Functional Verification Strategy for an Open-Source High-Performance L1 Data-Cache for RISC-V cores

Tanuj Khandelwal[1], Ludovic Pion[1], César Fuguet[1], Adrian Evans[1] *

[1]Univ. Grenoble Alpes, CEA, LIST
F-38000 Grenoble, France

Abstract

The verification of caches is particularly challenging, as it is necessary to ensure that the memory consistency is ensured in all conditions and modern caches contain many complex features (out of order execution, write buffer, multiple request ports) and a complex micro-architecture in order to ensure high-performance. The recently released, Open-Source High Performance L1 Data Cache (HPDCache) for the RISC-V is delivered as a highly configurable RTL model, compounding the verification challenge as correct behaviour must be ensured for all combinations of the parameters. An approach combining pseudo random constraint and directed random test within the framework of Universal Verification Methodology (UVM) is used to be able to verify HPDcache.

Introduction

Cache memories are an essential component of modern processors. Caches are part of the memory hierarchy and help fill the increasing processor performance versus memory bandwidth gap.

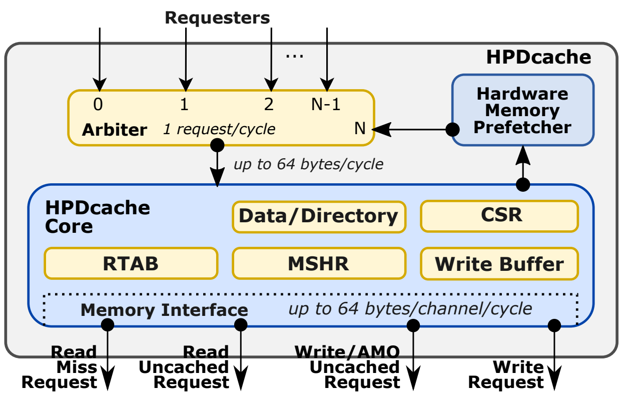

Figure 1: *HPDcache Top*

This work discusses the verification of the HPDcache. Figure 1 shows the top level of the HPDcache. The HPDcache is a highly-configurable, high-performance L1 Data Cache for RISC-V cores. It is implemented in SystemVerilog, uses a permissive open-source license, and is accessible through the OpenHW Github[1]. The objectives of this cache are on the one hand to enable both a high throughput and energy efficiency, and on the other hand, to provide several static (compilation-time) and dynamic (run-time) parameters to enable fine-tuning for systems with different performance, power and area (PPA) constraints.

*Corresponding author: `Tanuj-Kumar.KHANDELWAL@cea.fr`
[1] https://github.com/openhwgroup/cv-hpdcache

UVM Testbench Architecture

The Universal Verification Methodology (UVM) is used to verify the HPDcache. The architecture of UVM testbench is shown in Fig. 2.

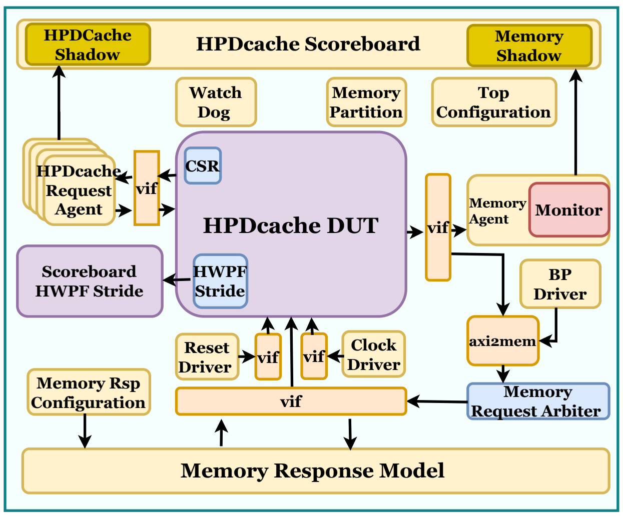

Figure 2: *UVM Testbench Environment*

Following are the key components of environment:

- An out-of-order memory response model
- HPDcache request agent
- Memory partition agent
- Reset, clock and back-pressure drivers, watch dog.

Checking

The scoreboard is extended from *uvm scoreboard*. The scoreboard performs the following notable checks:

1. Data consistency using shadow memory at byte granularity to check RISC-V Weak Memory Ordering (RVWMO)
2. Full prediction of Write Buffer Coalescing
3. Partial eviction (PLRU) prediction
4. Full error prediction

Directed Random Stimulus

The stimulus is grouped into tests using fully random and directed random sequences. Some notable example sequences include:

1. Totally random sequences which respect correctness constraints
2. Targeting a small memory partition to provoke conflicts
3. Focused on atomic operations
4. Sequences focusing on back-pressure to stress the write-buffer
5. Designed to stress hit,miss and eviction

Coverage/Assertion Driven Verification

The verification strategy is documented in a testplan containing an enumeration of :

1. Tests
2. Black box functional coverage
3. White box functional coverage
4. Assertion checks
5. Assertion coverage/schmoos

Functional Coverage SystemVerilog covergroups are used to cover stimulus, this example covers the cross of sets and words in the cache:

```
cov_set : coverpoint set
cov_word : coverpoint word;
cross : cross cov_set, cov_word, cov_op;
```

Coverage Assertions Assertions can be used to ensure all temporal relationships between two events have been covered (called a *schmoo*).

```
for(clk=0; clk<= 5 ; clk++) begin
 property schmoo;
 @( posedge clk_i )
   (arb_req_valid & arb_req_ready,
    set = dcache_req_set,
    tag = dcache_req_tag)
    |-> ##clk
   (mem_req_miss_read_valid_o &
    mem_req_miss_read_ready_i &
    mem_rd_miss_set == set &
    mem_rd_miss_tag == tag) ;

 endproperty
  u_cov: cover property (schmoo)
end // for
```

Checking Assertions Assertions are also used to check the correctness of protocols and handshakes.

```
arbiter_ready_onehot :
assert property (
@(posedge clk_i)
  ($onehot0(core_req_ready)) else
  $error("More than one request");
```

Compile Time Parameters

The HPDcache, like many IPs, is parameterizable. Verilog parameters in the RTL, define, for example, the number of SETS or of WAYS. There are 19 different parameters, which can give millions of possible combinations, making HPDcache verification very challenging. At compile time, the design is elaborated based on a single parameter configuration.

To ensure the HPDcache is functionally correct for all values of the parameters, a SystemVerilog module with knowledge of the parameters and associated constraints, is used to generate valid sets of parameters. A new, generic tool is used to generate the configurations (random, corners, exhaustive) and launch a full regression with the selected parameter values.

Conclusions

For the HPDcache to reach a TRL7, it will be delivered with the industrial quality verification suite described in this paper that will be made available as open source. Cache verification is challenging [1]. In this paper, we have focused on functional verification, however, the performance verification [2] is also an important aspect that must also be addressed.

Acknowledgements

This work has been performed in the context of the TRISTAN project. TRISTAN received funding from the Key Digital Technologies Joint Undertaking (KDT JU) under Grant Agreement no. 101095947.

References

[1] Henschel *et al.* "Pre-silicon verification of multiprocessor SoCs". In: *2013 IEEE ICECS*. 2013.

[2] Desalphine *et al.* "Novel Method for Verification and Performance Evaluation of a Non-Blocking Level-1 Instruction Cache". In: *2020 IEEE VDAT*. 2020.