On Benefits of Modeling the HPDcache in LNT

Zachary Assoumani[1]*, César Fuguet[2], Radu Mateescu[1], and Wendelin Serwe[1]

[1]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP,† LIG, 38000 Grenoble, France
[2]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP,† TIMA, 38000 Grenoble, France

Abstract

Stepping from natural language towards modern formal languages such as LNT is beneficial for specifying hardware architectures. We illustrate this on the HPDcache, the informal specification of which contains numerous fragments in pseudo-code. Due to the syntactical similarities between the latter and LNT, modeling the HPDcache's informal specification in LNT was greatly facilitated. The CADP tools supporting LNT enabled us to spot an error in the informal specification of the HPDcache, which might have led to a violation of the memory consistency rules of the RISC-V.

1 Introduction

To reduce costs in hardware design, it is crucial to spot any unwanted behaviours early in the design process of complex architectures, starting with the informal specifications, which are particularly prone to ambiguities and oversights. Formal methods are suitable vehicles to describe, simulate, and also verify specifications of architectures.

We illustrate these steps on the High-Performance Data Cache (HPDcache) [1], a non-blocking L1 data cache for RISC-V cores and accelerators. Starting from its informal specification, we produced a formal model of the HPDcache in the LNT language [2]. We also formally expressed the memory consistency rules of the RISC-V specification [3] as MCL [4] formulas. The CADP tools [5] supporting LNT and MCL allowed us to uncover a possible violation of the memory consistency rules by the informal specification. The remaining sections present these steps in more detail.

This is not the first illustration of the benefits of formal verification in hardware design, but it highlights that modern languages, such as LNT and MCL, substantially reduce the cost of formal verification due to their syntactical similarities with the informal specifications.

2 Description of the HPDcache

The most prominent features of this cache are:

- Highly configurable to match the Performance-Power-Area (PPA) requirements of different application domains (from low-energy to high-performance embedded systems)
- Support of overlapping of a high number of requests to hide the memory latency
- Pipelined architecture to reach high-clock frequencies in advanced technology nodes
- Support of Atomic Memory Operations (AMOs) and Cache Management Operations (CMOs) to enable multi-core processor integration
- Hybrid write-policy: It implements both write-through and write-back policies, which are either supported simultaneously at cacheline granularity (at runtime) or separately (statically configured at synthesis time)

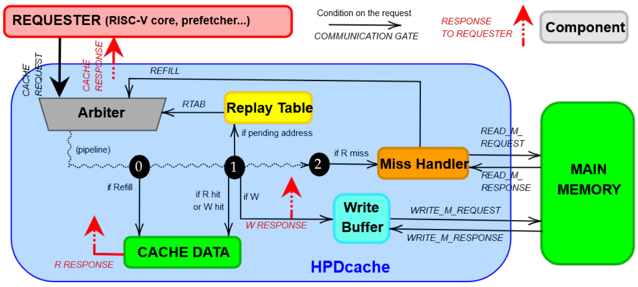

Figure 1: *Overview of the HPDcache*

The HPDcache has been integrated into the OpenHW CVA6 RISC-V core, and became its official cache. Different companies (e.g., Bosch or Thales) and academic institutions (e.g., BSC, CEA, ETH-Z, Inria, UCSB) use it for their products or research.

The natural-language (informal) specification and the RTL model in SystemVerilog (SV) of the HPDcache are available in open-source[1] and hosted by the OpenHW Foundation.

3 Formal Model

We translated the English specification of the HPDcache into the LNT formal language [2], totaling around 1,200 lines of code. The LNT model follows

*Corresponding author: `zachary.assoumani@inria.fr`
†Institute of Engineering Univ. Grenoble Alpes

[1] https://github.com/openhwgroup/cv-hpdcache

```
int rtab_find_ready(int last) {
  int i = (last + 1) mod % RTAB_NENTRIES;
  for (;;) {
    if (rtab[i].valid && rtab[i].ll_head &&
        (rtab[i].deps == 0))
      return i;
    if (i == last)
      return -1;

    i := (i + 1) % RTAB_NENTRIES;
  }
}
```

```
function rtab_find_ready(rtab:RTAB_Array, last:int): int is
var i:int in i := (last + 1) mod RTAB_NENTRIES;
  loop
    if rtab[Nat (i)].valid and rtab[Nat (i)].ll_head and
       (rtab[Nat (i)].deps == NoDeps) then
      return i
    elsif i == last then
      return -1
    end if;
    i := (i + 1) mod RTAB_NENTRIES
  end loop
end var end function
```

Figure 2: *Pseudo-code (left) and its translation to LNT (right)*

the cache architecture on Fig. 1, i.e., the parallel composition of one process per component, each one with its own local data structure. All components synchronize on a dedicated gate to give the arbiter an atomic access to all local states. Fig. 2 shows the translation of a HPDcache routine from pseudo-code to LNT.

Our model also includes the processor generating requests and the main memory, each communicating with the cache through a dedicated load/store interface.

The correct functioning of the cache is defined by memory consistency rules [3, chapter 14.1]: *safety* (a read request of an address returns the last written value) and *liveness* (every request is eventually executed). We formalized these properties in MCL [4] and verified them on the HPDcache LNT model using CADP. For instance, the safety memory consistency rule is specified using the MCL formula below:

```
[ true*.
  {REQ ?any ?any ?s:nat ?t1:nat ?a:string}.
  not ({RSP_W ?any !s !t1} or {RSP_R ?any !s !t1})*.
  {REQ ?any ?any !s ?t2:nat !a where t1 <> t2}.
  not ({RSP_W ?any !s !t1} or {RSP_R ?any !s !t1})*.
  {RSP_W ?any !s !t2} or {RSP_R ?any !s !t2}
] false
```

The modality "[...] false" forbids the wrong execution sequences consisting of two consecutive requests (transactions `t1` and `t2`) made by a source `s` on the same address `a`, followed by a response to the second request.

To avoid combinatorial explosion, we limited the LNT model to two different data values and two memory locations. Using CADP, we generated the corresponding state space ($814,479$ states and $2,919,608$ transitions, reduced for strong bisimulation).

Attempting to verify the consistency property above, it turned out to *not* hold. Investigating the counterexample sequence of 37 transitions provided by CADP, we uncovered an error in the specification, for which we created an issue[2] in the official HPDcache repository, which has been fixed (the SV code was not affected).

In a nutshell, the problem was that attempting to reply a request from the replay table modified the internal state of the replay table, and these changes were not undone when the request due to a conflict could not be executed and had to be rolled back.

4 Conclusion

Using formal languages brings the benefits of automated verification and simulation already in early design stages. We showed that for the modern formal language LNT this benefit comes with little cost, the formal specification being very close to the pseudo-code often present in informal specifications. For the HPDcache, our model enabled to detect and fix an error in the informal specification. The formal modeling task was carried out in less than two months, in the context of a training exercise to learn LNT.

It is planned to include our formal model in the official repository as additional reference and enable modeling of further, and future, features of the HPDcache. We believe that this provides a valuable example for other designers of similar hardware architectures involving several parallel, interacting components.

References

[1] C. Fuguet. "HPDcache: Open-source high-performance L1 data cache for RISC-V cores". In: *Computing Frontiers 2023*.

[2] H. Garavel, F. Lang, and W. Serwe. "From LOTOS to LNT". In: *ModelEd, TestEd, TrustEd*. Vol. 10500. Lecture Notes in Computer Science. Springer, Oct. 2017, pp. 3–26.

[3] A. Waterman and K. Asanović. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA*. University of California, Berkeley, 2019. URL: https://archive.org/details/the-risc-v-instruction-set-manual-volume-i-user-level-isa-document-version-20191213.

[4] R. Mateescu and D. Thivolle. "A Model Checking Language for Concurrent Value-Passing Systems". In: *Formal Methods*. Vol. 5014. Lecture Notes in Computer Science. Springer, May 2008, pp. 148–164.

[5] H. Garavel et al. "CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes". In: *Springer Int. J. on Software Tools for Technology Transfer (STTT)* 15.2 (Apr. 2013), pp. 89–107.

[2] https://github.com/openhwgroup/cv-hpdcache/issues/60