IOb-Cache

A Configurable Cache

October 30, 2022

User Guide, V0.10 , Build 5077cb0

Document Version History

Version	Date	Person	Changes from previous version
V0.10	May/30/2022	JTS	Document released.

Contents

List of Tables

List of Figures

1 Introduction

IOb-Cache is an open-source configurable pipelined memory cache. The processor-side interface (front-end) uses IObundle's Native Pipelined Interface (NPI). The memory-side interface (back-end) can also be configured to use NPI or the widely used AXI4 interface. The address and data widths of the front-end and back-end are configurable to support multiple user cores and memories. IOb-Cache is a K-Way Set-Associative cache, where K can vary from 1 (directly mapped) to 8 or more ways, provided the operating frequency after synthesis is acceptable. IOb-Cache supports the two most common write policies: Write-Through Not-Allocate and Write-Back Allocate.

IOb-Cache was developed in the scope of João Roque's master's thesis in Electrical and Computer Engineering at the Instituto Superior Técnico of the University of Lisbon. The Verilog code works well in IObundle's IOb-SoC system (https://github.com/IObundle/iob-soc) both in simulation and FPGA. To be used in an ASIC, it would need to be lint-cleaned and verified more thoroughly by RTL simulation to achieve 100% code coverage desirably.

Figure 1: IP Core Symbol

1.1 Features

- Pipelined operation allowing consecutive one-cycle reads and writes
- IObundle's Native Pipelined Interface (NPI) front-end native interface on the processor side (front-end)
- NPI or AXI4 interface on the memory side (back-end)
- Configurable address and data widths on the front-end and back-end interfaces for supporting a variety of different systems
- Configurable number of lines and words per line
- Configurable K-Way Set-Associativity ($k \geq 1$)
- Configurable line replacement policy: LRU, MRU-based PLRU, and tree-based PLRU.
- Configurable Write-Through Not-Allocate and Write-Back Allocate policies
- Configurable Write-Through buffer depth
- Optional control address space for cache invalidation, accessing the write through buffer status and read/write hit/miss counters

1.2 Benefits

- Easy to integrate hardware and bare-metal software
- Low usage of FPGA resources or silicon area
- Operating frequency suitable for low-cost FPGAs and less recent ASIC nodes
- Low power consumption

1.3 Deliverables

- Verilog source code and testbench
- Verilator testbench
- FPGA implementation scripts
- Bare-metal software driver
- Documentation Latex sources
- Example System on Chip using IOb-Cache

2 Description

2.1 Block Diagram

Figure 2 presents a high-level block diagram of the core, followed by a brief description of each block.

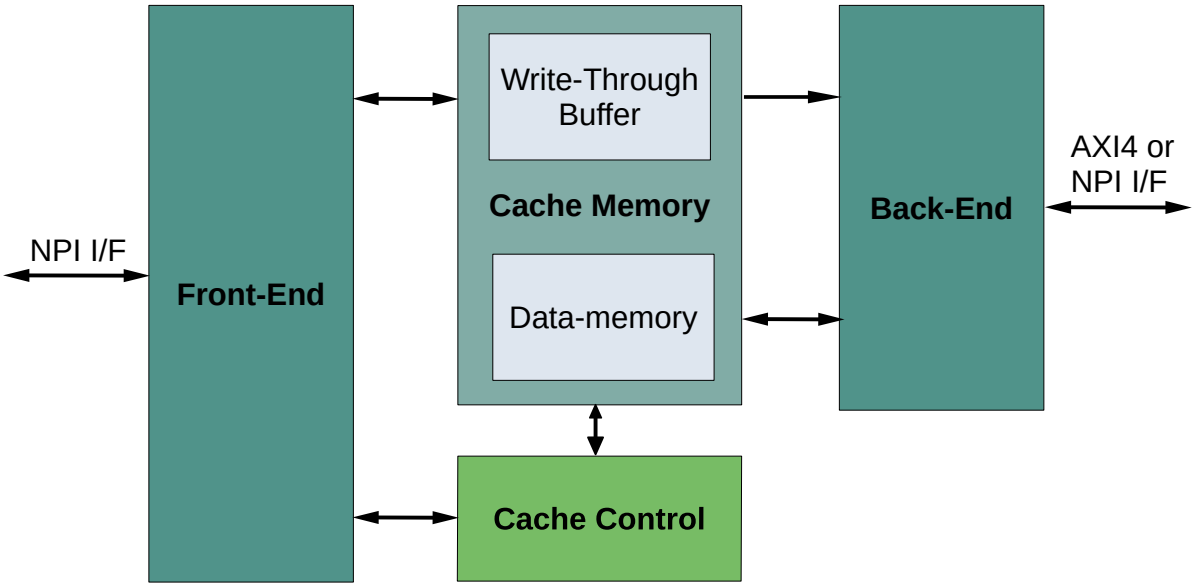

Figure 2: High-Level Block Diagram

FRONT-END This NIP interface is connected to a processor or any other processing element that needs a cache buffer to improve the performance of accessing a slower but larger memory.

CACHE MEMORY This block contains the tag, data storage memories and the Write Through Buffer if the correspeonding write policy is selected; these memories are implemented either with RAM if large enough, or with registers if small enough.

BACK-END INTERFACE Memory-side interface: if the cache is at the last level before the target memory module, the back-end interface connects to the target memory (e.g. DDR) controller; if the cache is not at the last level, the back-end interface connects to the next-level cache. This interface can be of type NPI or AXI4 as per configuration. If it is connected to the next-level IOb-Cache, the NPI type must be selected; if it is connected to a third party cache or memory controlller featuring an AXI4 interface, then the AXI4 type must be selected.

CACHE CONTROL Cache controller: this block is used for invalidating the cache, monitoring the status of the Write Thorough buffer, and accessing read/write hit/miss counters.

2.2 Configuration

2.2.1 Parameters

The configuration parameters of the core are presented in Table 1. Configuration parameters can vary from instance to instance.

Table 1: Configuration Parameters

Parameter	Min	Typ	Max	Description
ADDR_W		None	64	Front-end address width (log2): defines the total memory space accessible via the cache, which must be a power of two.
DATA_W	32	None	64	Front-end data width (log2): this parameter allows supporting processing elements with various data widths.
BE_ADDR_W		None		Back-end address width (log2): the value of this parameter must be equal or greater than ADDR_W to match the width of the back-end interface, but the address space is still dictated by ADDR_W.
BE_DATA_W	32	None	256	Back-end data width (log2): the value of this parameter must be an integer multiple $k \geq 1$ of DATA_W. If $k > 1$, the memory controller can operate at a frequency higher than the cache's frequency. Typically, the memory controller has an asynchronous FIFO interface, so that it can sequentially process multiple commands received in paralell from the cache's back-end interface.
NWAYS_W	0	None	8	Number of cache ways (log2): the miminum is 0 for a directly mapped cache; the default is 1 for a two-way cache; the maximum is limited by the desired maximum operating frequency, which degrades with the number of ways.
NLINES_W		None		Line offset width (log2): the value of this parameter equals the number of cache lines, given by 2**NLINES_W.
WORD_OFFSET_W	0	None		Word offset width (log2): the value of this parameter equals the number of words per line, which is 2**OFFSET_W.

WTBUF_DEPTH_W		None		Write-through buffer depth (log2). A shallow buffer will fill up more frequently and cause write stalls; however, on a Read After Write (RAW) event, a shallow buffer will empty faster, decreasing the duration of the read stall. A deep buffer is unlkely to get full and cause write stalls; on the other hand, on a RAW event, it will take a long time to empty and cause long read stalls.
REP_POLICY	0	None	3	Line replacement policy: set to 0 for Least Recently Used (LRU); set to 1 for Pseudo LRU based on Most Recently Used (PLRU_MRU); set to 2 for tree-based Pseudo LRU (PLRU_TREE).
WRITE_POL	0	None	1	Write policy: set to 0 for write-through or set to 1 for write-back.
USE_CTRL	0	None	1	Instantiates a cache controller (1) or not (0). The cache controller provides memory-mapped software accessible registers to invalidate the cache data contents, and monitor the write through buffer status using the front-end interface. To access the cache controller, the MSB of the address mut be set to 1. For more information refer to the example software functions provided.
USE_CTRL_CNT	0	None	1	Instantiates hit/miss counters for reads, writes or both (1), or not (0). This parameter is meaningful if the cache controller is present (USE_CTRL=1), providing additional software accessible functions for these functions.

2.3 Interface Signals

Table 2: General Interface Signals

Name	Direction	Width	Description
clk_i	INPUT	1	System clock input.
rst_i	INPUT	1	System reset, asynchronous and active high.

Table 3: Front-End Interface Signals

Name	Direction	Width	Description
req	INPUT	1	Read or write request from host. If signal `ack` raises in the next cyle the request has been served; otherwise `req` should remain high until `ack` raises. When `ack` raises in response to a previous request, `req` may keep high, or combinatorially lowered in the same cycle. If `req` keeps high, a new request is being made to the current address `addr`; if `req` lowers, no new request is being made. Note that the new request is being made in parallel with acknowledging the previous request: pipelined operation.

addr	INPUT	USE_CTRL+ADDR_W-IOB_CACHE_NBYTES_W	Address from CPU or other user core, excluding the byte selection LSBs.
wdata	INPUT	DATA_W	Write data fom host.
wstrb	INPUT	IOB_CACHE_NBYTES	Byte write strobe from host.
rdata	OUTPUT	DATA_W	Read data to host.
ack	OUTPUT	1	Acknowledge signal from cache: indicates that the last request has been served. The next request can be issued as soon as this signal raises, in the same clock cycle, or later after it becomes low.

Table 4: Invalidate and Write-Through Buffer Empty Chain Interface Signals

Name	Direction	Width	Description
invalidate_in	INPUT	1	Invalidates all cache lines instantaneously if high.
invalidate_out	OUTPUT	1	This output is asserted high when the cache is invalidated via the cache controller or the direct `invalidate_in` signal. The present `invalidate_out` signal is useful for invalidating the next-level cache if there is one. If not, this output should be floated.
wtb_empty_in	INPUT	1	This input is driven by the next-level cache, if there is one, when its write-through buffer is empty. It should be tied high if there is no next-level cache. This signal is used to compute the overall empty status of a cache hierarchy, as explained for signal `wtb_empty_out`.
wtb_empty_out	OUTPUT	1	This output is high if the cache's write-through buffer is empty and its `wtb_empty_in` signal is high. This signal informs that all data written to the cache has been written to the destination memory module, and all caches on the way are empty.

Table 5: Native Back-End Interface Signals

Name	Direction	Width	Description
be_req	OUTPUT	1	Read or write request to next-level cache or memory.
be_addr	OUTPUT	BE_ADDR_W	Address to next-level cache or memory.
be_wdata	OUTPUT	BE_DATA_W	Write data to next-level cache or memory.
be_wstrb	OUTPUT	IOB_CACHE_BE_NBYTES	Write strobe to next-level cache or memory.
be_rdata	INPUT	BE_DATA_W	Read data from next-level cache or memory.
be_ack	INPUT	1	Acknowledge signal from next-level cache or memory.

Table 6: Back-End Interface Signals

Name	Direction	Width	Description
axi_awid_o	OUTPUT	AXI_ID_W	Address write channel ID.
axi_awaddr_o	OUTPUT	AXI_ADDR_W	Address write channel address.
axi_awlen_o	OUTPUT	AXI_LEN_W	Address write channel burst length.
axi_awsize_o	OUTPUT	3	Address write channel burst size. This signal indicates the size of each transfer in the burst.
axi_awburst_o	OUTPUT	2	Address write channel burst type.
axi_awlock_o	OUTPUT	2	Address write channel lock type.

axi_awcache_o	OUTPUT	4	Address write channel memory type. Transactions set with Normal Non-cacheable Modifiable and Bufferable (0011).
axi_awprot_o	OUTPUT	3	Address write channel protection type. Transactions set with Normal, Secure, and Data attributes (000).
axi_awqos_o	OUTPUT	4	Address write channel quality of service.
axi_awvalid_o	OUTPUT	1	Address write channel valid.
axi_awready_i	INPUT	1	Address write channel ready.
axi_wdata_o	OUTPUT	AXI_DATA_W	Write channel data.
axi_wstrb_o	OUTPUT	AXI_DATA_W/8	Write channel write strobe.
axi_wlast_o	OUTPUT	1	Write channel last word flag.
axi_wvalid_o	OUTPUT	1	Write channel valid.
axi_wready_i	INPUT	1	Write channel ready.
axi_bid_i	INPUT	AXI_ID_W	Write response channel ID.
axi_bresp_i	INPUT	2	Write response channel response.
axi_bvalid_i	INPUT	1	Write response channel valid.
axi_bready_o	OUTPUT	1	Write response channel ready.
axi_arid_o	OUTPUT	AXI_ID_W	Address read channel ID.
axi_araddr_o	OUTPUT	AXI_ADDR_W	Address read channel address.
axi_arlen_o	OUTPUT	AXI_LEN_W	Address read channel burst length.
axi_arsize_o	OUTPUT	3	Address read channel burst size. This signal indicates the size of each transfer in the burst.
axi_arburst_o	OUTPUT	2	Address read channel burst type.
axi_arlock_o	OUTPUT	2	Address read channel lock type.
axi_arcache_o	OUTPUT	4	Address read channel memory type. Transactions set with Normal Non-cacheable Modifiable and Bufferable (0011).
axi_arprot_o	OUTPUT	3	Address read channel protection type. Transactions set with Normal, Secure, and Data attributes (000).
axi_arqos_o	OUTPUT	4	Address read channel quality of service.
axi_arvalid_o	OUTPUT	1	Address read channel valid.
axi_arready_i	INPUT	1	Address read channel ready.
axi_rid_i	INPUT	AXI_ID_W	Read channel ID.
axi_rdata_i	INPUT	AXI_DATA_W	Read channel data.
axi_rresp_i	INPUT	2	Read channel response.
axi_rlast_i	INPUT	1	Read channel last word.
axi_rvalid_i	INPUT	1	Read channel valid.
axi_rready_o	OUTPUT	1	Read channel ready.

2.4 Software Accessible Registers

The software accessible registers of the core are described in the following tables. The tables give information on the name, read/write capability, word aligned addresses, used word bits, and a textual description.

Table 7: Software Accessible Registers

Name	R/W	Addr	Bits	Initial Value	Description
WTB_EMPTY	R	0	8	0	Write-through buffer empty (1) or non-empty (0).
WTB_FULL	R	1	8	0	Write-through buffer full (1) or non-full (0).

RW_HIT	R	4	32	0	Read and write hit counter.
RW_MISS	R	8	32	0	Read and write miss counter.
READ_HIT	R	12	32	0	Read hit counter.
READ_MISS	R	16	32	0	Read miss counter.
WRITE_HIT	R	20	32	0	Write hit counter.
WRITE_MISS	R	24	32	0	Write miss counter.
RST_CNTRS	W	28	8	0	Reset read/write hit/miss counters by writing any value to this register.
INVALIDATE	W	32	8	0	Invalidate the cache data contents by writing any value to this register.
VERSION	R	36	32	0	Cache version.

3 Usage

Figure 3 illustrates how to instantiate the IP core and, if applicable, the required external blocks.

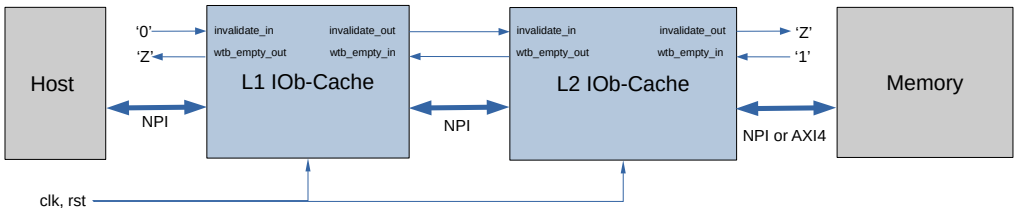

Figure 3: Core Instance and Required Surrounding Blocks

The figure shows a three-level memory hierarchy comprising L1 and L2 caches and a memory module. The Host drives the L1 cache through its front-end NPI interface (the user is free to develop other interfaces). The L1 and L2 caches are connected using another NPI interface since IOb-Cache's front-end interface only supports this.

The `wtb_empty_in` and `wtb_empty_out` signals form a chain from the L1's front-end to the L2's back-end. As explained in the description of these signals, this chain ensures that the user's core knows that all write-through buffers across the cache hierarchy are empty. Note that the L1's `wtb_empty_out` signal is floating because the Host uses the cache controller to query the write-through buffer status. The L2's `wtb_empty_in` is tied high as L2 is the last cache in the hierarchy, and there are no more write-through buffers to its right-hand side.

The `invalidate_in` and `invalidate_out` signals form another chain that ensures that the data in the whole cache hierarchy is invalidated, as explained in these signal's descriptions. Note that the L1's `invalidate_in` signal is tight to low as L1 is invalidated via the cache controller by writing to the respective address. The L2's `invalidate_out` signal is floating because L2 is the last cache in the hierarchy, and there are no more caches to invalidate.

Finally, L2 is connected to a memory module, and one can choose between NPI or AXI4 interfaces. In practice, most memory modules have a standard interface such as AXI4, which is themost common choice, although one may choose NPI in less usual simulation or FPGA prototyping scenarios.

3.1 Simulation

The provided testbench uses the core instance described in Section 3. A high-level block diagram of the testbench is shown in Figure 4. The testbench is organized in a modular fashion, with each test described in a separate file. The test suite consists of all the test case files to make adding, modifying, or removing tests easy.

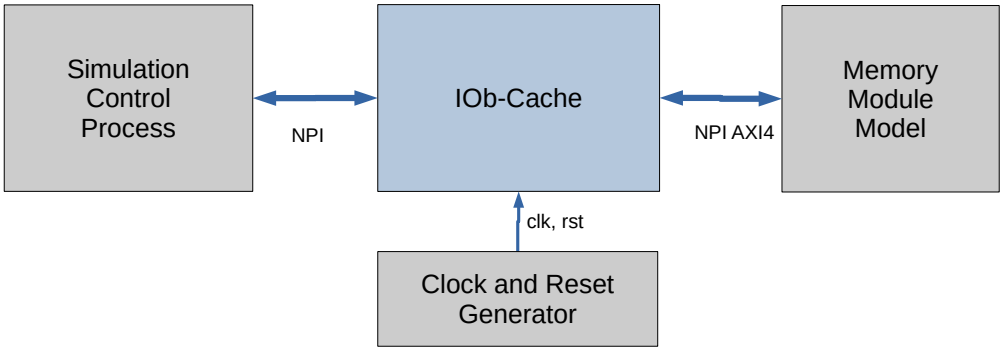

Figure 4: Testbench Block Diagram

The above paragraph describes a desirable simulation setup, but IOb-Cache's simulation environment still lacks a modular simulation structure. Currently, only a set of primary non-pipelined write followed by read tests is implemented. However, IOb-Cache has been thoroughly verified in-system, with two cache levels. Various open-source RISC-V processors have proven that IOb-Cache works well: PicoRV32, SSRV, VexRISCV, and DarkRV.