2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

FV: A Robust Solution for Tackling Design Complexities –A Case Study on In-Band ECC

Yash Sachin Pawar, Aarti Gupta, Disha Puri
Intel Corporation
yash.sachin.pawar@intel.com; aarti.gupta@intel.com; disha.puri@intel.com

***Abstract*- Formal Verification (FV) is universally accepted as an effective approach to exhaustively validate IPs. Traditionally, FV has supplemented simulation-based testing to achieve complete coverage of the design. However, with the enhanced maturity of FV and utilization of advanced techniques it has the potential to replace traditional dynamic validation methodologies and be the primary validation vehicle. One of the primary obstacles to this is the design complexity that can be broadly categorized into sequential, spatial, and computational complexities. Usually, FV signoff in large designs consisting of all these complexities is achieved after multiple generations of the IP or by heavily extending the verification cycle. This paper presents a case study on the validation of an In-Band ECC design comprising of the afore-mentioned complexities. The validation effort resulted in a short turn-around time of a few months, during which multiple corner-case bugs were found. It also highlights targeted approaches in FV such as abstractions and symbolic simulation that were key in dealing with individual design complexities and significantly improving proof convergence.**

I Introduction

Modern-day processors and SoCs have significantly increased in their complexity. Each generation introduces new performance enhancing IPs that need to be integrated with legacy designs and verified thoroughly to make the desired impact. It follows that the verification problem also becomes tougher, almost exponentially in proportion. FV, in conjunction with traditional simulation-based verification, has been used extensively to deliver bug-free IPs by exhaustively verifying the state-space derived from the design under test (DUT). However, due the breadth-first search nature of FV proofs, the design size and complexity exponentially affect FV's ability to deliver conclusive verification results.

Design complexity can arise due to multiple reasons, and can broadly be categorized into sequential, spatial and computational complexities based on their source and impact on the design. Sequential complexity is associated with behavior that occurs after multiple clock cycles such as design initialization, timeout, or the presence of FIFOs. Identifying these would usually require highlighting blocks that add to the sequential depth of the system. Spatial complexity, on the other hand, refers to the size and complexity of the system's state space, which is the set of all possible configurations or states that the system can be in. Memories and large data bus sizes are common indicators of this. Computational complexity arises due to involved arithmetic algorithms such as hashing, floating-point arithmetic, Error Correction Codes (ECC) or Cyclic Redundancy Codes (CRC).

It is not, however, the case that FV can't handle these complexities. Multiple advanced methodologies have been developed to effectively deal with the complexity problem in FV. There are several instances of large DUTs (Design Under Test) that have been solely verified by FV. [1] highlights the use of FV as the primary validation vehicle for the core execution cluster in the Intel® Core™ i7 design. Similarly, the benefits of Symbolic Trajectory Evaluation (STE) in the complete verification of Floating-Point Units (FPUs) are described in [2]. However, both activities were centered around computational arithmetic verification.

This paper aims to demonstrate an effective usage of the available techniques in the FV arsenal to find critical bugs and present a mainstream FV sign off on a large design spanning across its control path and datapath. For this, an In-Band ECC (IBECC) design has been considered as a reference. Different FV techniques, such as abstraction were, adapted to resolve the sequential and spatial complexities in the control path. Likewise, symbolic simulation was used to deal with computational complexity in the datapath.

II Case Study: In-Band ECC

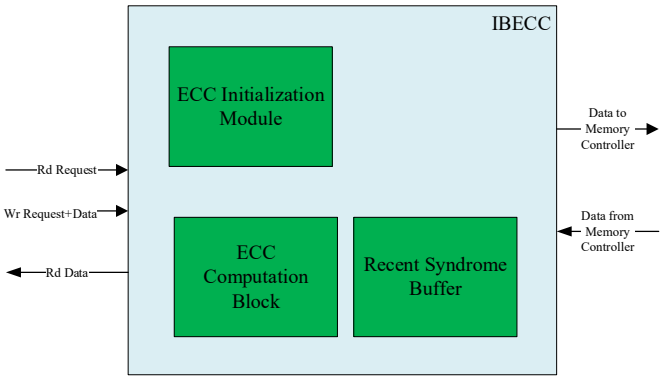

Figure 1 : Complexity Inducing blocks in IBECC

In-Band ECC is designed to provide ECC support to DDR memory. The design in consideration consists of the main packet transport logic, an ECC initialization engine and a local cache tracker memory in its control path as shown in Figure 1. Its datapath comprises of the ECC computation and correction blocks. Analyzing how each of these blocks introduce design complexities, we can correlate the ECC initialization module to sequential complexity, the recent syndrome buffer to spatial complexity and the ECC checking block to computational complexity.

A. Tackling Sequential Complexity

Sequential complexity associates with behavior that occurs after several clock cycles, such as the ECC initialization sequence here. The initialization engine writes to all protected addresses before requests can be sent to the memory. This stall to regular requests increases the sequential depth to be analyzed in Formal environment. To deal with this, an initial value abstraction was applied on the state elements of the initialization engine. The block was separately verified in an abstracted FV environment where the FSM could be at any state of reset. This allowed all state transitions and outputs to be covered and verified at a low sequential depth. Once verified, it was replaced with a formal model in the main environment which allowed the main environment to start out of reset in an initialized state.

ECC storage initialization module Verification for IBECC:

A Finite State Machine (FSM) as shown in Figure 2 that generates write that zero out protected memory regions during initialization. Blocks regular read and writes to protected regions during initialization. Initialization even for minimum address space of 32MB with 32B data bus involves minimum of 10^6 clock cycles. Thus, witness on properties for regular reads and writes are practically impossible to observe in a Formal Verification environment. Therefore, we need to eliminate the sequential complexity introduced by the initialization module without loss of generality.

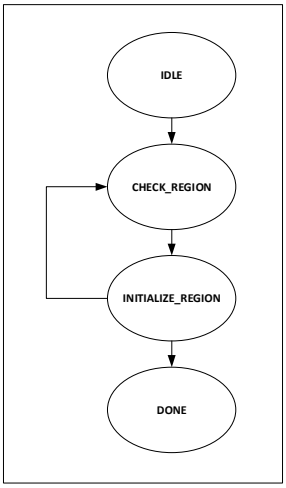

Figure 2: ECC Initialization FSM

We break down the problem into two parts:

1) Verification of Functionality of ECC storage Initialization Module
2) Replacement of the RTL module with an Abstract model and then proving other properties

Verification of Functionality of ECC storage Initialization Module

Due to huge sequential complexity involved in the Initialization finite state machine, Initial value abstraction is applied on all state elements involved with the FSM. The objective is to reach any possible state of state machine out of reset. The method involves writing a set of properties that define all states of the FSM. The reset state of the state elements involved in FSM can take random value out of reset as shown in Figure 3. The properties are applied as constraints for the first cycle. With these constraints in place, the same properties are to be proved for all cycles of the state machine. The properties largely revolve around current state of FSM and previous state of FSM. The legal transitions between states define the values that these variables can take together.

In the FSM, we need to make sure that all protected addresses must be initialized. For this, we create a symbolic address such that the address is a protected address belonging to range scheduled for initialization. We need to prove that the symbolically chosen address will be written during initialization. Since Init-value abstraction is applied on the FSM, the flag which registers that symbolic address was written to is also abstracted to take suitable value depending on FSM state, previous FSM state and other state elements.

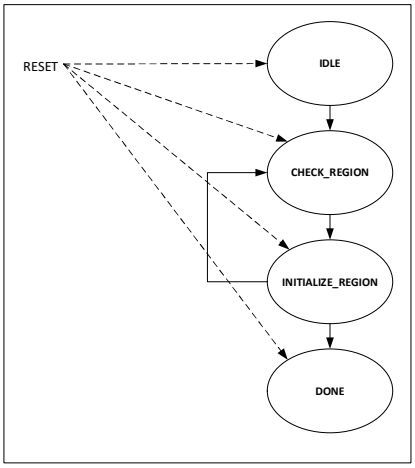

Figure 3: Initial Value Abstraction on ECC Init FSM

Replacement of the RTL module with an Abstract model and then proving other properties

In the abstract model FSM defines the conditions for state transition as free signals instead. The outputs of the RTL design are cut from their drivers and will be driven by the abstract model by applying constraints on those signals as shown in Figure 4.

The properties in these constraints need to be proved by the Init Value Abstracted Init FSM module. While doing this the free signals that define state transition in the new state machine are constrained to be driven as per the conditions that appear in the original state machine in the design.

The new FSM causes the initialization process to be quickly done.

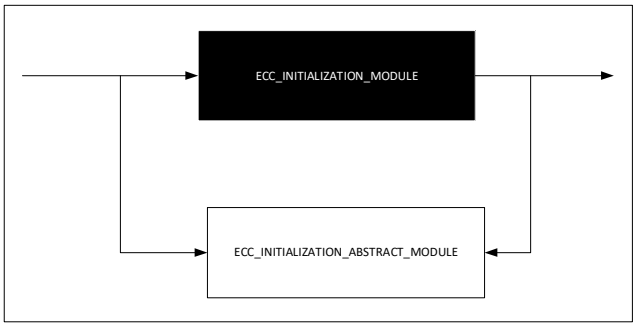

Figure 4 : Black boxing ECC Init FSM and driving its outputs with the abstract module

B. Tackling Spatial Complexity

The recent syndrome buffer control logic and memory added to the spatial complexity of the design. Memories and large data bus sizes increase the number of possible configurations or states that the system could be in. To cut down on the state space, the cache memory of the design was black boxed. The cache controller and tracker unit were abstracted out and separately verified. In doing this, inputs to the cache controller were tool driven and the sequential depth of the main request flow was cut out. Like the initialization engine, once verified, the cache controller and memory were replaced with a formal equivalent in the main environment.

Recent Syndrome Buffer Verification for IBECC

Recent Syndrome Buffer is a fully associative cache. The tag width for this cache is 34-bit address. The cache stores ECC values for read requests. The cache controller is involved in decision for allocation and evictions of entries. For eviction, the controller maintains count of reads which would consume the ECC value stored in the cache.
Each new read can hit and miss with the cache, to know the result, the incoming read address serving as a tag needs to be matched with all valid addresses (tags).
We break down the problem into two parts:

1) Verification of Functionality of Recent Syndrome Buffer
2) Replacement of the RTL module with an Abstract model and then proving other properties

Verification of Functionality of Recent Syndrome Buffer

Initial value abstraction is applied on all state elements. The objective is to reach many advanced states out of reset as shown in Figure 5. The method involves writing a set of properties that define all legal condition between the state elements. The reset state of the state elements can take random value out of reset. The properties are applied as constraints for the first cycle. With these constraints in place, the same properties are to be proved for all cycles. Define Invariants that will be true in the design and use these properties to constraint out of reset and then check if those properties hold in further cycles.
In the cache, we need to make sure that hit/miss hold for all entries. For this, we create a symbolic address. We need to prove that the symbolically chosen address provides correct hit/miss response and does not get evicted till it has no pending consumers.

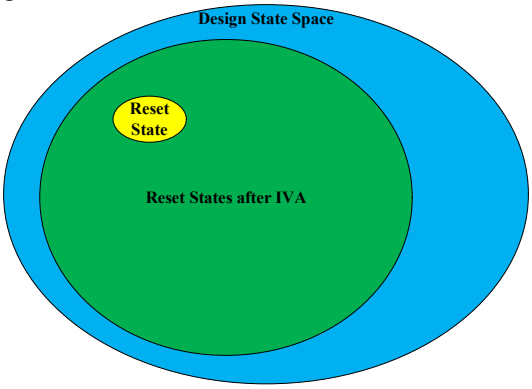

Figure 5 : Applying IVA to begin cache controller from any advanced state.

Replacement of the RTL module with an Abstract model and then proving other properties

The properties that are proved earlier are used as constraints to drive outputs of RSB as shown in Figure 6. The constraints on input used for proving the abstraction are proved. The Outputs of Content Address matching operation for all addresses other than a symbolic address are free, while the model maintains proper outputs for the symbolic address.

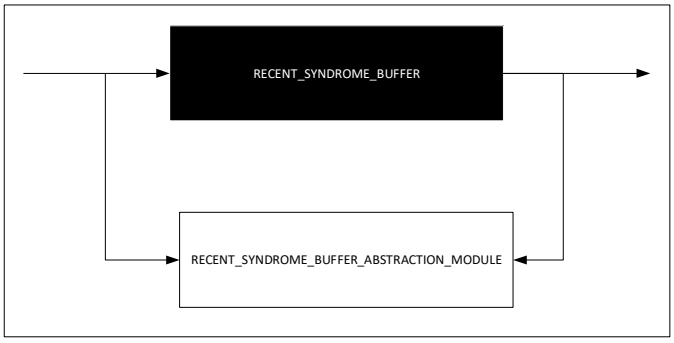

Figure 6 : Blackboxing Recent Syndrome Buffer and driving its outputs with the abstract module.

C. Tackling Computational Complexity

The ECC module in the design supports two modes of protection i.e., Single Error Correction Double Error Detection (SECDED) and Zero Error Correction Triple Error Detection (ZECTED), and this mode selection can be done in configuration settings. DUT supported an ECC implementation that provides protection on 64B or 32B granularity. ECC algorithms offer both spatial and computational complexity and that makes proof-convergence using traditional model-checking based FV tools difficult. Thus, a targeted approach is needed, that can tackle the computational challenge offered by layers of XOR trees in ECC design. As discussed in [3], symbolic simulation based FV approach suits ECC-based designs very well.

ECC Verification for IBECC:

Using this symbolic simulation approach, an end-to-end verification of ECC datapath logic was done with FV testbench set-up as shown in Figure 7. In this setup, a symbolic data D_w is driven to the Writer module, that generates the check-bits and appends with original data to form a write-code-word CW_w. This write code word in real design gets stored in memory where it can be subject to corruption. In FV testbench, the memory storage is abstracted and replaced with a corruption modeling, where CW_w is bit-wised XOred with a symbolic corruption vector C, to form a read-code-word, CW_r. This code word then goes to the Reader block which uses the redundant information to detect and correct the corruption in accordance with the algorithm's capabilities. The Reader block outputs data D' and Error-signals: No-Error (NE), Correctable Error (CE), and Detectable but Uncorrectable Error (DUE). After symbolic simulation of this FV TB, symbolic expressions can be extracted out at Reader's output and then compared with original Data and corruption symbols to make logical inferences about the correctness of the ECC algorithm.

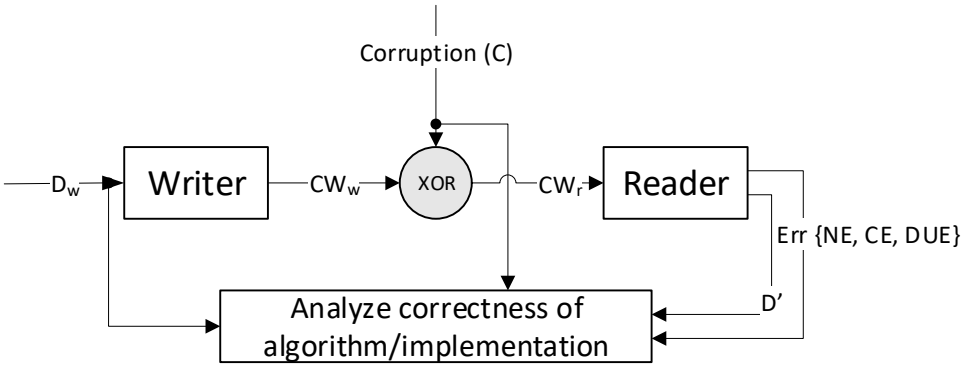

Figure 7 : Datapath Verification for ECC

III Results

The approaches discussed above drastically decreased design complexity and helped uncover 20 bugs. 14 of these were discovered in the control and data-transport logic with the help of the ECC initialization sequence abstraction. The abstraction of the cache memory and tracker caught three bugs. Each of these three were caught within 10-20 cycles in the abstracted formal environment, which would have otherwise required at least 400 cycles to be caught. Three critical datapath bugs were discovered using the ECC verification approach. One of these had an incidence rate of 0.000002% and would have probably gone undetected by all other forms of verification, had this verification strategy not been applied.

Example: While Verifying the RSB cache controller separately with IVA, we found a mismatch in full indications between RTL and FV model. The cause was found in RTL counter being 6 bits, not sufficient to hold value 64, which is size of cache. This scenario is difficult to achieve if the design starts empty(rtl_pending_entries_count=0). After Applying Initial Value Abstraction, we have this scenario where we start from advanced state where (rtl_pending_entries_count=60). Thus, using Initial Value Abstraction we can uncover bugs which are at deeper bounds and difficult to be proved without use of complexity reduction methods.

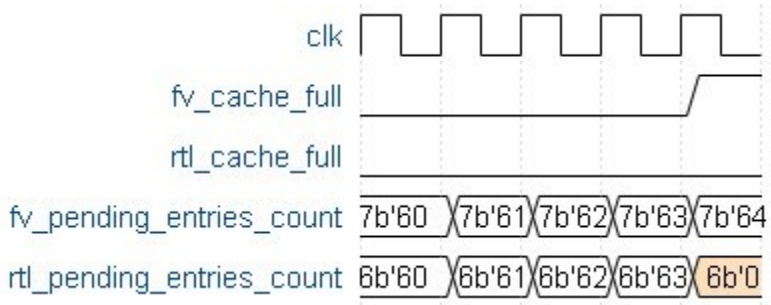

Figure 8 : Bug found after applying complexity reduction techniques.

IV Conclusion

Any large DUT like the one considered in the paper is expected to have significant sequential depth, and spatial and computational complexity. These can have a significant impact on the feasibility and effectiveness of FV techniques. In general, as these complexity inducing factors increase, so does the difficulty in verifying the design's functionality. The techniques mentioned in this paper can efficiently analyze large state spaces or deep sequential depths. Incorporating such techniques in a formal validation flow can substantially cut-short the timeline and yield results.

V References

[1] Kaivola R. et al. "Replacing Testing with Formal Verification in Intel® Core™ i7 Processor Execution Engine Validation," CAV 2009.

[2] M, Achutha Kirankumar V, Gupta, Aarti, Ghughal, Rajnish. "Symbolic Trajectory Evaluation. The Primary Validation Vehicle for Next Gen Intel® Processor Graphics FPU," FMCAD 2012.

[3] Aarti Gupta, Roope Kaivola, Mihir P Mehta, Vaibhav Singh, "Error Correction Code Algorithm and Implementation Verification Using Symbolic Representations," FMCAD 2022.