To understand the complexity involved in the constraint modelling let us understand the ordering requirements of the design first.

II. DESIGN REQUIREMENTS OF ORDERING

For our Design-under-test, the design had multiple instances which required ordering in the sequence of inputs received. The design is required to ensure that no deadlock is seen when all the inputs are received in order and also is required to ensure that no input which has been received by the design-under-test is dropped by the design.

The total number of types of inputs received by the design-under-test is N. The design-under-test has M streams or pipes of data flowing through it. The ordering requirement is a strict requirement within a given pipe but it doesn't need to be maintained among the pipes. There are D number of outstanding input types that the design can process within it. Thus for ordering the inputs using constraints, we would need to maintain strict ordering among all the interfaces within a given pipe, but not across pipes. Let us attempt to capture this information through a diagram as represented in Figure 1.

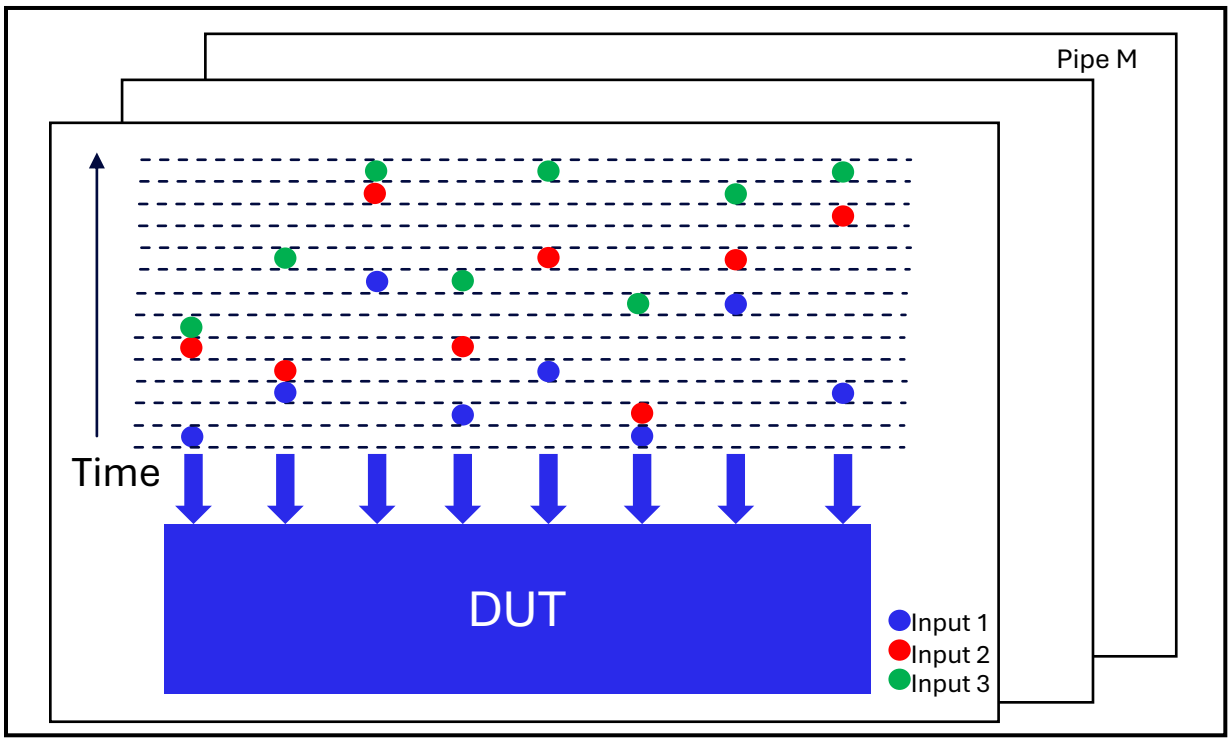

Figure 1. Ordering requirements of the design-under-test

As seen in the above diagram, the input 1 needs to be seen in all the interfaces of the DUT before input 2 is seen. There is no temporal restrictions on the time at which this input can be seen and formal testbench is expected to exercise random delay values across the interfaces to thoroughly test the design-under-test. We also need to maintain the same ordering requirements for each of the M pipes. Across the pipes we are expected to keep the inputs random. If these ordering requirements are not met, then legal hang scenarios are expected to be seen.

II. TESTBENCH IMPLEMENTATION

Since design-under-test is very sensitive to the ordering and to avoid debugging multiple false legal hang scenarios we needed to consider the ordering during test planning phase itself. The main checks of interest for the design-under-test were related to checking for false hang scenarios where the design-under-test would deadlock illegally, that is apart from the legal scenarios where it is expected to hang if input ordering is violated.

A. DIFFERENT APPROACHES FOR CONSTRAINING THE INTERFACES – PROS AND CONS

Different approaches were considered and rejected for constraining the interface before we arrived at our given methodology. Below we will discuss in brief the various approaches possible and why they were rejected.

A.1 FIFO Based Approach

A FIFO based approach would involve having a randomly filled FIFO with legal input values which would be popped when the input is to be sent to the various interfaces. There are a number of drawbacks of using this approach in our testbench.

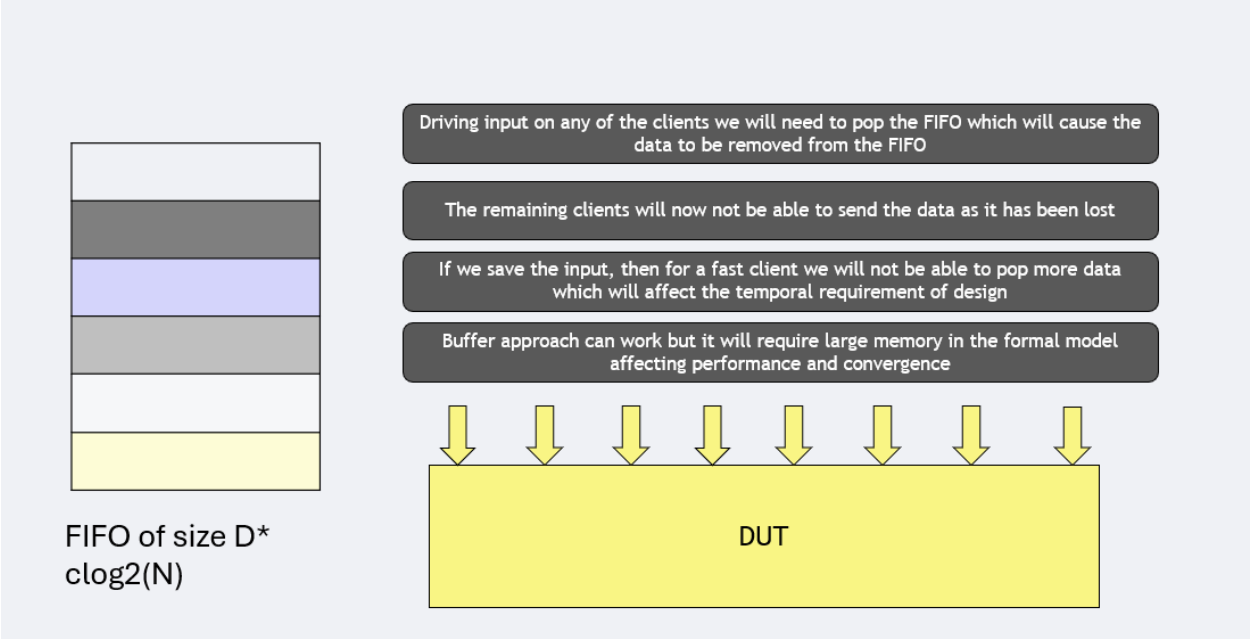

Figure 2. FIFO Based Approach and its drawbacks

When we pop the data from the FIFO the data will cease to exist in the FIFO thereafter. Thus if we attempt to introduce temporal randomness to the various interfaces, this will not be possible once the FIFO is popped. The only way all the interfaces can see the same data would then be to either send it at the same time to all the interfaces or to store in local storage till all the interfaces have been served. If this local storage is a shared resource this will stall all later inputs till all interfaces have been sent the previous data again affecting the temporal randomness of the inputs seen. If each interface had an identical filled fifo which was popped per interface, it would be very poor on performance as it would introduce severe complexity to the formal testbench and degrade the checker depths and hence verification quality. Thus the FIFO based approach was considered but discarded

A.2. BUFFER BASED APPROACH

If instead of a FIFO based approach we go for a buffer based approach, we will improve the performance of the checks slightly by being able to reuse the buffer to serve N^D input combinations using randomly filled buffers with legal input combinations. However, it will still entail a use of M large memory arrays of N*D size which will greatly impact performance of the checks still. Thus we didn't go for a buffer based approach as well though it was slightly better in performance than the FIFO approach.

A.3 COLORING APPROACH

A coloring based approach was not feasible as there were M parallel pipes and each input value had to be preserved to the used by the design-under-test. Thus the input value could not be constrained or "colored" in different patterns as then the input value would be lost and design-under-test would not be able to process the request in that case.

A.4 SYMBOLIC VARIABLE APPROACH

Use of symbolic variable was next considered[1][2]. Symbolic variables allow for checking a range of random values, which in our case would comprise the legal input values. In a given formal run these values will remain constant and allow the formal tools to explore the wide range of legal values while holding them stable for a given formal run. However, in our case a single symbolic variable would not solve the problem as we also needed to check for a sequence of legal inputs and the symbolic variable would remain stable for a given run.

B. SYMBOLIC MAP APPROACH

While symbolic variable would have worked well if there was only one input to be driven across the different interfaces with temporal randomness, it fell short when it had to send a sequence of different inputs with D number of outstanding input values that could be processed.

To be able to cater to this we used a 2-D "symbolic map" initially. A symbolic map comprised of a 2-D array of symbolic variables where each element in the array was a symbolic variable, and the different elements contained random symbolic variables. Glue logic was added in the testbench to allow the reading of this "symbolic map". Thus, using this map, for a given pipe, we were able to create a sequence of legal input values with fully random temporal arrival. The glue logic added was such that it allowed for completely random arrival of input to the interfaces.

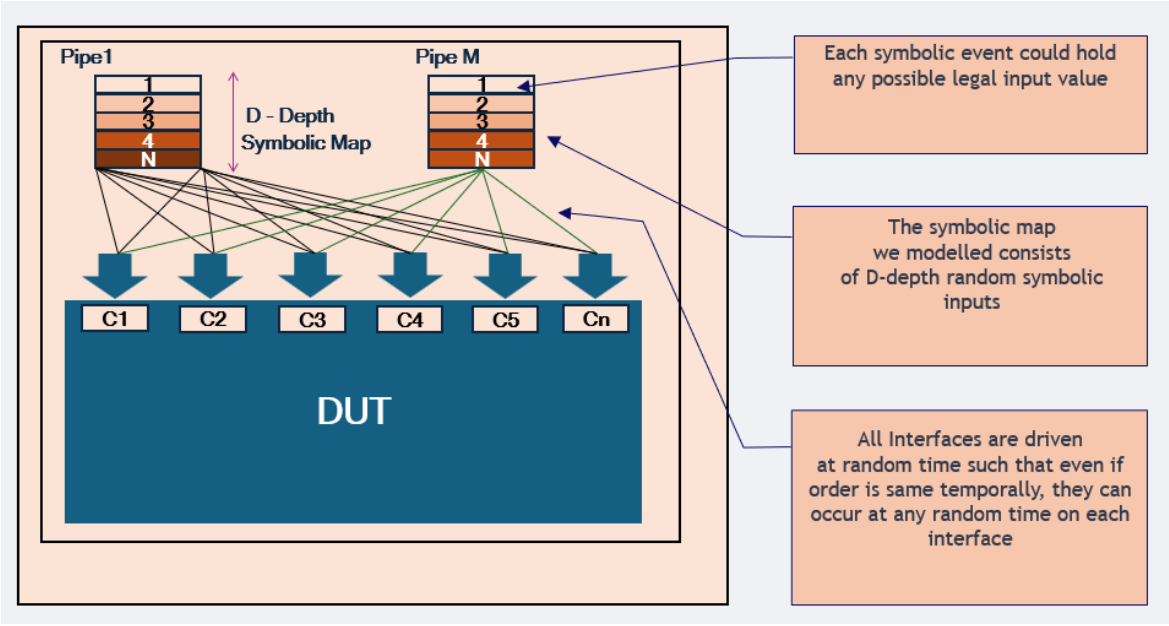

Figure 3. Symbolic map driving the interface of design-under-test

To allow for randomness across multiple M pipes while maintaining ordering within the pipe, we used an N dimensional "symbolic map" eventually to be able to exhaustively cover all the cases as seen below.

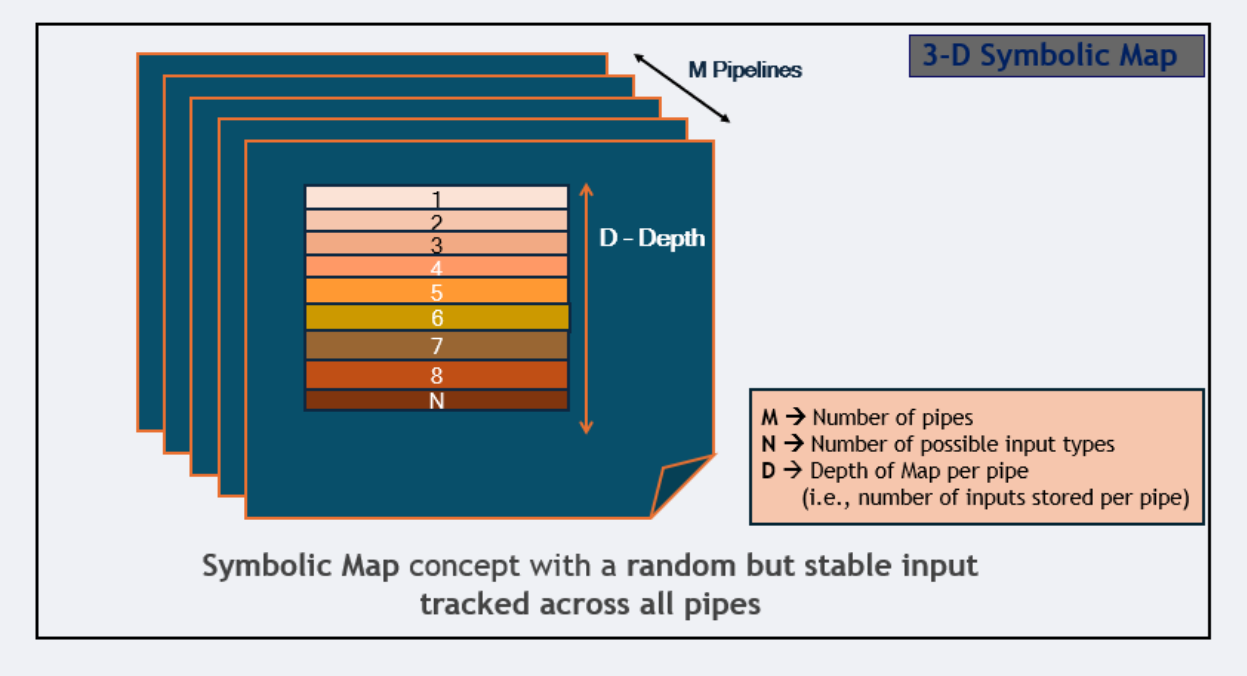

Figure 4. M dimensional symbolic map

This approach allowed us to check for N^{DM} possible combinations of sequences with complete temporal randomness across the interfaces.

III. RESULTS & CONCLUSION

Using this approach, we were able to completely order the inputs within the pipe and randomize across the pipes for N types of inputs with D outstanding inputs with complete temporal randomness across the interfaces. The use of symbolic map was very good on performance as very few flops were added due to this addition mainly in the glue logic. Using this approach all our checks were able to cross the desired required proof depth and 2 issues were found in the design-under-test. These issues were very corner scenarios mainly due to temporal arrival of inputs and sequencing of inputs seen thus giving good return for the creation of the formal model.

IV. ACKNOWLEDGMENTS

We would like thank Hu Yi (Shining) for providing detailed understanding of the design requirement.

REFERENCES

[1] Souri, A., Rahmani, A.M., Navimipour, N.J. *et al.* A symbolic model checking approach in formal verification of distributed systems. *Hum. Cent. Comput. Inf. Sci.* **9**, 4 (2019)

[2] Pandey, Manish and Raimi, Richard and Bryant, Randal E. and Abadir, Magdy S. "Formal verification of content addressable memories using symbolic trajectory evaluation" Proceedings of the 34th Annual Design Automation Conference, pp. 167 - 172

[3] I.Tripathi, A. Saxena, A. Verma, P. Aggarwal, "The Process and Proof for Formal Sign-off: A Live Case Study", DVCon 2016.

[4] P. Aggarwal, D. Chu, V. Kadamby, and V. Singhal, “Planning for end-to-end formal with simulation-based coverage”, Proc. Formal Methods in Computer-Aided Design FMCAD 2011, Austin, TX, USA, 2011, pp. 9-16.
[5] P. Wolper, “Expressing interesting properties of programs in propositional temporal logic”, POPL '86 Proc. of the 13th ACM SIGACT-SIGPLAN symposium on Principles of programming languages”, pp. 184-19
[6] I. Tripathi, A. Velayoudame, R. Gupta, V. Garipelli, “Deadlock & Livelock Detection in Power Controller using under-constrained Formal Testbench”, Proc. Of the Design Automation Conference, 2019
[7] N. Kim, J. Park, H. Singh, and V. Singhal. “Sign-off with Bounded Formal Verification Proofs”, DVCon 2014.