IOb-Cache

A Configurable Cache

June 7, 2022

User Guide, 0.1 , Build 935c572

The following table shows the revision history for this document.

Date	Summary
May/30/2022	Document released with product version 0.1 .

Contents

List of Tables

List of Figures

1 Introduction

The IObundle CACHE is an open-source pipelined-memory cache. It is a performance-wise and highly configurable IP core. The cache core is isolated from the processor and memory interfaces in order to make it easy to adopt new processors or memory controllers while keeping the core functionality intact. It implements a simple front-end native interface. It also implements an AXI4 interface with configurable data width which allows maximum use of the available memory bandwidth. The IObundle CACHE can be implemented as a Direct-Mapped cache or K-Way Set-Associative cache. It supports both fixed write-through not-allocate policy and write-back policy.

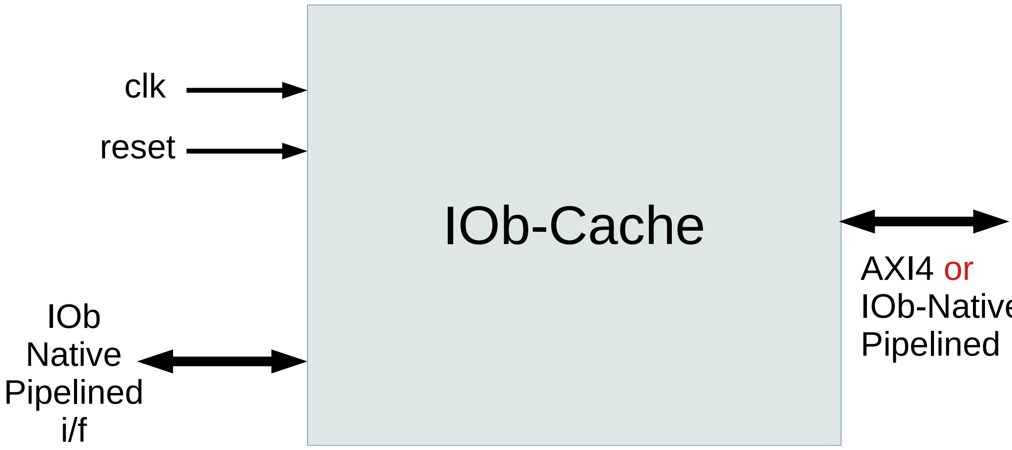

Figure 1: IP Core Symbol.

1.1 Features

- Pipelined-memory (1 request/clock-cycle)
- AXI4 interface with configurable data width
- Simple front-end native interface
- Direct-Mapped or K-Way Set-Associative
- Fixed write-through not-allocate policy
- Write-back policy

1.2 Benefits

- Compact and easy to integrate hardware and software implementation
- Can fit many instances in low cost FPGAs and ASICs
- Low power consumption

1.3 Deliverables

- ASIC or FPGA synthesized netlist or Verilog source code, and respective synthesis and implementation scripts

- ASIC or FPGA verification environment by simulation and emulation
- Bare-metal software driver and example user software
- User documentation for easy system integration
- Example integration in IOb-SoC (optional)

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.

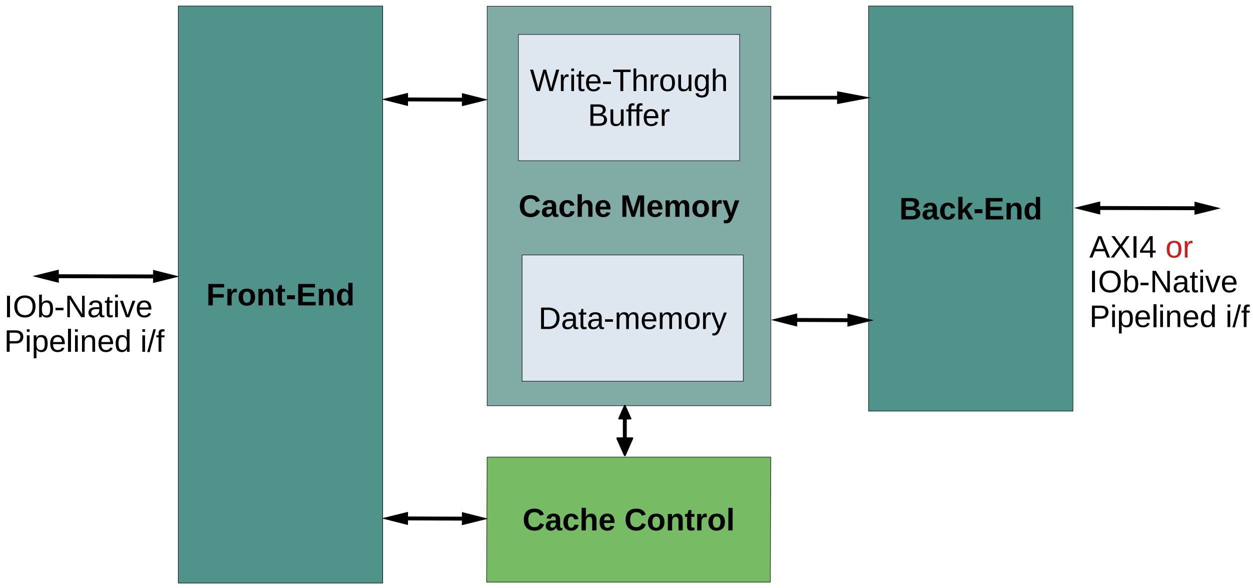

Figure 2: High-Level Block Diagram.

FRONT-END Front-end interface.

CACHE MEMORY This block implements the cache memory.

BACK-END INTERFACE This block interfaces with the system level or next-level cache.

CACHE CONTROL Cache control block.

FRONT-END Front-end interface.

CACHE MEMORY This block implements the cache memory.

BACK-END INTERFACE This block interfaces with the system level or next-level cache.

CACHE CONTROL Cache control block.

2.2 Configuration

2.2.1 Macros

The IP core has no user-definable macros other than the default configuration parameter values, described in Section 2.2.2 if available.

2.2.2 Parameters

This IP core has no synthesis parameters.

2.3 Interface Signals

Name	Direction	Width	Description
clk	INPUT	1	System clock input
rst	INPUT	1	System reset, asynchronous and active high

Table 1: General Interface Signals

Name	Direction	Width	Description
req	INPUT	1	Request signal
addr	INPUT	CTRL_CACHE+ADDR_W-NBYTES_W	Address signal
wdata	INPUT	DATA_W	Write data signal
wstrb	INPUT	NBYTES	Write strobe signal
rdata	OUTPUT	DATA_W	Read data signal
ack	OUTPUT	1	Acknowledgment signal

Table 2: Front-End Interface Signals

Name	Direction	Width	Description
invalidate_out	OUTPUT	1	Cache invalidate signal
wtb_empty_out	OUTPUT	1	Write-through buffer empty signal

Table 3: Invalidate and Write-Through Buffer Empty Chain Interface Signals

Name	Direction	Width	Description
axi_awid	OUTPUT	1	Address write channel ID
axi_awaddr	OUTPUT	AXI_ADDR_W	Address write channel address
axi_awlen	OUTPUT	8	Address write channel burst length
axi_awsize	OUTPUT	3	Address write channel burst size. This signal indicates the size of each transfer in the burst
axi_awburst	OUTPUT	2	Address write channel burst type
axi_awlock	OUTPUT	1	Address write channel lock type
axi_awcache	OUTPUT	4	Address write channel memory type. Transactions set with Normal Non-cacheable Modifiable and Bufferable (0011).
axi_awprot	OUTPUT	3	Address write channel protection type. Transactions set with Normal, Secure, and Data attributes (000).
axi_awqos	OUTPUT	4	Address write channel quality of service
axi_awvalid	OUTPUT	1	Address write channel valid
axi_awready	INPUT	1	Address write channel ready
axi_wid	OUTPUT	1	Write channel ID
axi_wdata	OUTPUT	AXI_DATA_W	Write channel data
axi_wstrb	OUTPUT	AXI_DATA_W/8	Write channel write strobe
axi_wlast	OUTPUT	1	Write channel last word flag
axi_wvalid	OUTPUT	1	Write channel valid
axi_wready	INPUT	1	Write channel ready
axi_bid	INPUT	1	Write response channel ID
axi_bresp	INPUT	2	Write response channel response
axi_bvalid	INPUT	1	Write response channel valid
axi_bready	OUTPUT	1	Write response channel ready
axi_arid	OUTPUT	1	Address read channel ID
axi_araddr	OUTPUT	AXI_ADDR_W	Address read channel address
axi_arlen	OUTPUT	8	Address read channel burst length
axi_arsize	OUTPUT	3	Address read channel burst size. This signal indicates the size of each transfer in the burst
axi_arburst	OUTPUT	2	Address read channel burst type
axi_arlock	OUTPUT	1	Address read channel lock type
axi_arcache	OUTPUT	4	Address read channel memory type. Transactions set with Normal Non-cacheable Modifiable and Bufferable (0011).
axi_arprot	OUTPUT	3	Address read channel protection type. Transactions set with Normal, Secure, and Data attributes (000).
axi_arqos	OUTPUT	4	Address read channel quality of service
axi_arvalid	OUTPUT	1	Address read channel valid
axi_arready	INPUT	1	Address read channel ready
axi_rid	INPUT	1	Read channel ID
axi_rdata	INPUT	AXI_DATA_W	Read channel data
axi_rresp	INPUT	2	Read channel response
axi_rlast	INPUT	1	Read channel last word
axi_rvalid	INPUT	1	Read channel valid
axi_rready	OUTPUT	1	Read channel ready

Table 4: Back-End Interface Signals

3 Usage

Figure 4 illustrates how to instantiate the IP core and, if applicable, the required external blocks.

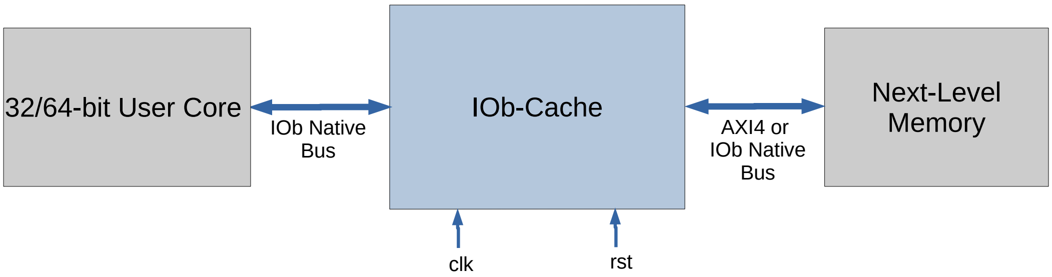

Figure 3: Core Instance and Required Surrounding Blocks

bla bla bla...

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.

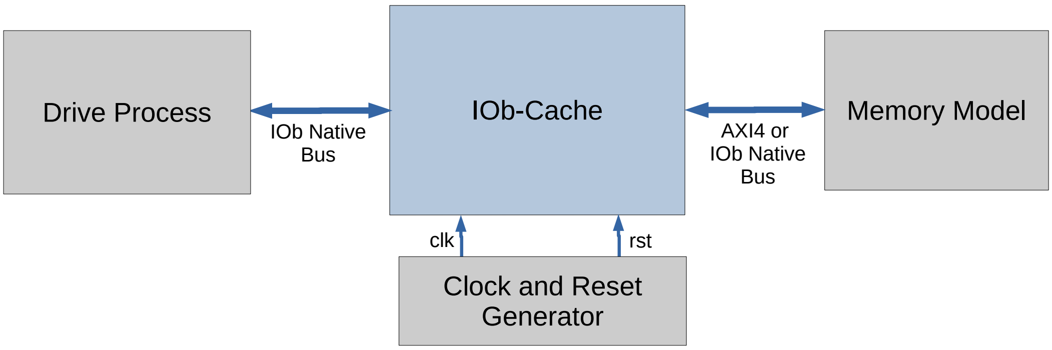

Figure 4: Testbench Block Diagram

In this preliminary version, simulation is not yet fully functional. The provided testbench merely allows compilation for simulation, and drives the clock and reset signals. Behavioural memory models to allow pre-synthesis simulation are already included. In the case of ROMs, their programming data is also included in the form of .hex files.

3.2 Synthesis

A simple `.tcl` script is provided for the Cadence Genus synthesis tool. The script reads the technology files, compiles and elaborates the design, and proceeds to synthesize it. The timing constraints are contained within the constraints file provided, or provided in a separate file.

After synthesis, reports on silicon area usage, power consumption, and timing closure are generated. A post-synthesis Verilog file is created, to be used in post-synthesis simulation.

In this preliminary version, synthesis of the IP core without the memories is functional. The memories are for now treated as black boxes.

It is important not to include the memory models provided in the simulation directory in synthesis, unless they are to be synthesised as logic. In a next version, the memories will be generated for the target technology and included.

4 Implementation Results

4.1 FPGA

This section presents FPGA implementation results.

Resource	Used
LUTs	2084
Registers	1157
DSPs	0
BRAM	0

Table 5: AMD Kintex Ultrascale FPGAs.

Resource	Used
ALM	804
FF	590
DSP	0
BRAM blocks	68

Table 6: Intel Cyclone V GT FPGAs.

4.2 ASIC

No ASIC implementation results have been obtained for this IP core.