HPDcache Subsystem Verification Environment

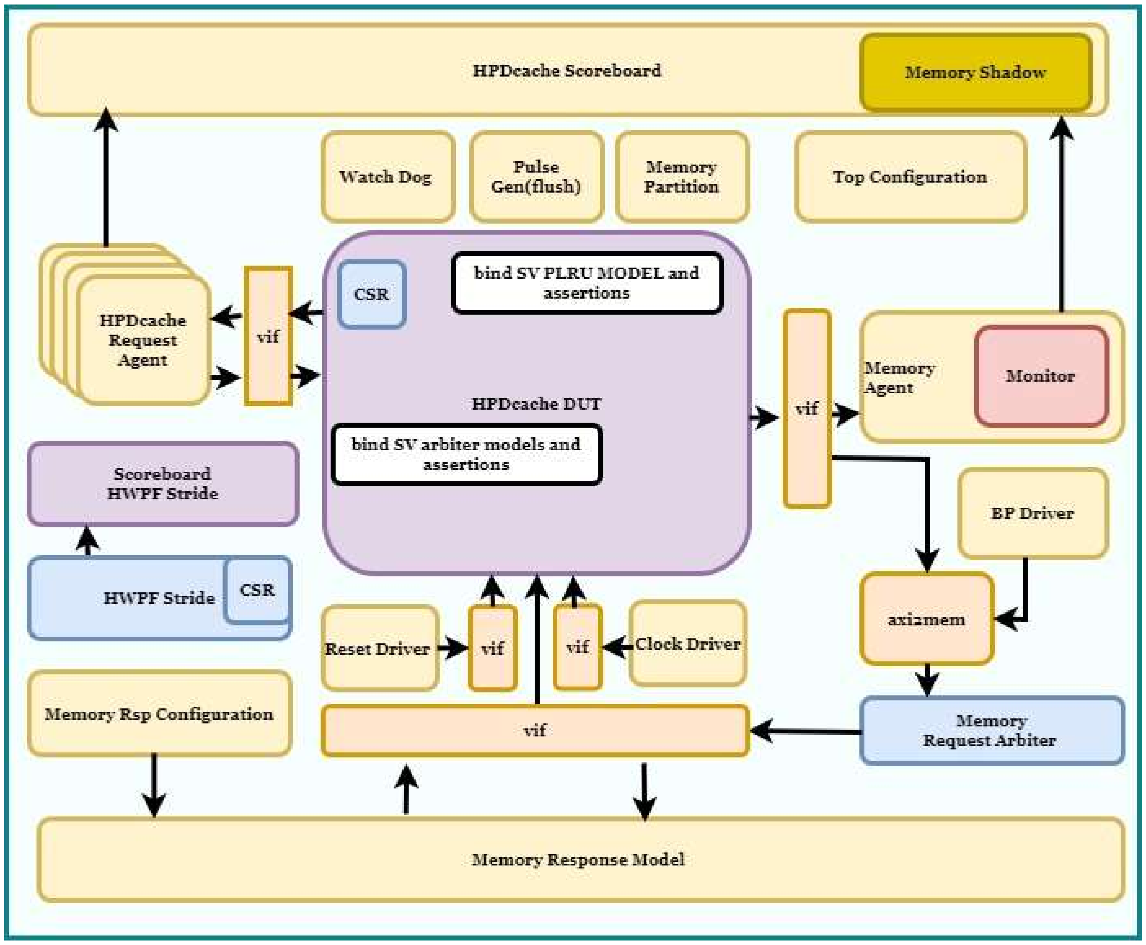
Table of content

1. Introduction

The purpose of this document is to outline the details of the verification environment of HPDcache Subsystem. This verification environment is used to verify HPDcache DUT and prefetcher DUT.

2. Environnement

The environment is shown in the figure below. The HPDcache subsystem contains two DUTs. HPDcache and Prefetcher. An optional AXI5 (not verified yet) adapter is provided to convert memory request to AXI request.



1.1 Reusable objects

Following existing reusable objects are used in the environment (cv_dv_utils):

1. Memory Response Model
    a. Provide out of order with random delay memory responses.
2. AXI2MEM converter
    a. Convert axi request to memory response model requests
3. Reset Driver
    a. Generate reset
4. Clock Driver
    a. Generate clock
5. Watch Dog
    a. Generate timeout
6. Memory Partition
    a. Divide large memory in small memory section to be able to run simulation within a limited region
7. BP
    a. Generate back pressure on memory interface of HPDcache
8. Pulse Gen
    a. Generate pulse to implement flush (not well tested yet)

2.1 Key Components

Following are the key components of the environment.

1. HPDcache Request Agent: This object is used to generate, drive and monitor the requests to the HPDcache. Following are some of the important pseudo random sequences :
    a. LR/SC sequence: The multiple LR/SC sequence runs single LR SC sequence. The single LR/SC sequence send 10 requests with the following constraints.
        i. 90% same address, 10% different random address
        ii. 40% LR, 40% SC, 10%LOAD, 1% each for the rest of STORE and AMOs.
        iii. The aim is to have all possible LR/SC scenarios.
    b. Directed address sequence: In this sequence, NWAYS + one TAG/SET combination are chosen randomly at the beginning of the test. All requests are sent to these addresses. The aim is to have many hits with an eviction from time to time.
    c. Random sequence with in a region: The memory partition agent is used to partition the memory space. One region is randomly selected and requests are made within this region.
    d. Congestion sequences: Congestion sequences block memory response model for predefined random number of cycles. Many requests are (load or store) sent, on the same set or different. The idea is to fill write buffer and rtab in the case of write or mshr sets or rtab in the case of load. Please refer to section 2.3 if testplan to get the details of these sequences.

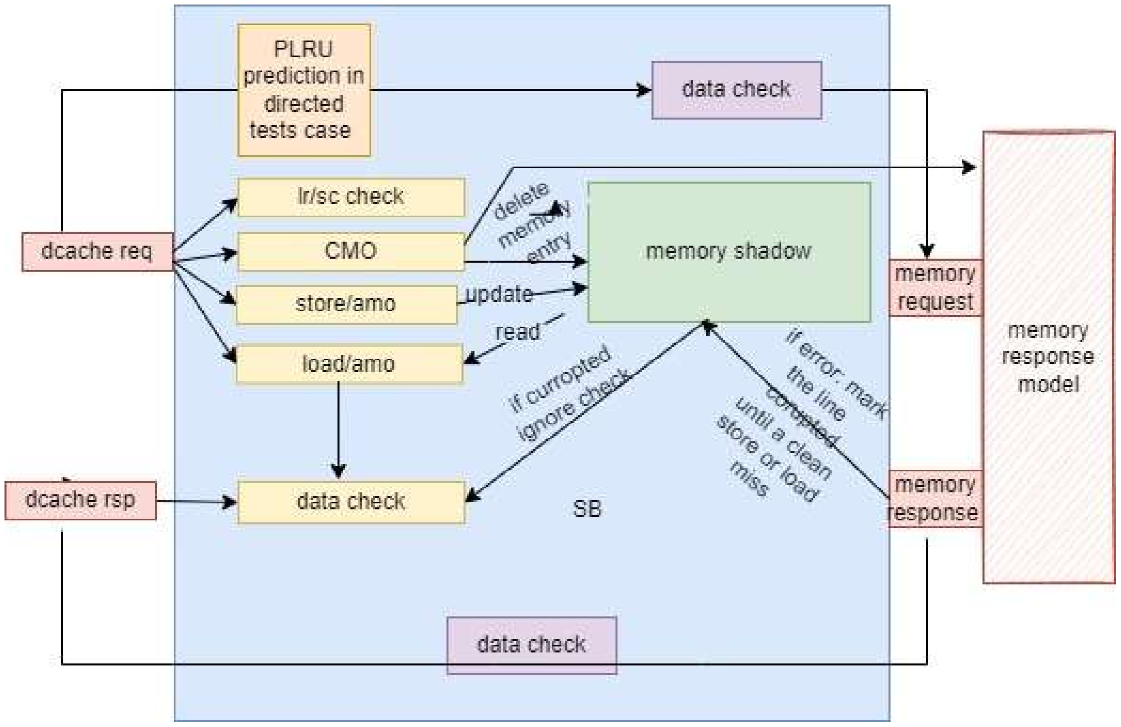
2. Config Agent: This object is used to generate and drive the different configuration variables of HPDcache.
3. AXI2MEM converter: The memory interface of the HPDcache is an AXI inspired interface. The data and meta data channel are independent to one another. This object is used to streamline the data and the meta data to be able to drive it to the memory response model.
4. Memory Agent: It monitors the memory interface of HPDcache.
5. Memory response model: The response model is used to drive the responses of memory requests made by HPDcache. The response model implements following features :
   a. It can be configured to drive the responses in order or out of order.
   b. It drives read and write responses independently
   c. It can be used to send error responses
   d. It supports AMOs
   e. It can be used to insert back pressure

3.1 Environment configuration

See the README.md file under **HPDcache_uvm_tb/testbench/docs/**

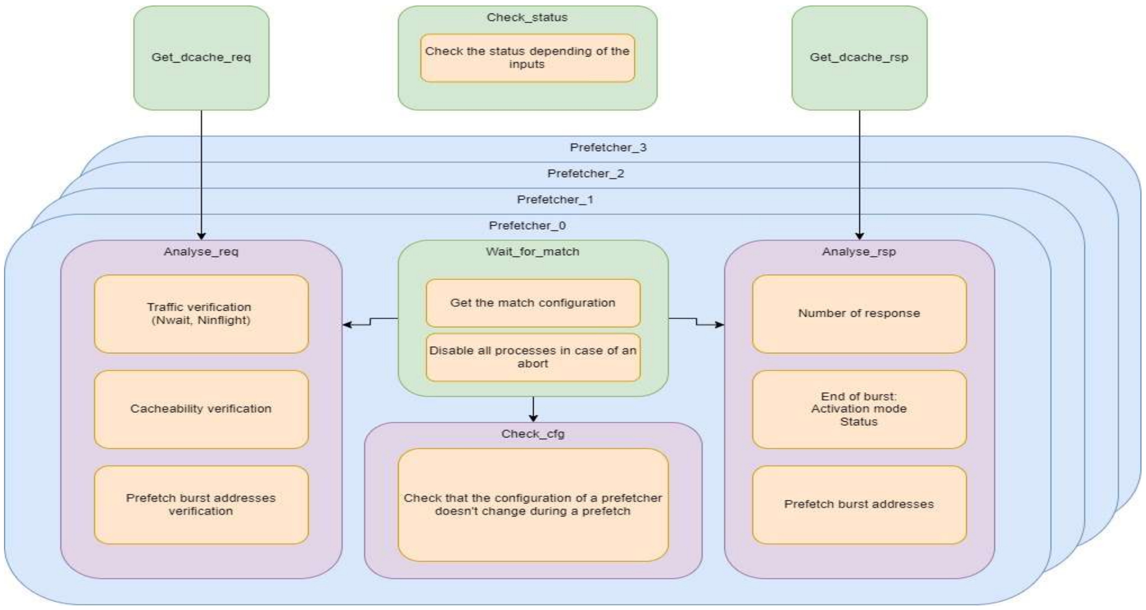
4.1 Scoreboard

1.1.1 Data cache



1. When HPDcache request is received :
    a. LOAD/LR/AMO: memory shadow is read and data is stored in a list (m_load_list)
    b. STORE/SC (pass)/ AMO: memory shadow is updated. In the case of STORE, a merge list is maintained and updated. It is used to correctly predict the write request at memory interface.
    c. CMO: The corresponding entries are removed from the memory response model and shadow memory. We expect a new data from the CACHE now @the same address.
    d. LR/SC check is performed
2. When HPDcache response is received
    a. LOAD/LR/AMO: Data received is checked against the data read from the memory shadow(m_load_list)
    b. An exact prediction of error response is made and verified.
3. Bit-LRU prediction: A special test is considered to verify the LRU algorithm used. In the test, some directed combination (NWAY +1) of set and tags are chosen. They should trigger an eviction from time to time. These requests are made every N number of cycles to avoid request and refill race condition. This way without spying internal signal the LRU algorithm can be verified.
4. Memory request: Memory response model is read or updated.
5. Memory response: Response is sent to data cache. An error is inserted from time to time.

5.1 Prefetcher (not maintained for the moment)


Get_dcache_req
Check_status
Check the status depending of the inputs
Get_dcache_rsp
Prefetcher_3
Prefetcher_2
Prefetcher_1
Prefetcher_0
Analyse_req
Traffic verification (Nwait, Ninflight)
Cacheability verification
Prefetch burst addresses verification
Wait_for_match
Get the match configuration
Disable all processes in case of an abort
Check_cfg
Check that the configuration of a prefetcher doesn't change during a prefetch
Analyse_rsp
Number of response
End of burst: Activation mode Status
Prefetch burst addresses

- The functional verification of the prefetcher is centered around three phases:
    - Detecting a new match to start the corresponding prefetcher
    - Getting and checking new requests/responses
    - Checking the end of prefetch burst features

- Detecting a new match :
    - When a new match is detected (signal match for the prefetcher is driven high and the corresponding prefetcher is enabled), the current configuration is saved in variables
        - Depending of the precedent burst activation mode, the current configuration may be overridden by the previous configuration to follow the activation mode feature.
    - When the configuration for the new burst is okay, three processes are generated: one to check the requests, one to check the responses and the last one to check the stability of the configuration during the prefetch burst
    - When these processes are created, until the prefetch burst ends, the task waits for an abortion: if one happens during a prefetch, all the precedent task are killed, and the wait for another match starts again
- Processing the requests/responses
    - The prefetcher wrapper only has one HPDcache interface shared between all prefetchers. To associate a request/response to its prefetcher, the field TID is used.
    - When a request/response is received by the get_req/rsp tasks, they are stored in an associative array to be treated by the corresponding prefetcher process.
    - When a requests is received, here are the following verification:

        - The address of the request: depending of its position in the prefetch burst, it is possible to calculate the address of the request
        - The traffic controls of the prefetcher: the delay between two requests and the number of on the fly request can be controlled via the configuration, the scoreboard then check that these features are respected during the burst
        - The cacheability of the request: depending of the address of the request and of the cacheability table configuration, the field uncacheable of the transaction is verified.
        - As all transactions from the same prefetcher have the same TID, it is impossible to associate a request with its response. The verification of the error is then not possible. Only the number of responses is checked, which should correspond to the number of requests and burst length.
    - End of burst of prefetch verification
        - End of prefetch verification: checking that the last address of the prefetch corresponds to the formulas in the documentation
        - Check that the outputs of the prefetcher at the end of the prefetch corresponds to its activation mode (prefetcher status and snoop address).

Apart from these three points, the last verification of the prefetcher wrapper status: depending of which prefetcher is activated, busy or aborted, one-task checks that the output status corresponds to the state of all prefetchers.

3. Verification Methodologies

Following methodologies are used to verify the HPDcache subsystem:

1.1 Universal Verification Methodology (UVM)

UVM libraries are used to develop the verification environment. All objects are written in system Verilog. Some of the following points are to be noted:

1. UVM messaging is used to for easy debug (section 6)
2. Objects are built in the build phase
3. Objects and analysis ports are connected to each other in the connect phase
4. Configuration objects are randomize in the end of elaboration phase (test base).
5. Static configurations are done in the configure phase (test base).
6. Sequences are run in the main phase
7. Phase jump is used to do the reset on the fly verification (test base).
8. Cover groups are used to do the coverage

2.1 Assertion Based Verification (ABV)

Assertions are used at different level in the design.

1. Assertions are used within the RTL to verify different properties of the design.
2. Assertions are used in the system Verilog interfaces to do basic verification ex $unknown, $onehot, etc.
3. Assertions are used to cover some functionalities of design.
4. System Verilog based model: bind to following module of HPDcache to do assertion based white box testing
   a. Arbiter
   b. PLRU and TAG directory modelling

4. Simulation

See the README.md file under **HPDcache_uvm_tb/testbench/docs/**

5. Regression

See the README.md file under **HPDcache_uvm_tb/testbench/docs/**

6. Debugging

Following is an example to debug an UVM_ERROR on the *test_HPDcache_multiple_directed_addr* with seed *709346482*. The grep on a UVM_ERROR gives the set and tag, which have an error as follows.

grep UVM_ERROR test_HPDcache_multiple_directed_addr_709346482.log

# UVM_ERROR HPDcache_sb.svh(556) @ 8063 ns: uvm_test_top.env.m_HPDcache_sb [SB HPDCACHE DATA ERROR] ADDR=16b19f744e580(x), ***SET=22(d), TAG=b58cfba27(x)*** *BYTE=0(d) ACC DATA=e1(x) EXP DATA=86(x)*

# UVM_ERROR HPDcache_sb.svh(556) @ 8063 ns: uvm_test_top.env.m_HPDcache_sb [SB HPDCACHE DATA ERROR] ADDR=16b19f744e580(x), ***SET=22(d), TAG=b58cfba27(x)*** *BYTE=0(d) ACC DATA=73(x) EXP DATA=f3(x)*

# UVM_ERROR :   2

Further, a grep on set and tag gives the flow of the data.

```
grep "SET=22(d), TAG=b58cfba27" test_HPDcache_multiple_directed_addr_709346482.log | grep SB | grep UVM_ERROR -
B20
```

```
# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 7894 ns: reporter [SB HPDCACHE REQ 1]
OP=HPDCACHE_REQ_AMO_XOR SID=1(x), TID=53(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=30ff51e7c17b7d3d67dc80fbf245b930(x) BE=ff(x) SIZE=3(x) NEED_RSP=1(x) UNCACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(113) @ 7897 ns: reporter [SB MEM REQ]
ID=80(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x) SIZE=3(d) LEN=0(d),
CMD=HPDCACHE_MEM_ATOMIC ATOMIC=HPDCACHE_MEM_ATOMIC_EOR CACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(128) @ 7899 ns: reporter [SB MEM EXT REQ]
ID=80(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
Data=30ff51e7c17b7d3d67dc80fbf245b93030ff51e7c17b7d3d67dc80fbf245b93030ff51e7c17b7d3d67dc80fbf245b93030ff5
1e7c17b7d3d67dc80fbf245b930(x) BE=ff(x) SIZE=3(d) LEN=0(d), CMD=HPDCACHE_MEM_ATOMIC
ATOMIC=HPDCACHE_MEM_ATOMIC_EOR CACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(89) @ 7902 ns: reporter [SB MEM READ RSP]
ID=80(x), SET=22(d), TAG=b58cfba27(x), WORD=0(x)  ERROR=0(x), LAST=0(x)
DATA=8742ea60050f3a837fd3f1db5f349916899c82edd519d10968357dd75b88910f69462f02a6f3adbd3bea28eab711bd6f0
09f09e91bfdeb87e17f73b37b658b96(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(128) @ 7902 ns: reporter [SB MEM EXT RSP]
ID=80(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
Data=30ff51e7c17b7d3d67dc80fbf245b93030ff51e7c17b7d3d67dc80fbf245b93030ff51e7c17b7d3d67dc80fbf245b93030ff5
1e7c17b7d3d67dc80fbf245b930(x) BE=ff(x) SIZE=3(d) LEN=0(d), CMD=HPDCACHE_MEM_ATOMIC
ATOMIC=HPDCACHE_MEM_ATOMIC_EOR CACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 7903 ns: reporter [SB HPDCACHE RSP]
OP=HPDCACHE_REQ_AMO_XOR SID=1(x), TID=53(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=9f09e91bfdeb87e17f73b37b658b96(x) BE=ff(x) SIZE=3(x) NEED_RSP=1(x) UNCACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 8016 ns: reporter [SB HPDCACHE REQ 1]
OP=HPDCACHE_REQ_LOAD SID=1(x), TID=54(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=ac4200ec07b7662a975873830f25ae32(x) BE=d(x) SIZE=2(x) NEED_RSP=0(x) UNCACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 8035 ns: reporter [SB HPDCACHE REQ 1]
OP=HPDCACHE_REQ_AMO_MAX SID=1(x), TID=4c(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=fac95c55a9bdf9f1eaeef173217a9670(x) BE=f(x) SIZE=2(x) NEED_RSP=1(x) UNCACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(113) @ 8038 ns: reporter [SB MEM REQ]
ID=80(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x) SIZE=2(d) LEN=0(d),
CMD=HPDCACHE_MEM_ATOMIC ATOMIC=HPDCACHE_MEM_ATOMIC_SMAX CACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(128) @ 8040 ns: reporter [SB MEM EXT REQ]
ID=80(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
Data=fac95c55a9bdf9f1eaeef173217a9670fac95c55a9bdf9f1eaeef173217a9670fac95c55a9bdf9f1eaeef173217a9670fac95
c55a9bdf9f1eaeef173217a9670(x) BE=f(x) SIZE=2(d) LEN=0(d), CMD=HPDCACHE_MEM_ATOMIC
ATOMIC=HPDCACHE_MEM_ATOMIC_SMAX CACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(89) @ 8042 ns: reporter [SB MEM READ RSP]
ID=80(x), SET=22(d), TAG=b58cfba27(x), WORD=0(x)  ERROR=0(x), LAST=0(x)
DATA=8742ea60050f3a837fd3f1db5f349916899c82edd519d10968357dd75b88910f69462f02a6f3adbd3bea28eab711bd6f0
09f09e91bfdeb8786a3f348892032a6(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(128) @ 8042 ns: reporter [SB MEM EXT RSP]
ID=80(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
Data=fac95c55a9bdf9f1eaeef173217a9670fac95c55a9bdf9f1eaeef173217a9670fac95c55a9bdf9f1eaeef173217a9670fac95
c55a9bdf9f1eaeef173217a9670(x) BE=f(x) SIZE=2(d) LEN=0(d), CMD=HPDCACHE_MEM_ATOMIC
ATOMIC=HPDCACHE_MEM_ATOMIC_SMAX CACHEABLE=0(x)
```

```
# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 8045 ns: reporter [SB HPDCACHE RSP]
OP=HPDCACHE_REQ_AMO_MAX SID=1(x), TID=4c(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=9f09e91bfdeb8786a3f348892032a6(x) BE=f(x) SIZE=2(x) NEED_RSP=1(x) UNCACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 8062 ns: reporter [SB HPDCACHE REQ 1]
OP=HPDCACHE_REQ_LOAD SID=1(x), TID=73(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=a3c6d2b301a03f80f1fac4ade93a421d(x) BE=93a5(x) SIZE=4(x) NEED_RSP=1(x) UNCACHEABLE=0(x)

# UVM_INFO <path_to_tb>testbench/common/HPDcache_common_pkg.sv(58) @ 8063 ns: reporter [SB HPDCACHE RSP]
OP=HPDCACHE_REQ_LOAD SID=1(x), TID=73(x), ADDR=16b19f744e580(x) SET=22(d), TAG=b58cfba27(x), WORD=0(x)
DATA=9f09e91bfdeb87e17f73b3217a9670(x) BE=93a5(x) SIZE=4(x) NEED_RSP=1(x) UNCACHEABLE=0(x)

# UVM_ERROR HPDcache_sb.svh(556) @ 8063 ns: uvm_test_top.env.m_HPDcache_sb [SB HPDCACHE DATA ERROR]
ADDR=16b19f744e580(x), SET=22(d), TAG=b58cfba27(x) BYTE=0(d) ACC DATA=e1(x) EXP DATA=86(x)

# UVM_ERROR HPDcache_sb.svh(556) @ 8063 ns: uvm_test_top.env.m_HPDcache_sb [SB HPDCACHE DATA ERROR]
ADDR=16b19f744e580(x), SET=22(d), TAG=b58cfba27(x) BYTE=0(d) ACC DATA=73(x) EXP DATA=f3(x)
```