Early Bird Catches the Bug - The Arch Formal Way

Aatreyi Bal, Mahendrawada Sai Kameshwara Rao, Sireesha Tulluri, Robert Beers
Formal Verification Central Tech Office (FVCTO), Intel Corporation
SRR Campus
Bangalore, India-560100

***Abstract*- Home Agents (HA) that maintain coherent memory space are complex in design. They interact with multiple Caching Agents (CAs) using a cache coherence protocol and interacts with memory, supporting multiple protocols including CXL.mem protocol. The design complexity of such HAs is huge and architectural bugs are a recurrent problem. Such architectural bugs (e.g., deadlocks) and behavioral issues are costly to fix in RTL, even worse in silicon. As incorrect architecture/specification could result in re-spins, finding bugs at architecture definition is crucial. This paper talks about shifting left formal verification to architectural definition stage and validating features within a short verification cycle which would help in solidifying the specification.**

I. Introduction

The ever-changing landscape of modern age microprocessors has introduced many complex architectural paradigms. However, these cutting-edge architectures are still plagued with the age-old problem of architectural bugs. Deep-rooted and disguised architectural and micro-architectural bugs are one of the biggest pain-point of architects. Architectural bugs, when caught at later stages of development, for example in the Register Transfer Logic (RTL) or in silicon, can be costly in terms of both efforts to fix and time-to-market. Architectural Formal Verification (ArchFV) left shifts the formal verification exercise to the design architecture specification phase, sometimes even before the specifications are documented. The early discovery of the critical architectural bugs ensures a shorter feedback loop for correction and a quicker turn-around-time.

In this paper, we describe the usage of our ArchFV methodology in validating an Intel Next-Gen Home Agent (HA) and highlight the discovery of a critical architectural bug in the same. The HA supports multi-protocol transactions. The consequent design complexity makes the HA an ideal case study for our ArchFV methodology.

The rest of the paper is organized as follows:

- Section II highlights some of the related works in this domain.
- Section III sets the motivation for this paper and presents some cases that highlight the need for our ArchFV methodology.
- Section IV briefly discusses the tools and techniques involved in our ArchFV methodology.
- Section V discusses an intriguing architectural bug, thereby highlighting the success of our ArchFV methodology.
- Section VI concludes the paper.

II. Related Works

This paper highlights the Architectural Formal Verification (ArchFV) methodology and its role in a real-world scenario. Over the years, many research papers have explored different ArchFV tools and methodologies, like Murphi and TLA+ [1,2]. Beers et al., on the other hand, have explored the concept of pre-RTL verification and its implementation in Intel products [3]. However, the novelty of this paper lies in the fact that this paper shows a practical implication of ArchFV on a next generation HA.

III. Motivation

In this section, we will briefly describe the HA we are considering for our case study, the architectural properties that we aim to verify and present some corner cases that will establish the need for ArchFV methodology in formalizing system design specification. Finally, we will discuss the agenda for our ArchFV methodology.

A. *Design Overview*

To maintain the confidentiality of the IP, we will present a “necessary and sufficient” abstracted model of the HA. The HA is complex as it interacts with multiple caching agents (CAs) using an in-house cache coherence protocol (say CCP) and talks to memory supporting multiple protocols including CXL.mem. HAs are responsible for servicing all requests to the coherent memory space. For read requests, HA must generate necessary snoops, process their responses, send data and completion response if necessary. For write requests, these must commit the data to memory and send completion response. Fig. 1 shows a block diagram of the design IP, consisting of the HA, the CAs and the CXL devices, along with the respective communication protocols.

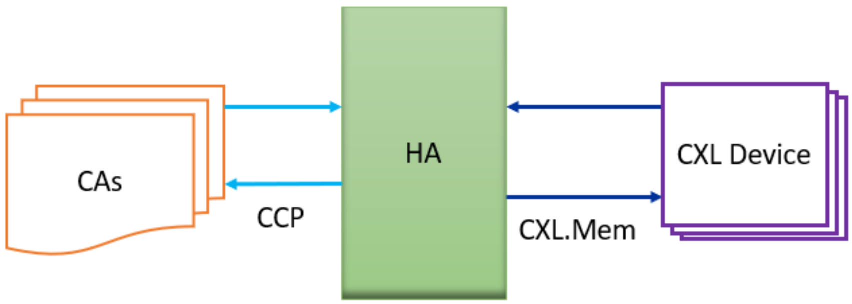

Figure 1: Abstract block diagram of the IP

Both the protocols support requests of different message classes, for example read request (READ) message class, write request (WRITE) message class etc. These varied incoming requests can give rise to control hazards like Read-After-Read (RAR), Read-After-Write (RAW), Write-After-Read (WAR) and Write-After-Write (WAW). HA uses an ordering mechanism to handle internal resource allocation and deallocation of these requests, to avoid the control hazards. The primary goal of this ordering mechanism is to ensure that the protocols are not violated and there is no corruption of data across the system. In the next section, we highlight the properties we aim using ArchFV methodology.

B. *Architectural Property*

Table 1: Table of ArchFV Properties

ArchFV Properties	Description
Data Consistency	To check whether the latest data is maintained among the caches. This is done by streamlining memory accesses by the requests.
Cache Coherence	To check whether the cache states do not violate the MESI protocol. Which means no two caches can be in Exclusive or Modified state or no cache should be in Exclusive State while another is in Shared state.

Table 1 shows the properties we verify using our ArchFV methodology. The idea is to map the properties to rules in our architecture model. The rules are enabled by the current value of the state elements and trigger state transitions according to the FSM model. For example, to verify that the cache coherency is maintained after each state transition in the HA, we design an invariant property. This property ensures that the cache coherency is always maintained. The modelling and verification of these properties allow us to unravel interesting corner cases in the design, as discussed next.

C. *Corner Cases*

The corner cases, discussed here, depict deviations from usual case scenarios. ArchFV methodology ensures that these special cases do not violate any architectural property and fortify the correctness and soundness of the model. In this section, we discuss three of the corner cases corresponding to WAR hazard, HA eviction flow and CXL request clash with CCP requests.

a) WAR Hazard Scenario

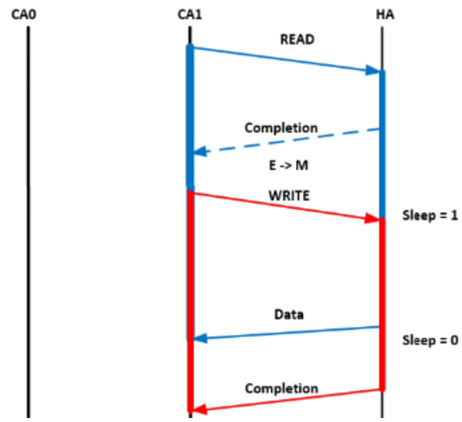

Figure 2: Feynman Diagram for an exception case of WAR conflict

Fig. 2 depicts the WAR corner case when the READ request is completed from the CCP perspective, i.e., completion has been sent out by the HA, but the resource in the HA is not released. In this case, the ArchFV property checking for the data consistency in the CCP environment reveals that the usual priority of WRITE request over READ request needs to be modified. Therefore, the data consistency property forces the WRITE request to be deprioritized and processed after the HA resource is freed by the READ request.

b) HA Eviction Flow

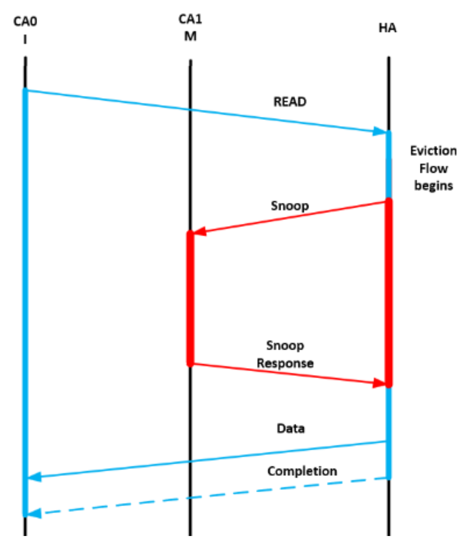

Figure 3: Feynman Diagram for Eviction Flow

Fig. 3 shows an abstracted flow of eviction flow. Since the HA needs to maintain cache coherence as per the CCP, it internally maintains a lookup-table like structure to record the cache state information. When the table is full, the HA follows a mechanism to evict an entry from the table to make space for newer entries. This mechanism enables the HA to send spontaneous snooping requests to the CAs. These requests are allowed to make progress along with any in-flight READ request that does not depend on the evicted entry. However, the CCP allows only one snooping resource for an address at a time. Consequently, a resource contention ensues between the active READ request and the eviction request. In ArchFV methodology, we model these eviction requests such that they neither violate the protocols, nor cause a resource deadlock. In addition, the ArchFV methodology also confirms that data consistency property is always upheld. And, to ensure that, ArchFV methodology verifies that the eviction requests do not interfere with any active WRITE request or any active READ request that refers to the evicted entry in the lookup table.

c) CXL.mem Request Clash with CCP Request

ArchFV model of the HA gets even more complicated with the interplay of CXL.mem and CCP requests. While CXL devices can initiate independent requests like Back-Invalidation Snoop (BISnp) requests, they also act as external memory devices for the HA. Therefore, requests from CCP can also spawn memory access requests to the CXL device from the HA. The interplay of these requests from multiple sources potentially gives rise to circular dependencies. The task of ArchFV in such a scenario is to scrutinize the architectural model to unravel any such circular dependency, and break them. In addition, ArchFV dives deeper to ensure the HA microarchitecture is equipped to handle subsequent resource management to counter any potential deadlock. One such circular dependency led us to a bug, that we discuss in Section V. But before that, we discuss our tools and methodology , briefly, in the next section.

IV. Methodology

The ArchFV methodology involves building a model, verifying predetermined architectural properties against the model and finally signing off on the golden specification of the design. ArchFV engineers work with architects to understand their ideas and create a model that correctly captures those ideas/specifications. ArchFV methodology involves modeling of all possible behaviors of a system, while abstracting out low-level details. The models are developed and refined using Failure Driven Development (FDD) approach, i.e., starting with a basic model and newer scenarios are modelled as and when related failures are observed in the model. First, we build our model using an in-house python-based tool. Next, we run the model through the Murphi verifier[1]. The formal verification engine throws counterexamples (CEX), if the model fails to validate the required architectural properties. Finally, we use these CEX to correct and refine our model. FDD ensures that the model is strictly sufficient.

The novelty of our methodology is that we judicially create an abstract model such that all the critical features of the design are covered while any non-critical low-level detail is left out. It also allows us to scale down the model so that we hit corner cases faster. In the next section, we discuss a bug which was caught early due to proper abstraction in our ArchFV methodology.

V. Results

Our ArchFV methodology discovered a hidden architectural bug in the design. The bug in question was a multi-layered problem that we identified in a very short span of time (~2-3 weeks) using ArchFV abstraction. We discussed the failing scenario with architects and even collaborated with them to devise a fix. We have incorporated the fix in the model and formally verified before confirming to the architects that the fix is working fine. The modeling and verification of the bug, as well as modeling of the fix were completed in a month.

Before digging deep into the bug, we will explain the dependency cycle that caused the bug, in the next section.

A. Dependency Cycle

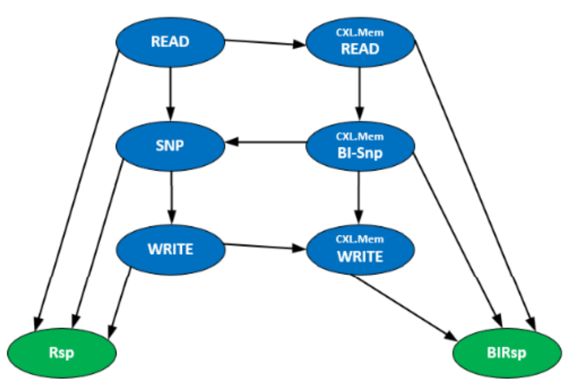

Figure 4: Dependency cycle among multi-protocol requests

The following steps describe the dependency cycle among the request classes, as shown in Fig. 4:

- A CCP WRITE request sends out a MemClnEvct request to the CXL device.
- At the CXL device, an outstanding BISnp blocks the MemClnEvct.
- At the HA, the BISnp is blocked by the CCP WRITE request.
- The CCP WRITE request keeps waiting for the MemClnEvct to complete.

To break this multi-protocol dependency cycle, the architects decided to send out the BIRsp, immediately. Though this implementation resolved one layer of the issue, a deeper issue prevailed regarding the resource blocking in HA.

B. *Bug Discovery*

While digging deep with our ArchFV methodology, we noticed that though the response to the BISnp is sent out, the HA resource is not freed. The high-level abstraction in our model helped us to minimize the HA resource pool and prove that a resource deadlock is possible in this case, thereby uncovering the hidden bug. We showed that, once the CXL device receives the response for the active BISnp, it can initiate a new BISnp transaction. This transaction will be blocked from entering the HA due to lack of available resources, considering all shared and reserved resources have been used up. Meanwhile, a dormant BISnp transaction keeps blocking the resource. Fig. 5 presents the flow that identified the bug. The overlapping colored lines on the bar in the figure depict the resource blocking. Using abstraction in ArchFV methodology, we proved that the resource blocking will eventually lead to a deadlock.

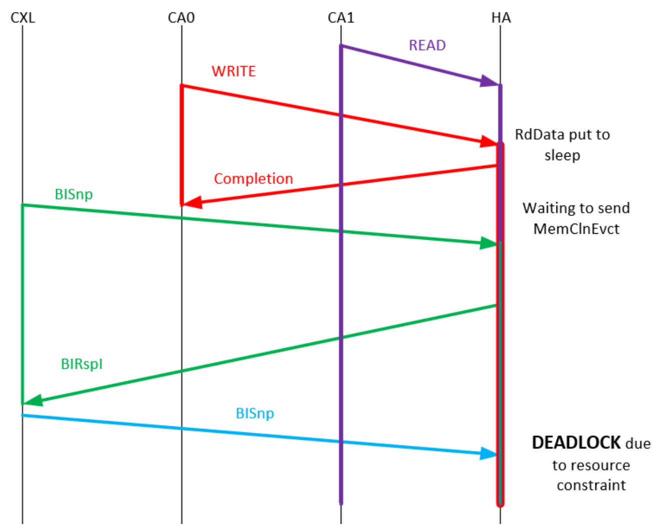

Figure 5: Feynman Diagram of the buggy flow causing the deadlock

To resolve the deadlock, it was formulated that the dormant BISnp transaction should be deallocated from the HA. This fix was also verified by our abstracted ArchFV model, within a short span of time. The focus here is that the abstraction in ArchFV methodology enabled us to catch the bug early on, as well as, verify the fix quickly. The same might have taken a long time in RTL. This is because the scenario requires all resources for BISnp to be blocked in transaction, which is an extreme corner case and hard to hit in RTL.

VI. Conclusion

This paper highlights how the left shift in the verification cycle, by the ArchFV methodology, facilitates the timely development of our next generation IPs. Over the years, ArchFV has helped architects to find deadlocks and fix them. ArchFV is especially efficient in identifying disguised dependencies and corner cases. In this case also, ArchFV uncovered a deadlock which was handled architecturally. Early detection of such deep-rooted bugs avoids bug escape to RTL and saves time and cost.

ACKNOWLEDGMENT

We would like to express our sincere thanks to Sharada Venkateswaran, Nishant Singh, Anupama Seshadri and Nithin Marepalli for their continuous support and knowledge-sharing regarding the IP design. We would also express our sincere appreciations to Achutha KiranKumar VM and Bathri Narayanan Subramanian for their relentless encouragement towards our work.

REFERENCES

[1] Dill, David L. "The Mur ϕ verification system." In Computer Aided Verification: 8th International Conference, CAV'96 New Brunswick, NJ, USA, July 31–August 3, 1996 Proceedings 8, pp. 390-393. Springer Berlin Heidelberg, 1996.

[2] Lamport, Leslie. "Specifying systems: the TLA+ language and tools for hardware and software engineers." (2002).

[3] Beers, Robert. "Pre-RTL formal verification: an intel experience." In Proceedings of the 45th annual design automation conference, pp. 806-811. 2008.