# DCache Interface DRAC specification version v0.2

Rubén Langarita Benítez

June 16, 2020

# Contents

# 1 General purpose of the module

*Person in Charge; Victor Soria*

The *Data Cache Interface* module is placed in the top module alongside the datapath and the Instruction Cache Interface. It is responsible of producing the necessary signals to control the data cache.

This module receives signals from the datapath and the response from the data cache, and it outputs the following signals:

- Request to data cache.
- Response to the CPU.

# 2 Design placement

The Data Cache Interface module is placed inside the top module. There is only one instance per core. We can differentiate two type of signals: the ones that are connected to the datapath and the ones that are connected to the data cache.

# 3 Parameters

All parameters, enums and types used in this module are defined in drac_pkg or risc_pkg.

# 4 Interface

Beside the clock (*clk_i*) and the reset (*rstn_i*) signals, we can differentiate two type of signals, the ones going to the datapath and the ones going to the data cache. To communicate with the datapath we use the following:

| **Signal name** | **Width** | **Type** | **Description** |
|---|---|---|---|
| req_cpu_dcache_i | struct | in | Signals from the CPU |
| resp_dcache_cpu_o | struct | out | Response to the CPU |

*resp_dcache_cpu_o* structure is explined in more detail in Section 7. The signals of the *req_cpu_dcache_i* structure are the following:

| Signal name | Width | Type | Description |
|---|---|---|---|
| io_base_addr | 40 | in | Lower limit of the address |
| kill | 1 | in | Kill current operation |
| valid | 1 | in | Sending valid operation |
| instr_type | enum | in | Instruction type: load, store or AMO |
| data_rs1 | 64 | in | For address calculation |
| data_rs2 | 64 | in | Data to store (only for stores) |
| data_rd | 64 | in | To build the tag (currently useless) |
| imm | 64 | in | For address calculation |
| mem_size | 3 | in | Size of the request (byte, halfword, word) |

(notice that signals to data cache are not gathered in a structure):

| Signal name | Width | Type | Description |
|---|---|---|---|
| dmem_resp_replay_i | 1 | in | Miss ready |
| dmem_resp_data_i | 64 | in | Readed data from Cache |
| dmem_req_ready_i | 1 | in | Dcache ready to accept request |
| dmem_resp_valid_i | 1 | in | Response is valid |
| dmem_resp_nack_i | 1 | in | Readed data from Cache |
| dmem_xcpt_ma_st_i | 1 | in | Missaligned store |
| dmem_xcpt_ma_ld_i | 1 | in | Missaligned load |
| dmem_xcpt_pf_st_i | 1 | in | DTLB miss on store |
| dmem_xcpt_pf_ld_i | 1 | in | DTLB miss on load |
| dmem_req_valid_o | 1 | out | Sending valid request |
| dmem_req_cmd_o | 5 | out | Type of memory access |
| dmem_req_addr_o | 40 | out | Address of memory access |
| dmem_op_type_o | 4 | out | Granularity of memory access |
| dmem_req_data_o | 64 | out | Data to store |
| dmem_req_tag_o | 8 | out | Tag for the MSHR |
| dmem_req_invalidate_lr_o | 1 | out | Reset load-reserved/store-conditional |
| dmem_req_kill_o | 1 | out | Kill actual memory access |

## 5 Reset behaviour

The reset signal *rstn_i* works on a negative edge. The outputs will be 0 and if some operation is being executed, it will be stopped.

## 6 What could not happen

There are some invalid combinations of signals. If the datapath is sending a valid request (*req_cpu_dcache_i.valid*), the remaining signals are correct: *instr_type*, *data_rs1*, *imm*...

The state machine has 4 states: *ResetState*, *Idle*, *MakeRequest*, and *WaitResponse*. It can not move outside these states.

## 7 Behavior

For each output signal, the behavior should be the following:

- **resp_dcache_cpu_o:** this structure is used to send signals to the datapath and it has 8 signals:
  - **lock**: the data cache is busy, and the datapath should be locked until the data cache finishes.
  - **ready**: the data cache is ready to send the response.
  - **data**: data to serve the load requests.
  - **xcpt_{ma,pf}_{st,ld}**: 4 signals to indicate exceptions. *ma* is misaligned, *pf* is DTLB miss, *st* is store, and *ld* is load.
  - **addr**: address of the exception.
- **dmem_req_valid_o:** sending a valid request to data cache.
- **dmem_req_cmd_o:** type of the memory access: load, store, load-link, store-conditional, and atomic memory operations.
- **dmem_req_addr_o:** address of the request.
- **dmem_op_type_o:** granularity of the memory access: byte, halfword, and word.
- **dmem_req_data_o:** data to store.
- **dmem_req_tag_o:** tag of the memory access, only for multi-processor.
- **dmem_req_invalidate_lr_o:** reset load-link/store-conditional transaction.
- **dmem_req_kill_o:** kill actual memory access.

## 8 Special cases, corner cases

If the datapath is sending from the execution stage a valid request, and at the same time other stage is flushing the pipeline, the interface should cancel the request.