IOb-Cache

A Configurable Cache

January 15, 2026

User Guide, V0.71, Build 5c596de

Document Version History

Version	Date	Person	Changes from previous version
0.71	May 30, 2022	JTS	Document released.
0.71	January 15, 2026	AN	Add support for new FE interfaces; Update Figures.

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) can be configured to use IObundle's Native Interface (IOb), AXI4-Lite or Wishbone interface. The memory-side interface (back-end) can also be configured to use IOb 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.

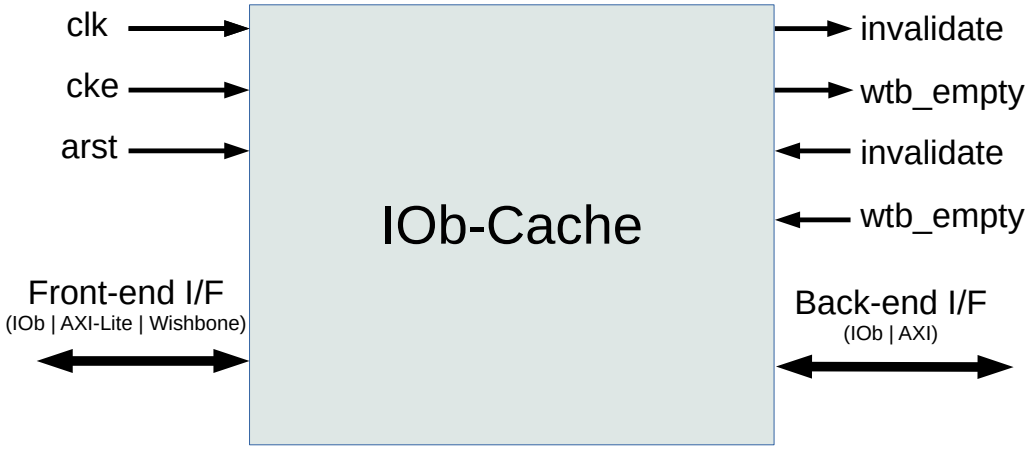

Figure 1: IP Core Symbol

1.1 Features

- Pipelined operation allowing consecutive one-cycle reads and writes
- Support for multiple interface types (selectable) on the processor side (front-end): IOb, AXI-Lite, Wishbone
- IOb 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 Deliverables

- Verilog RTL source code synthesizable for ASIC and FPGA
- Verilog testbench and simulation scripts for code coverage
- ASIC synthesis script and timing constraints
- FPGA synthesis scripts and timing constraints
- Bare-metal software driver
- Comprehensive user guide
- Example System on Chip using IOb-Cache

2 Description

This section gives a detailed description of the IP core. The high-level block diagram is presented, along with a description of its subblocks. The parameters and macros that define the core configuration are listed and explained. The interface signals are enumerated and described; if timing diagrams are needed, they are shown after the interface signals. Finally the Control and Status Registers (CSR) are outlined and explained.

2.1 Block Diagram

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

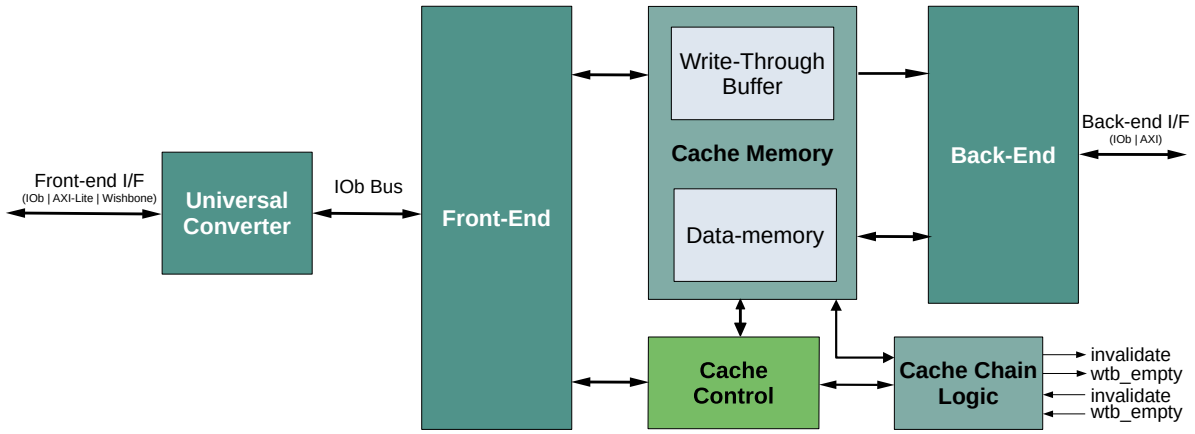

Figure 2: High-Level Block Diagram

The Verilog modules in the top-level entity of the core are described in the following tables. The table elements represent the subblocks in the Block Diagram.

Name	Description
iob_universal_converter	Convert front-end interface into internal IOb port
front_end	This IOb 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_axi	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 module implements an AXI4 interface

Table 1: Core subblocks.

2.2 Configuration

The following tables describe the IP core configuration. The core may be configured using macros or parameters:

'M' Macro: a Verilog macro or `define` directive is used to include or exclude code segments, to create core configurations that are valid for all instances of the core.

'P' Parameter: a Verilog parameter is passed to each instance of the core and defines the configuration of that particular instance.

Configuration	Type	Min	Typical	Max	Description
LRU	M	?	0	?	Least Recently Used – more resources intensive - N*log2(N) bits per cache line - Uses counters
PLRU_MRU	M	?	1	?	bit-based Pseudo-Least-Recently-Used, a simpler replacement policy than LRU, using a much lower complexity (lower resources) - N bits per cache line
PLRU_TREE	M	?	2	?	tree-based Pseudo-Least-Recently-Used, uses a tree that updates after any way received an hit, and points towards the oposing one. Uses less resources than bit-pseudo-lru - N-1 bits per cache line
WRITE_THROUGH	M	?	0	?	write-through not allocate: implements a write-through buffer
WRITE_BACK	M	?	1	?	write-back allocate: implementes a dirty-memory
ADDR_W_CSRS	M	?	5	?	Address width of CSRs
FE_ADDR_W	P	1	24	64	Front-end address width (log2): defines the total memory space accessible via the cache, which must be a power of two.
FE_DATA_W	P	32	32	64	Front-end data width (log2): this parameter allows supporting processing elements with various data widths.
BE_ADDR_W	P	1	24		Back-end address width (log2): the value of this parameter must be equal or greater than FE_ADDR_W to match the width of the back-end interface, but the address space is still dictated by ADDR_W.
BE_DATA_W	P	32	32	256	Back-end data width (log2): the value of this parameter must be an integer multiple $k>=1$ 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	P	0	1	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	P		7		Line offset width (log2): the value of this parameter equals the number of cache lines, given by 2**NLINES_W.
WORD_OFFSET_W	P	1	3		Word offset width (log2): the value of this parameter equals the number of words per line, which is 2**OFFSET_W.
WTBUF_DEPTH_W	P		4		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	P	0	0	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	P	0	0	1	Write policy: set to 0 for write-through or set to 1 for write-back.
USE_CTRL	P	0	0	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	P	0	0	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.
AXI	M	NA	NA	NA	AXI interface used by backend
AXI_ID_W	P	0	1	32	AXI ID width
AXI_ID	P	0	0	32	AXI ID
AXI_LEN_W	P	0	4	32	AXI length
AXI_ADDR_W	P	0	BE_ADDR_W	32	AXI address width
AXI_DATA_W	P	0	BE_DATA_W	32	AXI data width

VERSION	M	NA	16'h0071	NA	Product version. This 16-bit macro uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part.

Table 2: General operation group

2.3 Interface Signals

The interface signals of the core are described in the following tables. Note that the ouput signals are registered in the core, while the input signals are not.

Name	Direction	Width	Description
clk_i	input	1	Clock
cke_i	input	1	Clock enable
arst_i	input	1	Asynchronous active-high reset

Table 3: Clock, clock enable and reset

Name	Direction	Width	Description
iob_valid_i	input	1	Request address is valid.
iob_addr_i	input	ADDR_W	Byte address.
iob_wdata_i	input	DATA_W	Write data.
iob_wstrb_i	input	DATA_W/8	Write strobe.
iob_rvalid_o	output	1	Read data valid.
iob_rdata_o	output	DATA_W	Read data.
iob_ready_o	output	1	Interface ready.

Table 4: Front-end interface, when selecting the IOb FE interface.

Name	Direction	Width	Description
wb_dat_o	output	DATA_W	Data input.
wb_datout_i	input	DATA_W	Data output.
wb_ack_o	output	1	Acknowledge input. Indicates normal termination of a bus cycle.
wb_adr_i	input	ADDR_W	Address output. Passes binary address.
wb_cyc_i	input	1	Cycle output. Indicates a valid bus cycle.
wb_sel_i	input	DATA_W/8	Select output. Indicates where valid data is expected on the data bus.
wb_stb_i	input	1	Strobe output. Indicates valid access.
wb_we_i	input	1	Write enable. Indicates write access.

Table 5: Front-end interface, when selecting the Wishbone FE interface.

Name	Direction	Width	Description

axil_araddr_i	input	ADDR_W	AXI-Lite address read channel byte address.
axil_arvalid_i	input	1	AXI-Lite address read channel valid.
axil_arready_o	output	1	AXI-Lite address read channel ready.
axil_rdata_o	output	DATA_W	AXI-Lite read channel data.
axil_rresp_o	output	2	AXI-Lite read channel response.
axil_rvalid_o	output	1	AXI-Lite read channel valid.
axil_rready_i	input	1	AXI-Lite read channel ready.
axil_awaddr_i	input	ADDR_W	AXI-Lite address write channel byte address.
axil_awvalid_i	input	1	AXI-Lite address write channel valid.
axil_awready_o	output	1	AXI-Lite address write channel ready.
axil_wdata_i	input	DATA_W	AXI-Lite write channel data.
axil_wstrb_i	input	DATA_W/8	AXI-Lite write channel write strobe.
axil_wvalid_i	input	1	AXI-Lite write channel valid.
axil_wready_o	output	1	AXI-Lite write channel ready.
axil_bresp_o	output	2	AXI-Lite write response channel response.
axil_bvalid_o	output	1	AXI-Lite write response channel valid.
axil_bready_i	input	1	AXI-Lite write response channel ready.

Table 6: Front-end interface, when selecting the AXI-Lite FE interface.

Name	Direction	Width	Description
invalidate_i	input	1	Invalidates all cache lines instantaneously if high.
invalidate_o	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_i	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_o	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 7: Cache invalidate and write-trough buffer IO chain

Name	Direction	Width	Description
axi_araddr_o	output	AXI_ADDR_W	AXI address read channel byte address.
axi_arvalid_o	output	1	AXI address read channel valid.
axi_arready_i	input	1	AXI address read channel ready.
axi_rdata_i	input	AXI_DATA_W	AXI read channel data.
axi_rresp_i	input	2	AXI read channel response.
axi_rvalid_i	input	1	AXI read channel valid.
axi_rready_o	output	1	AXI read channel ready.
axi_arid_o	output	AXI_ID_W	AXI address read channel ID.
axi_arlen_o	output	AXI_LEN_W	AXI address read channel burst length.
axi_arsize_o	output	3	AXI address read channel burst size.
axi_arburst_o	output	2	AXI address read channel burst type.
axi_arlock_o	output	1	AXI address read channel lock type.

axi_arcache_o	output	4	AXI address read channel memory type.
axi_arqos_o	output	4	AXI address read channel quality of service.
axi_rid_i	input	AXI_ID_W	AXI Read channel ID.
axi_rlast_i	input	1	AXI Read channel last word.
axi_awaddr_o	output	AXI_ADDR_W	AXI address write channel byte address.
axi_awvalid_o	output	1	AXI address write channel valid.
axi_awready_i	input	1	AXI address write channel ready.
axi_wdata_o	output	AXI_DATA_W	AXI write channel data.
axi_wstrb_o	output	AXI_DATA_W/8	AXI write channel write strobe.
axi_wvalid_o	output	1	AXI write channel valid.
axi_wready_i	input	1	AXI write channel ready.
axi_bresp_i	input	2	AXI write response channel response.
axi_bvalid_i	input	1	AXI write response channel valid.
axi_bready_o	output	1	AXI write response channel ready.
axi_awid_o	output	AXI_ID_W	AXI address write channel ID.
axi_awlen_o	output	AXI_LEN_W	AXI address write channel burst length.
axi_awsize_o	output	3	AXI address write channel burst size.
axi_awburst_o	output	2	AXI address write channel burst type.
axi_awlock_o	output	1	AXI address write channel lock type.
axi_awcache_o	output	4	AXI address write channel memory type.
axi_awqos_o	output	4	AXI address write channel quality of service.
axi_wlast_o	output	1	AXI Write channel last word flag.
axi_bid_i	input	AXI_ID_W	AXI Write response channel ID.

Table 8: Back-end interface, when selecting the AXI4 BE interface.

Name	Direction	Width	Description
be_iob_valid_o	output	1	Request address is valid.
be_iob_addr_o	output	BE_ADDR_W	Byte address.
be_iob_wdata_o	output	BE_DATA_W	Write data.
be_iob_wstrb_o	output	BE_DATA_W/8	Write strobe.
be_iob_rvalid_i	input	1	Read data valid.
be_iob_rdata_i	input	BE_DATA_W	Read data.
be_iob_ready_i	input	1	Interface ready.

Table 9: Back-end interface, when selecting the IOb BE interface.

2.4 Control and Status Registers

The software accessible registers of the core are described in the following tables. The tables give information on the name, read/write capability, address, hardware and software width, and a textual description. The addresses are byte aligned and given in hexadecimal format. The hardware width is the number of bits that the register occupies in the hardware, while the software width is the number of bits that the register occupies in the software. In each address, the right-justified field having "Hw width" bits conveys the relevant information. Each register has only one type of access, either read or write, meaning that reading from a write-only register will produce invalid data or writing to a read-only register will not have any effect.

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		

WTB_EMPTY	R	0x0	1	8	0	Write-through buffer empty (1) or non-empty (0).
WTB_FULL	R	0x1	1	8	0	Write-through buffer full (1) or non-full (0).
RW_HIT	R	0x4	32	32	0	Read and write hit counter.
RW_MISS	R	0x8	32	32	0	Read and write miss counter.
READ_HIT	R	0xC	32	32	0	Read hit counter.
READ_MISS	R	0x10	32	32	0	Read miss counter.
WRITE_HIT	R	0x14	32	32	0	Write hit counter.
WRITE_MISS	R	0x18	32	32	0	Write miss counter.
RST_CNTRS	W	0x1C	1	8	0	Reset read/write hit/miss counters by writing any value to this register.
INVALIDATE	W	0x1D	1	8	0	Invalidate the cache data contents by writing any value to this register.

Table 10: CACHE software accessible registers.

Name	R/W	Addr	Width		Default	Description
			Hw	Sw		
VERSION	R	0x1E	16	16	0071	Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.

Table 11: General Registers.

3 Usage

3.1 Instantiation

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

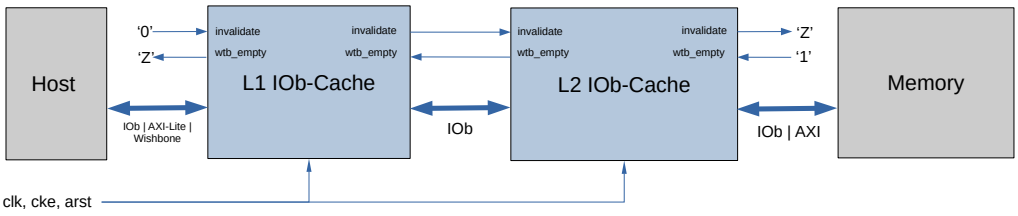

Figure 3: Core Instance and Required Surrounding Blocks

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 interface (IOb by default. The user is free to select other supported interfaces or develop new ones). The L1 and L2 caches are connected using another IOb interface.

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 IOb or AXI4 interfaces. In practice, most memory modules have a standard interface such as AXI4, which is the most common choice, although one may choose IOb in less usual simulation or FPGA prototyping scenarios.

3.2 Simulation

The provided testbench uses the core instance described in Section 3.1. 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.

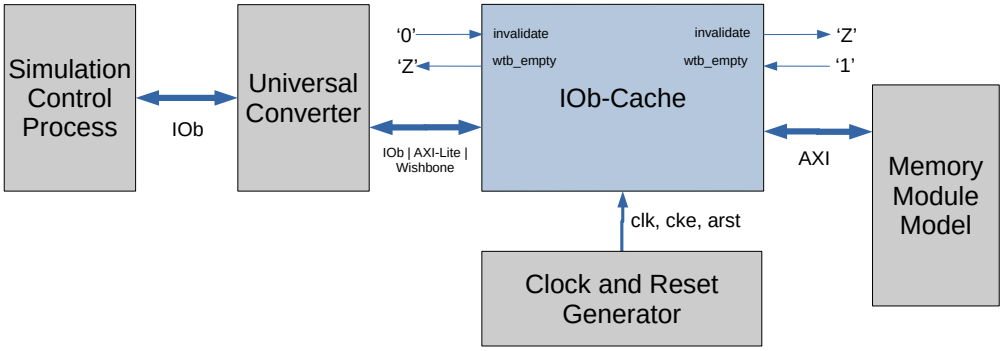

Figure 4: Testbench Block Diagram

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.

The testbench architecture involves the following components and data flow:

- Includes an iob_universal_converter module to test the various types of interfaces supported by the IOb-Cache front-end interface.
- The IOb-Cache core's back-end interface is connected to a model memory via AXI bus.

The testbench controller orchestrates the test sequence as follows:

1. Initializes cache.
2. Simple test: Writes a few random words to cache and reads them back for verification.
3. Data test: Writes custom data words to cache and reads them back for verification.
4. Address test: Writes words to specific addresses, including the highest one, and reads them back for verification.
5. LRU test: Writes words to specific addresses at regular interval and reads them back to verify cache replacement policy.
6. Controller test: Reads and modifies Cache controller CSRs to verify their functionality.

System-level Simulation

Upon request, simulation files to run the core embedded in a RISC-V system can be provided. The core is exercised in various modes by the RISC-V processor, using a bare-metal software program written in the C programming language.

4 Baremetal Drivers

iob_cache_axi_csrs.h File Reference

Function prototypes for the iob_cache_axi core.

```
   #include <stdint.h>
#include "iob_cache_axi_csrs_conf.h"
```

Functions

- void iob_cache_axi_csrs_init_baseaddr (uint32_t addr)
  Set core base address.
- void iob_write (uint32_t addr, uint32_t data_w, uint32_t value)
  Write access function prototype.
- uint32_t iob_read (uint32_t addr, uint32_t data_w)
  Read access function prototype.
- uint8_t iob_cache_axi_csrs_get_WTB_EMPTY ()
  Get WTB_EMPTY value. Write-through buffer empty (1) or non-empty (0).
- uint8_t iob_cache_axi_csrs_get_WTB_FULL ()
  Get WTB_FULL value. Write-through buffer full (1) or non-full (0).
- uint32_t iob_cache_axi_csrs_get_RW_HIT ()
  Get RW_HIT value. Read and write hit counter.
- uint32_t iob_cache_axi_csrs_get_RW_MISS ()
  Get RW_MISS value. Read and write miss counter.
- uint32_t iob_cache_axi_csrs_get_READ_HIT ()
  Get READ_HIT value. Read hit counter.
- uint32_t iob_cache_axi_csrs_get_READ_MISS ()
  Get READ_MISS value. Read miss counter.
- uint32_t iob_cache_axi_csrs_get_WRITE_HIT ()
  Get WRITE_HIT value. Write hit counter.
- uint32_t iob_cache_axi_csrs_get_WRITE_MISS ()
  Get WRITE_MISS value. Write miss counter.
- void iob_cache_axi_csrs_set_RST_CNTRS (uint8_t value)
  Set RST_CNTRS value. Reset read/write hit/miss counters by writing any value to this register.
- void iob_cache_axi_csrs_set_INVALIDATE (uint8_t value)
  Set INVALIDATE value. Invalidate the cache data contents by writing any value to this register.

- uint16_t iob_cache_axi_csrs_get_version ()

  Get version value. Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.

4.0.1 Detailed Description

Function prototypes for the iob_cache_axi core.

This file contains the function prototypes to access the Control and Status Registers (CSRs) for the iob_↩ cache_axi core.

This file is automatically generated by Py2HWSW

4.0.2 Function Documentation

iob_cache_axi_csrs_get_READ_HIT()

```
uint32_t iob_cache_axi_csrs_get_READ_HIT ()
```

Get READ_HIT value. Read hit counter.

Returns

uint32_t READ_HIT value.

iob_cache_axi_csrs_get_READ_MISS()

```
uint32_t iob_cache_axi_csrs_get_READ_MISS ()
```

Get READ_MISS value. Read miss counter.

Returns

uint32_t READ_MISS value.

iob_cache_axi_csrs_get_RW_HIT()

```
uint32_t iob_cache_axi_csrs_get_RW_HIT ()
```

Get RW_HIT value. Read and write hit counter.

Returns

uint32_t RW_HIT value.

iob_cache_axi_csrs_get_RW_MISS()

```
uint32_t iob_cache_axi_csrs_get_RW_MISS ()
```

Get RW_MISS value. Read and write miss counter.

Returns

uint32_t RW_MISS value.

iob_cache_axi_csrs_get_version()

```
uint16_t iob_cache_axi_csrs_get_version ()
```

Get version value. Product version. This 16-bit register uses nibbles to represent decimal numbers using their binary values. The two most significant nibbles represent the integral part of the version, and the two least significant nibbles represent the decimal part. For example V12.34 is represented by 0x1234.

Returns

uint16_t version value.

iob_cache_axi_csrs_get_WRITE_HIT()

```
uint32_t iob_cache_axi_csrs_get_WRITE_HIT ()
```

Get WRITE_HIT value. Write hit counter.

Returns

uint32_t WRITE_HIT value.

iob_cache_axi_csrs_get_WRITE_MISS()

```
uint32_t iob_cache_axi_csrs_get_WRITE_MISS ()
```

Get WRITE_MISS value. Write miss counter.

Returns

uint32_t WRITE_MISS value.

iob_cache_axi_csrs_get_WTB_EMPTY()

```
uint8_t iob_cache_axi_csrs_get_WTB_EMPTY ()
```

Get WTB_EMPTY value. Write-through buffer empty (1) or non-empty (0).

Returns

uint8_t WTB_EMPTY value.

iob_cache_axi_csrs_get_WTB_FULL()

```
uint8_t iob_cache_axi_csrs_get_WTB_FULL ()
```

Get WTB_FULL value. Write-through buffer full (1) or non-full (0).

Returns

uint8_t WTB_FULL value.

iob_cache_axi_csrs_init_baseaddr()

```
void iob_cache_axi_csrs_init_baseaddr (
              uint32_t addr)
```

Set core base address.

This function sets the base address for the core in the system. All other accesses are offset from this base address.

Parameters

addr	Base address for core.

iob_cache_axi_csrs_set_INVALIDATE()

```
void iob_cache_axi_csrs_set_INVALIDATE (
              uint8_t value)
```

Set INVALIDATE value. Invalidate the cache data contents by writing any value to this register.

Parameters

value	INVALIDATE Value.

iob_cache_axi_csrs_set_RST_CNTRS()

```
void iob_cache_axi_csrs_set_RST_CNTRS (
              uint8_t value)
```

Set RST_CNTRS value. Reset read/write hit/miss counters by writing any value to this register.

Parameters

value	RST_CNTRS Value.

iob_read()

```
uint32_t iob_read (
               uint32_t addr,
               uint32_t data_w)
```

Read access function prototype.

Parameters

addr	Address to write to.
data_w	Data width in bits.

Returns

uint32_t Read data value.

iob_write()

```
void iob_write (
               uint32_t addr,
               uint32_t data_w,
               uint32_t value)
```

Write access function prototype.

Parameters

addr	Address to write to.
data_w	Data width in bits.
value	Value to write.