

## Key Design Features

- Synthesizable, technology independent VHDL Core
- Fully pipelined non-stalling architecture
- Direct-mapped organization
- Cache Flush functionality and cache done flag to indicate the end of a flush sequence
- Register or RAM-based storage<sup>1</sup>
- Configurable address width, word size, line size and number of cache lines
- Fully configurable FIFO buffering to hide the latency of a memory access
- Fully configurable FIFO buffering on all input and output interfaces
- Cache performance metrics: hit/miss flags and buffer full/empty signalling
- Simple valid/ready pipeline protocol on all interfaces
- 8 cycle cache hit latency

## Applications

- Level-2 or Level-3 caches where memory writes are not necessary or infrequent
- Processor instruction caches
- Pixel caches for graphics pipelines

## Block Diagram

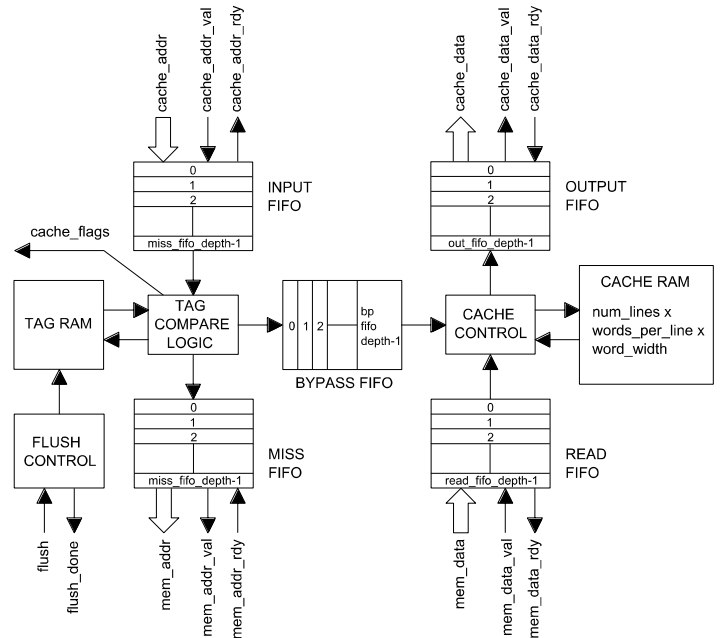


Figure 1: Direct-mapped cache architecture

## Pin-out Description

### SYSTEM SIGNALS AND CACHE CONTROL

Pin name	I/O	Description	Active state
clk	in	Synchronous clock	rising edge
reset	in	Asynchronous reset	low
flush	in	Initiate a cache flush	high
flush_done	out	Cache flush finished flag	high
cache_flags [1:0]	out	Bit 0 : Cache miss flag Bit 1 : Cache hit flag	high
fifo_flags [9:0]	out	Bit 0 : Input FIFO full Bit 1 : Input FIFO empty Bit 2 : Bypass FIFO full Bit 3 : Bypass FIFO empty Bit 4 : Miss FIFO full Bit 5 : Miss FIFO empty Bit 6 : Read FIFO full Bit 7 : Read FIFO empty Bit 8 : Output FIFO full Bit 9 : Output FIFO empty	high

<sup>1</sup> Type of inferred storage depends on synthesis tool configuration.

**INTERFACE WITH CACHE MEMORY**

Pin name	I/O	Description	Active state
cache_addr [addr_width-1:0]	in	Cache memory read address	address
cache_addr_val	out	Cache memory read address valid	high
cache_addr_rdy	out	Cache memory read address ready	high
cache_data [word_width-1:0]	out	Cache memory read data	data
cache_data_val	out	Cache memory read data valid	high
cache_data_rdy	in	Cache memory read data ready	high

**CACHE INTERFACE WITH EXTERNAL MEMORY**

Pin name	I/O	Description	Active state
mem_addr [addr_width - log2_words_per_line-1:0]	out	External memory address	address
mem_addr_val	out	External memory address valid	high
mem_addr_rdy	in	External memory address ready	high
mem_data [word_width * words_per_line-1:0]	in	External memory read data	data
mem_data_val	in	External memory read data valid	high
mem_data_rdy	out	External memory read data ready	high

**Generic Parameters**
**CACHE SPECIFICATION**

Generic name	Description	Type	Valid range
addr_width	Memory address width	integer	$> (\log_2 \text{num\_lines} + \log_2 \text{words\_per\_line}) + 2$
word_width	Memory word width	integer	$\geq 8$ (must be multiple of 8)
words_per_line	No. of words in a cache line	integer	$\geq 2$ (must be power of 2)
log2_words_per_line	Log2 no. of words per line	integer	Log2 (words_per_line)
num_lines	No. of cache lines	integer	$\geq 16$ (must be multiple of 8)
log2_num_lines	Log2 no. of cache lines	integer	Log2 (num_lines)

**BUFFER CONFIGURATION**

Generic name	Description	Type	Valid range
bp_fifo_depth	Bypass FIFO depth	integer	$\geq 2$
bp_fifo_depth_log2	Log2 depth of Bypass FIFO	integer	Log2 (bp_fifo_depth)
in_fifo_depth	Input FIFO depth	integer	$\geq 2$
in_fifo_depth_log2	Log2 depth of Input FIFO	integer	Log2 (in_fifo_depth)
out_fifo_depth	Output FIFO depth	integer	$\geq 4$
out_fifo_depth_log2	Log2 depth of Output FIFO	integer	Log2 (out_fifo_depth)
miss_fifo_depth	Miss FIFO depth	integer	$\geq 2$
miss_fifo_depth_log2	Log2 depth of Miss FIFO	integer	Log2 (miss_fifo_depth)
read_fifo_depth	Read FIFO depth	integer	$\geq 2$
read_fifo_depth_log2	Log2 Read FIFO depth	integer	Log2 (read_fifo_depth)

**General Description**

CACHE\_DIRECT\_MAP is a fully generic direct-mapped read cache. It has a fully pipelined architecture and permits consecutive hits and misses to be serviced sequentially without stalling<sup>2</sup>. A cache hit has a nominal latency of 8 clock cycles. A cache miss has a latency of 10 clock cycles plus the latency of the memory access.

All interfaces with the cache share a common valid/ready pipeline protocol. Data transfer occurs on a rising clock-edge when *val* is high and *rdy* is high.

**FIFO buffering**

In total, the cache architecture uses 5 distinct FIFOs. Figure 1 shows the situation of these FIFOs within the cache architecture. The input and output FIFOs respectively buffer the input addresses and output read data from the cache. The miss and read FIFOs respectively buffer cache miss addresses to external memory and the returning read data. The bypass FIFO buffers control information between the front-end and back-end of the cache and its depth should be sufficient to hide the latency of a memory access. If the depth of this FIFO is less than the number of clock cycles taken for an external memory read, then the performance of the cache will be severely degraded.

The cache will function correctly with all FIFOs set to a depth of 2 or more entries with the exception of the output FIFO, which must be at least 4 entries deep for correct operation of the cache.

The output signal *fifo\_flags* may be used during the set-up of the cache in order to determine which FIFOs are full/empty during operation. The depths of the FIFOs may then be adjusted accordingly to achieve optimal performance.

**TAG compare block**

Input addresses to the cache are partitioned into tag, index and word offset. The word offset is the offset of a word within a cache line, the

<sup>2</sup> Assuming the FIFO buffering is set up correctly.

index is the address of the line and the tag is the unique address in memory of the cache line. As an example, consider a 128 kbyte cache with a 64-byte line size having access to a 8 Mbyte external memory. An example set of generic parameters would be:

Generic name	Value	Notes
addr_width	20	Total addressable external memory = $2^{20} \times 64\text{-bit} = 8\text{ Mbytes}$
word_width	64	64-bit word width
words_per_line	8	64-bit $\times$ 8 = 64-byte line size
log2_words_per_line	3	$2^3 = 8$
num_lines	2048	2048 lines $\times$ 64 bytes per line = 128 kbytes
log2_num_lines	11	$2^{11} = 2048$

With the above configuration, the tag, index and offset are respectively 6-bits, 11-bits and 3-bits wide. Figure 2 demonstrates how the tag RAM would be organized in this example.

On receipt of an input address, the index selects a tag in the tag RAM. This tag is then compared with the input tag address. If the tags match and the tag valid bit is set then there is a cache hit, otherwise there is a cache miss.

In the event of a miss, then the memory address that missed is sent via the miss FIFO to main memory. The tag at the current index is updated with the new tag address.

Whether the input address is a hit or miss, in both instances, a unique ID of the line that hit or the line that is to be replaced is sent via the bypass FIFO to the cache-controller.

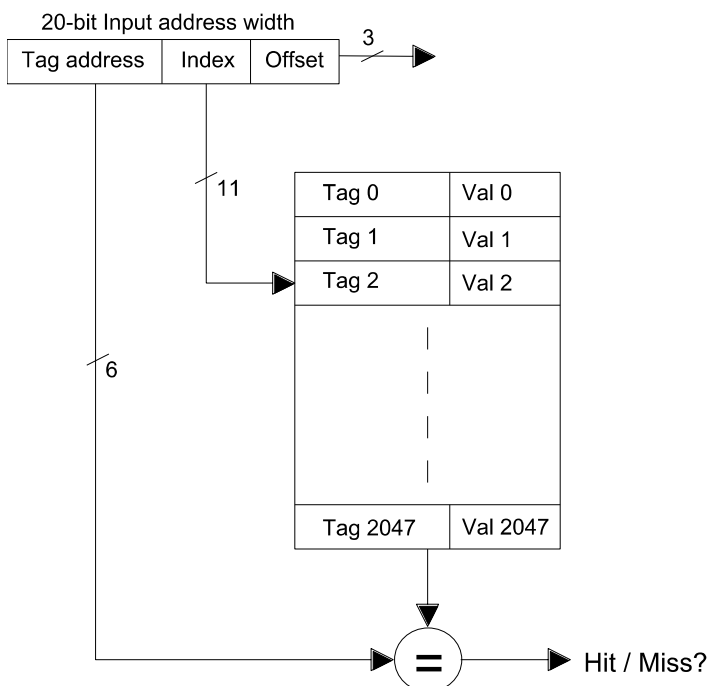


Figure 2: Organization of Tag RAM and compare logic

### Cache Controller

The cache controller services the hits and misses from the bypass FIFO. In the event of a cache hit, then the controller must read a cache line from the cache RAM and multiplex the correct word before presenting it to the output FIFO. In the event of a cache miss, the cache line read data is taken directly from the read FIFO and the cache RAM is updated accordingly.

### Cache Flush Control

In the event that the contents of the cache become incoherent with the contents of main memory (due to memory writes etc.) then the contents of the cache may be invalidated using the *flush* command. The *flush* signal must be asserted for at least one clock-cycle to initiate the internal flush state machine. Once initiated, the signal *cache\_addr\_rdy* is disabled and no further cache accesses are permitted until the flush operation is complete<sup>3</sup>. When the cache flush has finished, the state machine asserts *flush\_done* high for one clock cycle. Cache operation may then proceed as normal. Tags are invalidated one at a time on consecutive clock cycles, so the total delay of a cache flush operation is approximately equal to the generic parameter *num\_lines* plus the time taken for any existing requests to be flushed out of the pipeline.

Note that it is recommended that a cache flush operation is performed at least once after a system reset and before the cache is accessed for the first time. This ensures the tag valid bits are all reset.

### Functional Timing Diagrams

The following timing diagrams are valid for all cache configurations. Note that data is only transferred at the cache interfaces on a rising clock edge when valid and ready are both active high.

For detailed analysis of the valid/ready protocol see ZIPcores application note: [app\\_note\\_zc001.pdf](#).

Figure 3 shows a cache hit. The nominal latency for a cache hit is 8 clock cycles (assuming no misses are pending in the pipeline).

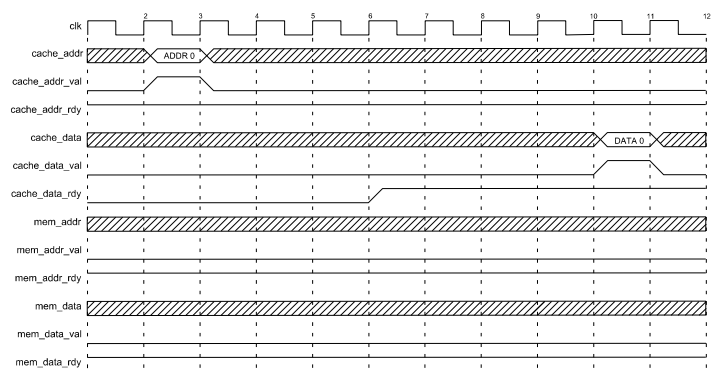


Figure 3: Cache hit

Figure 4 shows a cache miss (not to scale). Here we can see that the cache miss provokes an external memory read. The latency of a miss is 10 clock cycles plus the latency of the memory access.

<sup>3</sup> Any existing requests in the pipeline will be serviced before a cache flush commences.

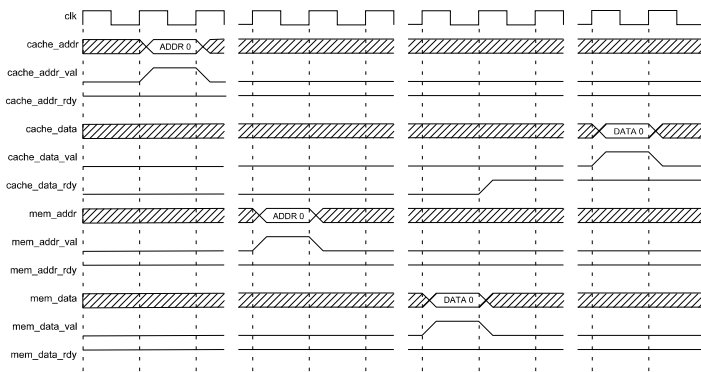


Figure 4: Cache miss

Finally, in Figure 5 we see a cache flush operation (again not to scale). Note that the signal `cache_addr_rdy` is held low until the flush operation is complete.

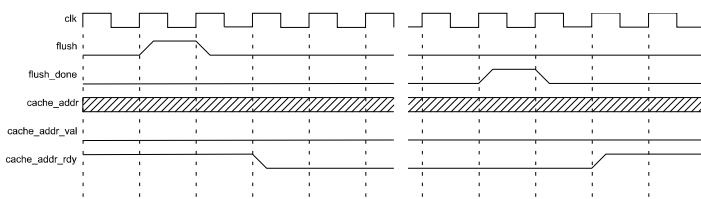


Figure 5: Cache flush

## Source File Description

All source files are provided as text files coded in VHDL. The following table gives a brief description of each file.

Source file	Description
<code>pipeline_reg.vhd</code>	Pipeline register
<code>fifo_sync.vhd</code>	Synchronous FIFO
<code>ram_dp_w_r_bp.vhd</code>	Dual-port RAM for tag storage
<code>ram_sp.vhd</code>	Single-port RAM for main cache storage
<code>rom_sp_16384x64bit.vhd</code>	ROM test bench model
<code>rom_sp_32768x16bit.vhd</code>	ROM test bench model
<code>rom_sp_65536x16bit.vhd</code>	ROM test bench model
<code>rom_sp_131072x8bit.vhd</code>	ROM test bench model
<code>mem_model_16384x64bit.vhd</code>	Main memory test bench model
<code>cache_direct_map.vhd</code>	Top-level block
<code>cache_direct_map_bench.vhd</code>	Top-level test bench

## Functional Testing

An example VHDL test bench is provided for use in a suitable VHDL simulator. The compilation order of the source code is as follows:

1. `pipeline_reg.vhd`
2. `fifo_sync.vhd`
3. `ram_dp_w_r_bp.vhd`
4. `ram_sp.vhd`
5. `rom_sp_16384x64bit.vhd`
6. `rom_sp_32768x32bit.vhd`
7. `rom_sp_65536x16bit.vhd`
8. `rom_sp_131072x8bit.vhd`
9. `mem_model_16384x64bit.vhd`
10. `cache_direct_map.vhd`
11. `cache_direct_map_bench.vhd`

The VHDL test bench instantiates the cache component, a memory model and a ROM model. In the example provided, the cache is configured with a 64-bit line size and a 16-bit word width. The memory model is organized as 16384x64-bit and the ROM is 65536x16-bit. Various ROM models are provided in case the user wishes to configure the cache with different word widths. The word width of the memory model is fixed.

The simulation must be run for at least 10 ms during which time a series of randomized cache accesses will be performed. In parallel, each access to the cache is mirrored by an identical access to the ROM. By capturing the output data from the cache and the output data from the ROM, the correct operation of the cache may be verified<sup>4</sup>.

In addition to randomized cache read accesses, the test bench also generates randomized valid/ready handshake signals at the cache output. The memory model also has a generic stalling function. A stall factor may be set ranging from 0 to 4 where 0 signifies no stalling and 4 signifies heavy stalling.

The simulation generates two text files: `cache_direct_map_out0.txt` and `cache_direct_map_out1.txt`. These files respectively contain the read data from the cache and ROM captured during the course of the test. If these files match then the test has been successful.

## Synthesis

The files required for synthesis and the design hierarchy is shown below:

- `cache_direct_map`
  - `pipeline_reg.vhd`
  - `ram_dp_w_r_bp.vhd`
  - `ram_sp.vhd`
  - `fifo_sync.vhd`
  - `pipeline_reg.vhd`

The VHDL core is designed to be technology independent. However, as a benchmark, synthesis results have been provided for the Xilinx Virtex 5 and the Altera Stratix III series of FPGA devices. The lowest and highest speed grade devices have been chosen in both cases for comparison.

It is important to note that the synthesis results will largely depend on the choice of generic parameters. As a general rule, the critical timing paths reside in the tag compare logic and the output word multiplexer. As the tag width or line width increase then the maximum attainable clock-speed generally decreases.

4 The memory model and ROM model contain the same data

Trial synthesis results are shown with the generic parameters set to: `addr_width = 16`, `word_width = 16`, `words_per_line = 4`, `num_lines = 4096`, `bp_fifo_depth = 32`. All the other FIFOs have the minimum allowed FIFO depth. These generic settings represent a 32 kbyte cache.

Resource usage is specified after Place and Route.

#### VIRTEX 5

<b>Resource type</b>	<b>Quantity used</b>
Slice register	290
Slice LUT	278
Block RAM	9
DSP48	0
Clock frequency (worst case)	198 MHz
Clock frequency (best case)	265 MHz

#### STRATIX III

<b>Resource type</b>	<b>Quantity used</b>
Register	1106
ALUT	477
Block Memory bit	274,432
DSP block 18	0
Clock frequency (worse case)	190 MHz
Clock frequency (best case)	250 MHz

#### Revision History

<b>Revision</b>	<b>Change description</b>	<b>Date</b>
1.0	Initial revision	25/04/2008