Embedded Systems 1 - 095906

mor1kx - Embedded CPU

Write Back Cache Implementation Reference Manual

Francesco Maio
Giovanni Scotti

Advisor: *Davide Zoni*

Academic Year 2016/2017

Document version: 1.0

Contents

Section 1

Introduction

1.1 Purpose

The Write Back Cache Implementation Reference Manual for the mor1kx embedded processor is intended to describe how the baseline architecture has been modified in order to provide the cache's write policy known as *write-back*.

This document aims to supply an adequate knowledge to those developers and hardware designers, who have to change, adjust and enhance for any reason the related Verilog HDL code, making the understanding of what has been implemented much easier.

1.2 Scope

The mor1kx is an open source, Verilog implementation that is fully compliant with the OpenRisc architecture. In its *Cappuccino* fashion, it offers a six stages pipeline with *write-through* L1 caches.

The following sections deal with all the adjustments required to offer the more aggressive *write-back* writing policy with respect to the *Cappuccino* pipeline only.

1.3 Definitions, Acronyms, Abbreviations

32-bit block/word: the unit of data contained in the cache block.

a.k.a: Also Known As.

Cache block: the basic unit for cache storage. It may contain multiple bytes/words of data. The minimum data size is assumed to be 32 bits in the whole documentation.

Cache line: same as cache block.

Cache set: a "row" in the cache. It is made of cache blocks, whose number is determined by the cache layout.

HDL: Hardware Description Language.

LSU: Load and Store functional Unit.

Section 2

Baseline Architecture

In this section the original operations of the mor1kx, concerning the write policy of the data cache, are briefly described to better understand which Verilog modules need to be modified and why.

2.1 Overview

The mor1kx is an OpenRISC 1000 compliant processor which is highly configurable and contains multiple pipeline implementations. In particular, the *Cappuccino* one is based on the following stages:

- **Address:** the pipeline branch control unit evaluates branches indicated by the control stage. Then, the block outputs to the fetch stage indications about the occurrence of a branch and the target address;
- **Fetch:** the fetch stage is tightly coupled with the instruction cache;
- **Decode:** it generates ALU, LSU and control operation signals;
- **Execute:** it performs all the arithmetic and logical operations supported by the ORBIS32 instruction set;
- **Control/Memory:** it mainly involves a load and store unit which is coupled with a store buffer and a data cache;
- **Write Back:** this is the writeback stage whose inputs are ALU and LSU results and SPR values.

As far as this document is concerned, the focus is basically set on the memory stage, in particular on the LSU and the data cache.

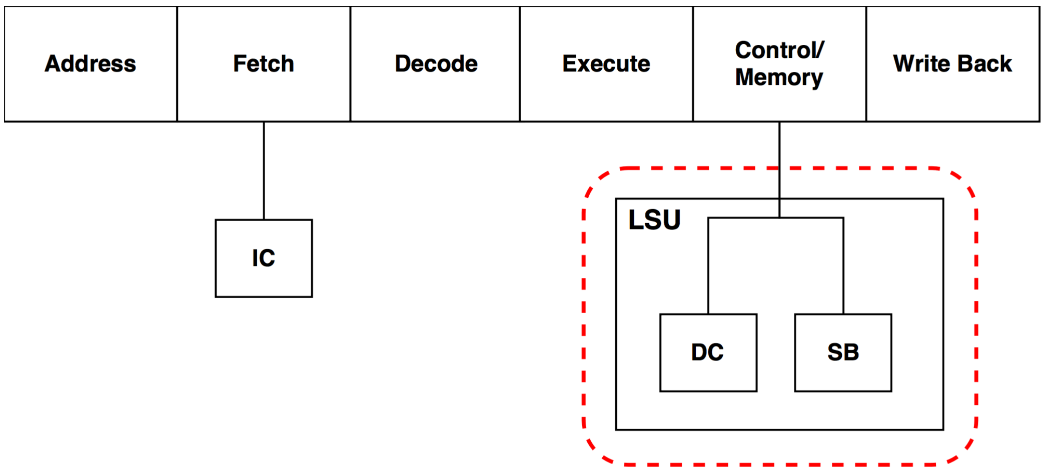

Figure 2.1: Schema of the Cappuccino pipeline with the main focus of the documentation highlighted in red.

2.2 Write Through Implementation

The followings are the relevant features offered before the changeover:

- ***write-through* write policy:** when a write occurs, data is written both into data cache and main memory;
- **no-write allocate:** on a write miss, data is directly written in main memory bypassing the data cache;
- **store buffer:** if enabled, during a store operation data is not only sent to the data cache, but also to the store buffer (a.k.a. write buffer). It is useful because bursts of writes are common. Moreover, it might prevent the processor from stalling. Data are then moved to main memory.

The flow diagram in Figure 2.2 depicts the series of procedures accomplished by the original implementation of the processor during read and write operations.

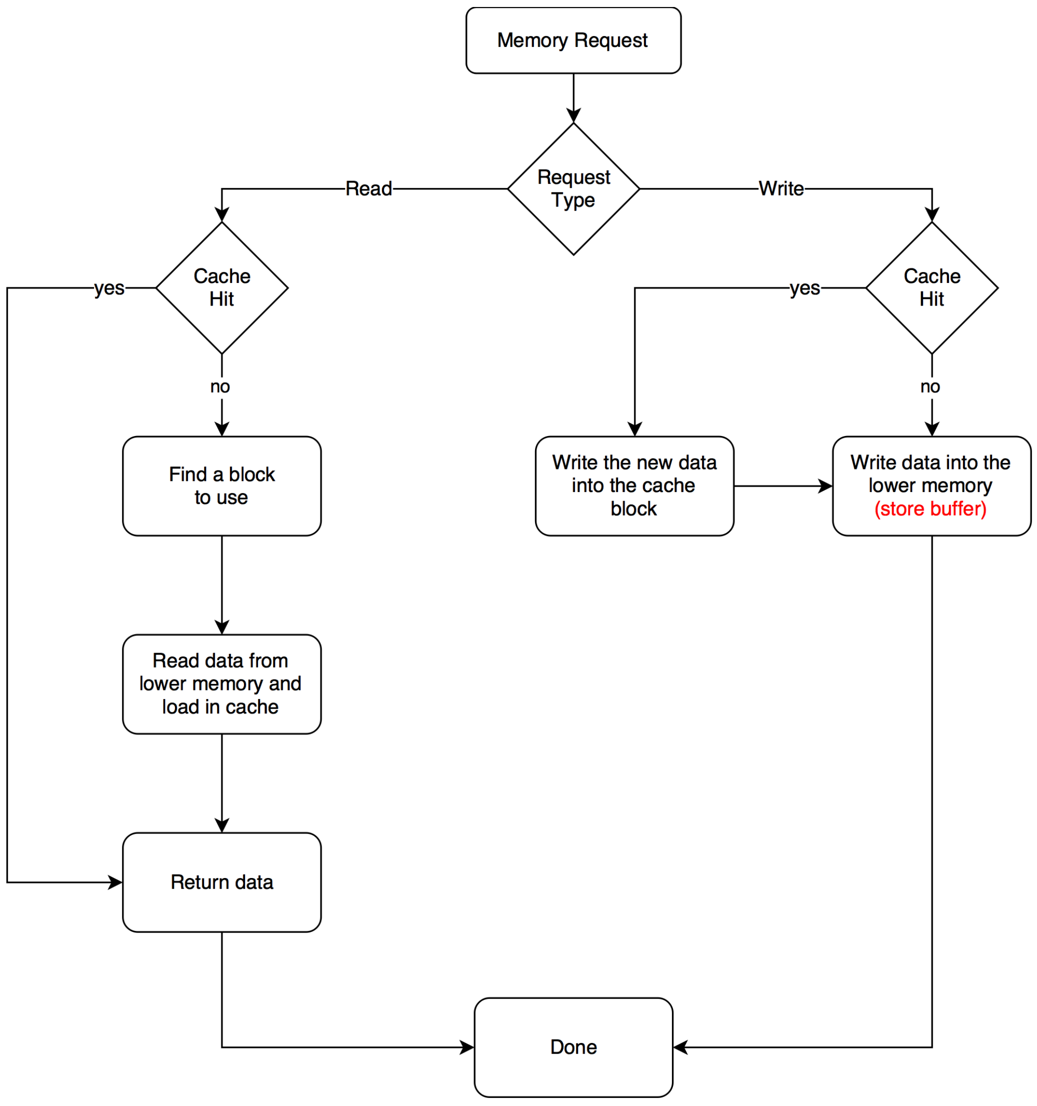
Memory Request
Request Type
Read
Write
Cache Hit
yes
no
Cache Hit
yes
no
Find a block to use
Write the new data into the cache block
Write data into the lower memory (store buffer)
Read data from lower memory and load in cache
Return data
Done

Figure 2.2: Flow diagram showing the original mor1kx *write-through* working principle.

2.3 System Configurations

Different system configurations can be achieved by enabling and disabling the data cache and/or the store buffer. Both the modules are generated in the scope of the LSU module.

Config.	**Data Cache**	**Store Buffer**
1	Enabled	Enabled
2	Enabled	None
3	None	Enabled
4	None	None

Table 2.1: Potential system configurations of the original mor1kx implementation. The data cache and the store buffer are generated within the LSU *Cappuccino* module.

Section 3

Write Back Implementation

3.1 Overview

In the *write-through* policy, each time a write request occurs, data are saved into both the cache and the main memory. This involves multiple bus requests, which need arbitration and may result in a loss of performance.

On the other hand, in the *write-back* policy with write allocate, data are written into the cache anytime a change occurs, but written in the corresponding location in main memory only under certain conditions.

Going into more details, there is the need to figure out whether a cache block has been modified or not. Such a control can be easily achieved by taking advantage of dirty bits. If they are asserted, the cache has a more updated version of data with respect to the main memory and has to move those data to main memory itself before refilling the cache block.

As can be observed from the following flow diagram, all the brand-new logic works in case of cache miss and the overall design gets complicated.

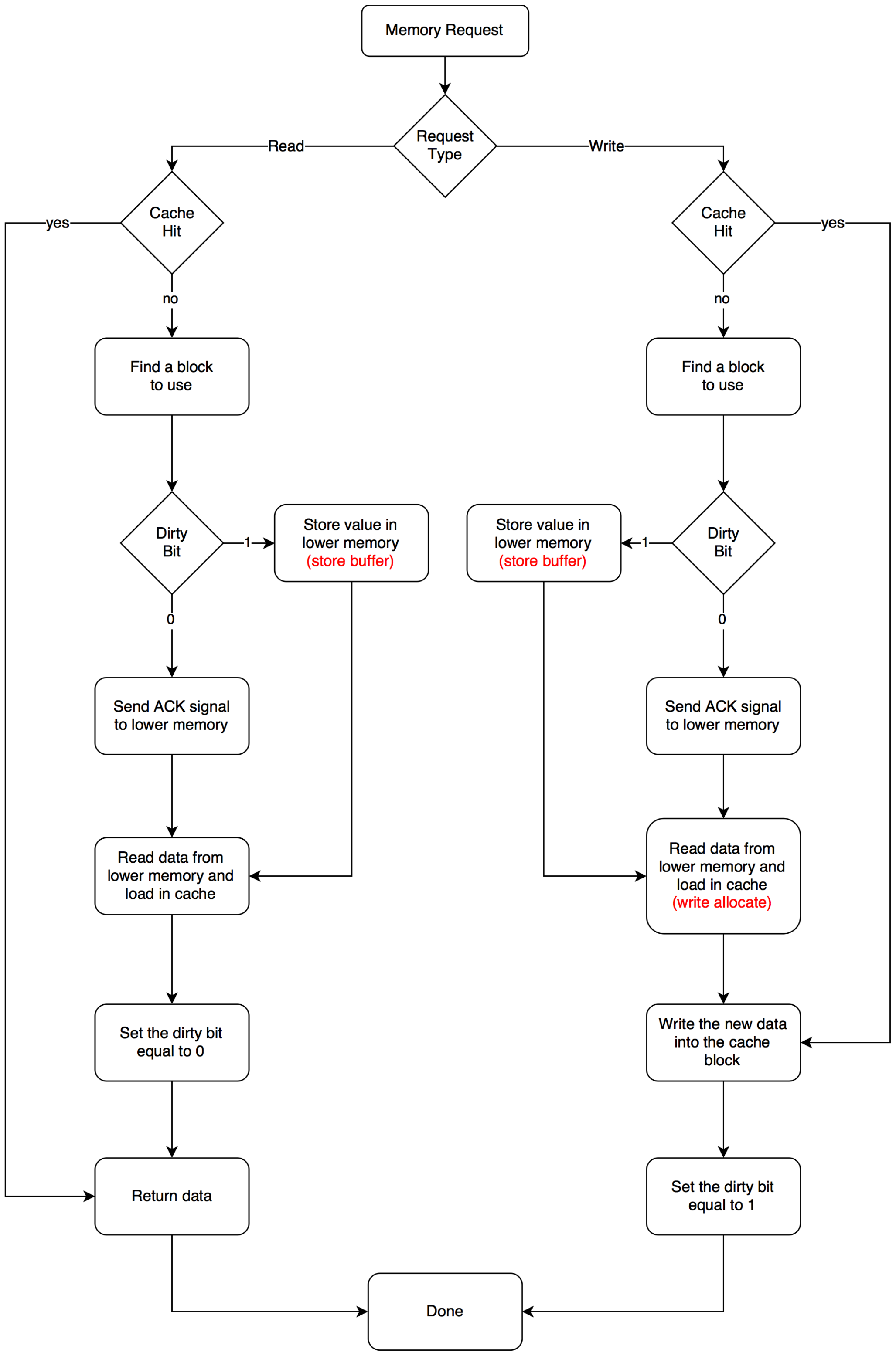
Memory Request
Request Type
Read
Write
Cache Hit
yes
no
Find a block to use
Dirty Bit
1
0
Store value in lower memory (store buffer)
Send ACK signal to lower memory
Read data from lower memory and load in cache
Set the dirty bit equal to 0
Return data
Cache Hit
yes
no
Find a block to use
Dirty Bit
1
0
Store value in lower memory (store buffer)
Send ACK signal to lower memory
Read data from lower memory and load in cache (write allocate)
Write the new data into the cache block
Set the dirty bit equal to 1
Done

Figure 3.1: Flow diagram showing the *write-back* working principle.

Dirty Bits

Per each cache block, there exists a sequence of bits equal in number to the 32-bit blocks inside the cache line. Each dirty bit is set to one if the respective word has been modified during the lifetime of that line.

The cache line is considered clean if all the dirty bits are clear, while it is considered dirty when at least one dirty bit is set to one. Whenever a cache line is replaced, all the dirty bits are set to zero.

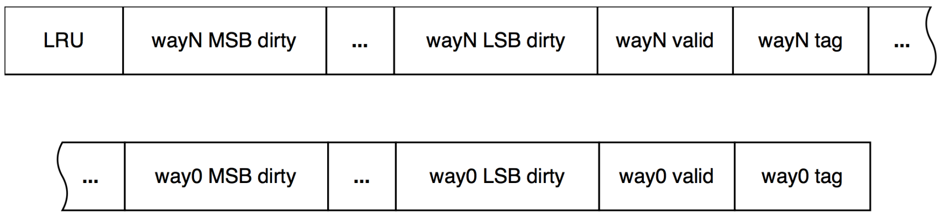

Figure 3.2: Tag memory layout with the dirty bits implementation.

3.2 Modes Of Operation

Multiple systems configurations are allowed by enabling and disabling the data cache and/or the store buffer. Table 3.1 sums them up.

Config.	Data Cache	Store Buffer
1	Enabled	X
2	None	Enabled
3	None	None

Table 3.1: 1. When the data cache is enabled, the store buffer is always instantiated and works as a temporary storage between cache and memory. 2,3. If the data cache is disabled, the store buffer can be enabled or not and the functioning is the same as in the baseline architecture. X = don't care.

3.3 Data Cache FSM

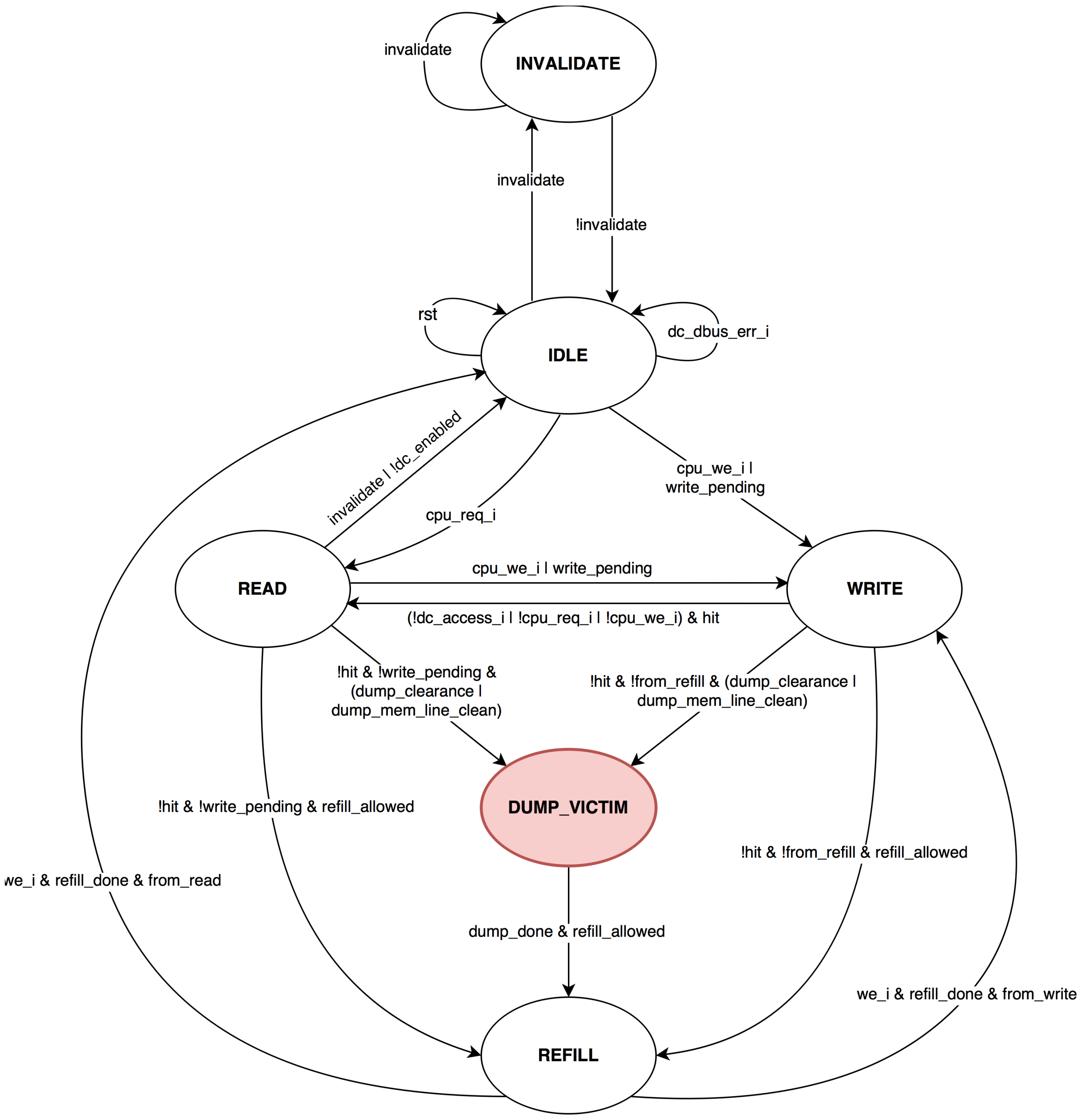

Figure 3.3: Finite state machine of the data cache.

3.4 Load And Store Unit FSM

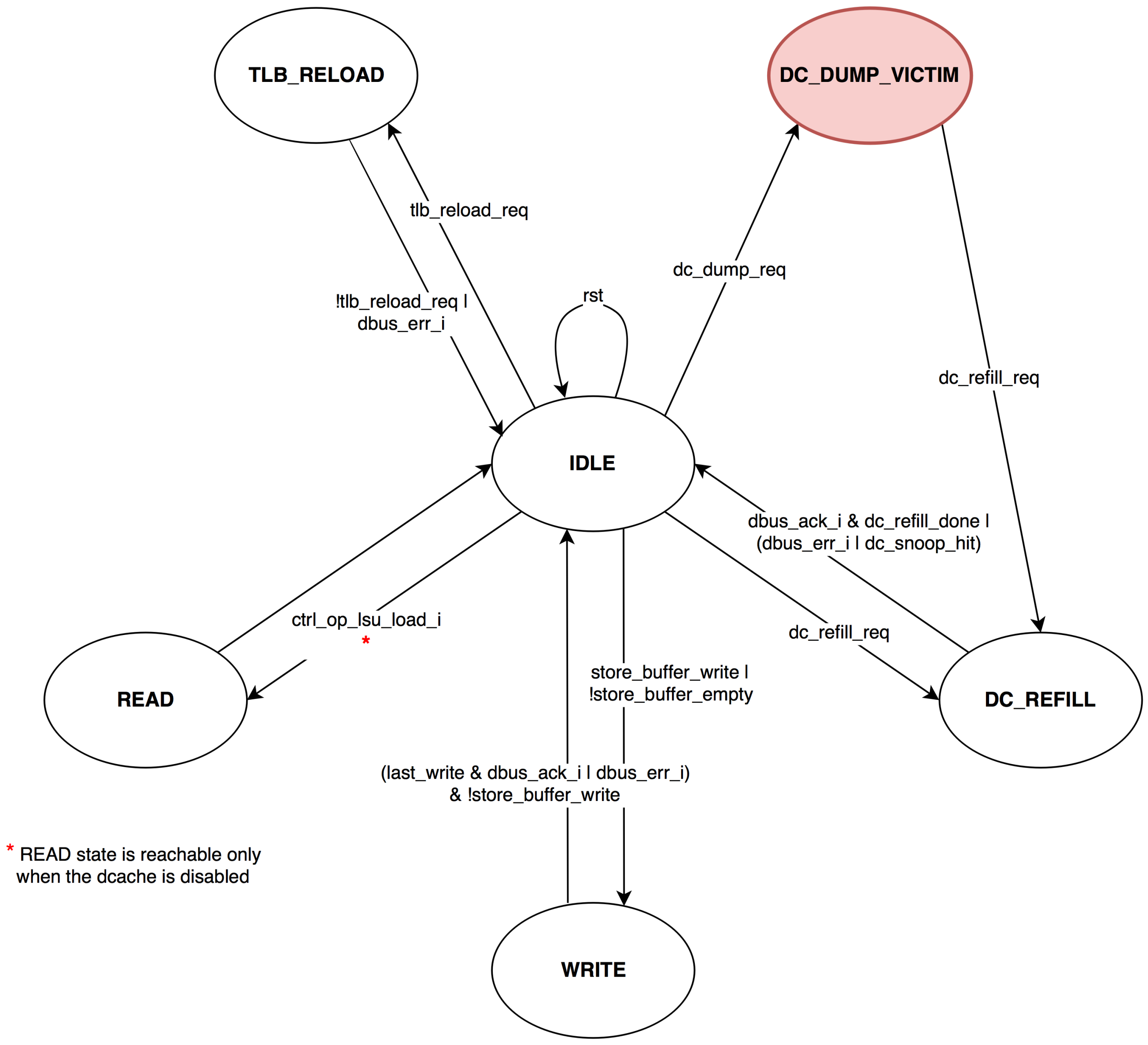
TLB_RELOAD
DC_DUMP_VICTIM
tlb_reload_req
!tlb_reload_req | dbus_err_i
rst
dc_dump_req
dc_refill_req
IDLE
dbus_ack_i & dc_refill_done | (dbus_err_i | dc_snoop_hit)
ctrl_op_lsu_load_i *
dc_refill_req
store_buffer_write | !store_buffer_empty
READ
DC_REFILL
(last_write & dbus_ack_i | dbus_err_i) & !store_buffer_write
* READ state is reachable only when the dcache is disabled
WRITE

Figure 3.4: Finite state machine of the load and store unit.

Note that in the DC_DUMP_VICTIM state, LSU waits for the store buffer to be empty before proceeding with the refill procedure. When the data cache is enabled, the WRITE state could be exploited to introduce a further optimization: as soon as the evicted cache line is saved in the store buffer, the LSU might proceed with its operations and empty the buffer later on.

3.5 Data Cache - LSU Interface

The data cache communicates actively with the LSU to manage the different operations carried out in the new `DUMP_VICTIM` state. Table 3.2 describes how the brand new cache signals are mapped to LSU signals and their purpose.

Data Cache	LSU	Description
dump_dat_o	dc_dump_dat	Data to be copied in the store buffer.
dump_adr_o	dc_dump_adr	Memory address of the data to be copied.
dump_req_o	dc_dump_req	Control signal that drives the LSU in the `DC_DUMP_VICTIM` state.
dump_done_o	dc_dump_done	Control signal asserted when the last 32-bit block to be saved is moved to the store buffer.
dump_mem_line_clean_o	dc_mem_line_clean	Control signal asserted when a memory acknowledgement dump operation takes place.
cpu_ack_o	dc_ack	Control signal asserted when either the cache returns data to be read to the LSU or a cache hit write happens.
dump_write_valid_o	dc_write_valid	This signal drives the `store_buffer_write` signal in order to save in the store buffer only the dirty 32-bit blocks.

Table 3.2: Mapping of cache interface signals on LSU signals, together with their description.

3.6 Data Cache `DUMP_VICTIM` State

This section aims to describe in details all the operations performed by the data cache in the `DUMP_VICTIM` state.

Before entering the `DUMP_VICTIM` state from either the `READ` or the `WRITE` one, the cache carries out the following operations:

- it prepares refill and dump related signals such as `refill_valid` and `dump_valid`;
- it sets specific flags in order to remember if `REFILL` will be accessed from `READ` or `WRITE`. In the latter case, the `WRITE` state will be entered

after refilling and the pending write operation will be carried out as a normal cache hit;

- it starts building the address to send to the store buffer together with the related 32-bit blocks of the cache line that needs to be refilled.

The `DUMP_VICTIM` state is reached only in case of cache miss and if the `(dump_clearance | dump_mem_line_clean)` condition is satisfied.

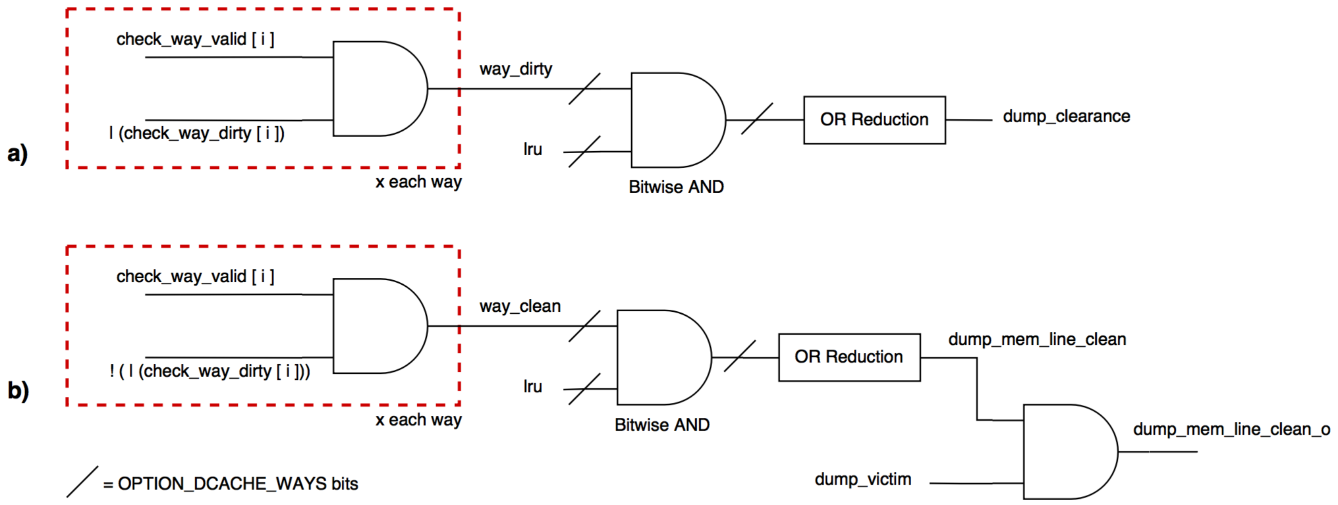

Figure 3.5: Generation of `dump_clearance` (a) and `dump_mem_line_clean` (b) control signals.

The dump procedure will be different depending on which of the mentioned-above control signals is asserted. They are set in mutual exclusion. The outcome is described by Table 3.3.

Control Signal	**Dump Procedure**
dump_clearance	The whole dirty cache line is sent to the store buffer in order to be copied in main memory.
dump_mem_line_clean	Only the first 32-bit block of the clean cache line is sent to the store buffer. This is a memory acknowledgment.

Table 3.3: Description of the dump procedure depending on the asserted control signal.

The transfer of the whole cache line to the store buffer requires a number of clock cycles equal to the quantity of 32-bit blocks in the cache line itself.

Blocks are moved sequentially from the one with the lowest address to the one with the highest.

Subsequently, the data cache waits for the store buffer to be empty before entering the `REFILL` state and replace the target cache block.

Please note that the data cache and the LSU move to the corresponding dump and refill states concurrently thanks to the `dump_req_o` and `refill_req_o` control signals.

3.7 LSU `DC_DUMP_VICTIM` State

This section contains the details of the operations performed by the LSU during the `DC_DUMP_VICTIM` state.

The load and store unit must manage and coordinate the writes and reads in both the cache and the main memory. To this extent, it takes advantage of a store buffer: basically, a FIFO queue in charge of storing couples "address-data", that are eventually placed on the bus. The buffer works in parallel with the cache, therefore it starts requesting the bus without waiting for the whole cache line to be copied in there.

As soon as the `dc_dump_req` signal is asserted by the cache, the LSU passes into the `DC_DUMP_VICTIM` state simultaneously with the data cache itself. On the top of that, the LSU manages the control signals related to the data bus and writes into the store buffer. When the dump ends and a refill is therefore needed, the `dc_refill_req` is asserted and the new state will be `DC_REFILL`.

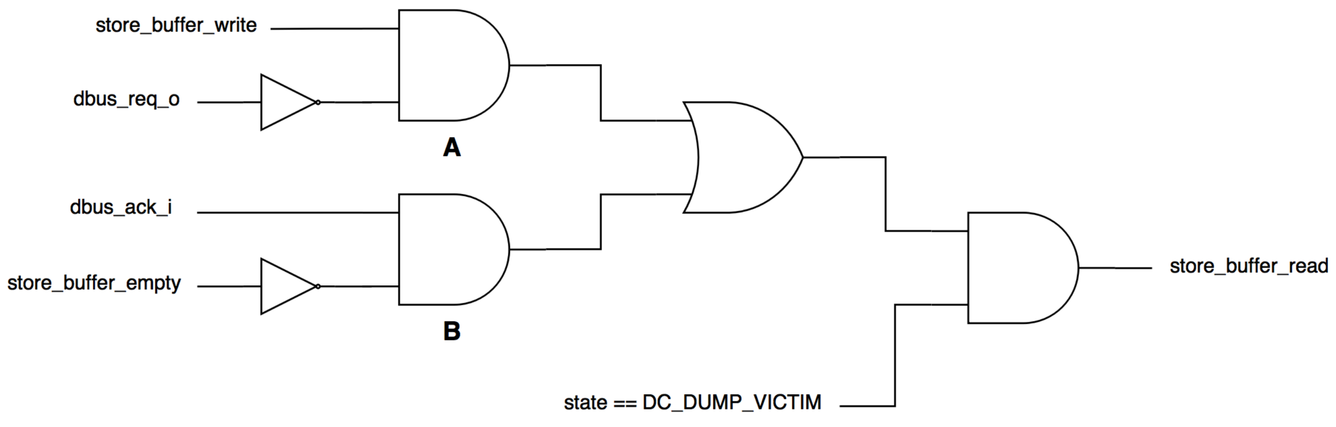

Figure 3.6: Combinational logic for controlling the reads in the store buffer.

The `store_buffer_read` signal is so structured in order to be asserted the first time a 32-bit block is copied from the cache to the store buffer (AND **A** in Figure 3.6). From now on, each time the LSU receives an acknowledgment from the data bus, the read signal is asserted again, until the store buffer will be empty (AND **B** in Figure 3.6).

During the `DC_DUMP_VICTIM` state, the address and the data to carry out a write operation in main memory come from the store buffer and are directly muxed to the data bus interface signals. In order to actually write data, two control signals must be asserted: `dbus_req_o` and `dbus_we`. The bus takes into account the request and when the write is completed, the `dbus_ack_i` signal is asserted.

`dbus_req_o` and `dbus_we` are set one clock cycle after `store_buffer_read`. This is necessary because of the synchronous dual port ram of the store buffer and the delay introduced to retrieve data from it.

3.8 Further Optimizations

The *write-back* cache implementation includes some interesting optimizations.

3.8.1 Optimized Data Cache Data Dump

As described in the above sections, especially in Section 3.1, each 32-bit block inside the cache line is associated with a dirty bit. This feature can be exploited in order to evict and send to the store buffer, located in the LSU, only those 32-bit blocks that have been modified during their stay in the cache line. In this way, the overall bus traffic will be greatly reduced.

3.8.2 Memory Ack As A Coherence Protocol Support

The mor1kx is a single core embedded processor, but the baseline architecture might be extended in order to support multiple cores.

Anytime a miss occurs and a clean cache block is replaced, the `dump_mem_line_clean_o` signal of the data cache module is asserted as thoroughly described in the above sections. The signal can be easily used to fulfill a memory coherence protocol when the system needs any kind of coherent memory.

3.9 Architectural Change

In order to change the behaviour of the cache and implement the *write-back* policy, only two modules need to be modified:

mor1kx_dcache.v: this is the data cache module. The major changes deal with the sequential/combinational internal FSM, new interfacing signals and new structure of the tag ram;

mor1kx_lsu_cappuccino.v: this is the load and store unit of the pipeline. Major changes involve the internal FSM as well as the store buffer. The latter is used to store the evicted cache line that needs to be copied into memory before being replaced during refilling.