TRISTAN

Together for RISc-V Technology and ApplicatioNs

WI2.3.5 VLSI Architecture Description and Design Specifications

Document Number	D2.1
Primary Author(s)	Timo Solla
Beneficiary/ies	VLSI
Document Date	2023-09-13
Document Version	V 1.0
Distribution Level	Public
Reference DoA	AMD-101095947-2
Project Coordinator	Patrick Pype, NXP Semiconductors, patrick.pype@nxp.com
Project Website	www.tristan-project.eu
JU Grant Agreement Number	101095947

1. Introduction

1.1 General Information

VSRISCV is a simple 32-bit RISC-V ISA CPU core that is capable of running standard off-the-shelf software protocols (such as Ethernet) under the latest Linux operating system kernel at low clock speeds.

The starting point of the project is single issue option of the Bi-RISC-V core found in https://github.com/ultraembedded/. Initial evaluation of this core revealed some weaknesses with the real software which are investigated and improved in TRISTAN project. These pertain to: cache architecture, low latency memory mapped IO, modification of the cache to support single-port SRAM, compact bridge with co-processors, MMU performance improvement, as well as cleaning/rewriting (from Verilog to VHDL) and documenting everything to a commercial IP block level.

VSRISCV implements ISA extensions “I” and “M” and it is capable of running operating systems like Linux. The core has separate data and instruction buses and default cache sizes are 16KB each. The design is written in VHDL. Design hierarchy is shown below.

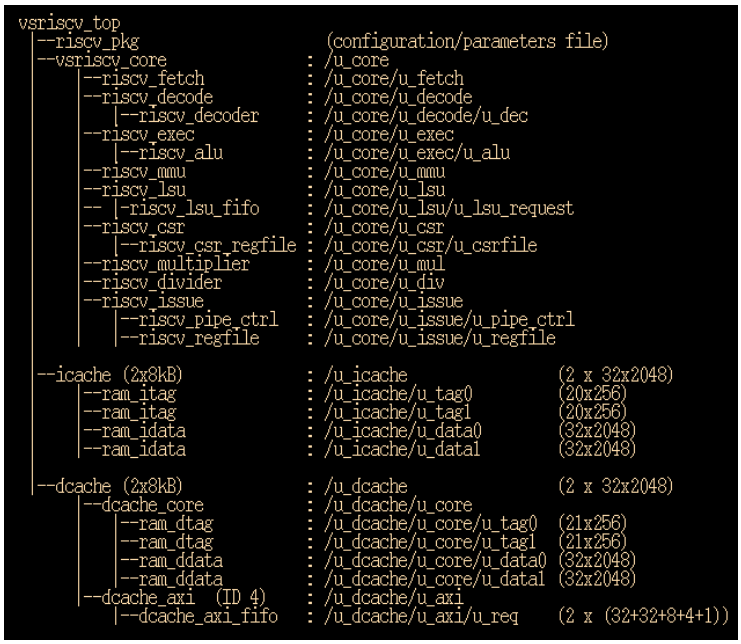

Figure ***1****: WI2.3.5 VHDL Design hierarchy*

1.2 Purpose and Scope

The document describes the RISC-V core developed by VLSI Solution. All information in the document is public and can be freely distributed.

1.3 Acronyms and Definitions

Acronym	Description
ASIC	Application Specific Integrated Circuit
CSR	Control and Status Registers
DDR	Double data rate high-speed dynamic random access memory
DRAM	Dynamic Random Access Memory
FPGA	Field Programmable Gate Array
IoT	Internet of Things
IP	Intellectual Property
ISA	Instruction Set Architecture
LSB	Least Significant Bit
MMU	Memory Management Unit
MSB	Most Significant Bit
OS	Operating System
RAM	Random Access Memory
SoC	System on a Chip
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver Transmitter

2. Architecture

2.1 Place in the System

To build a SoC capable of running a Linux operating system the processor core vsriscv_top needs peripherals, memory controller and means to upload software to DDR memory. Below is shown a minimal FPGA hardware for this purpose.

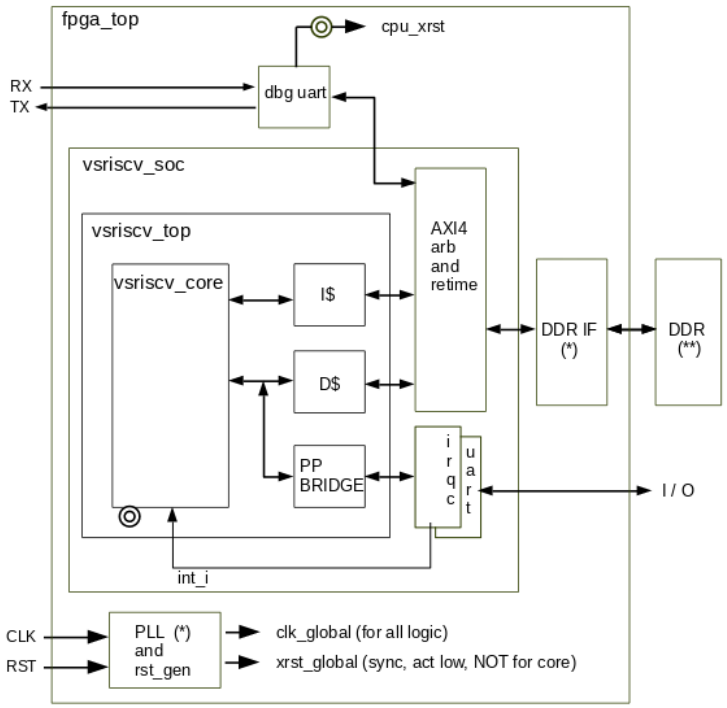

Figure ***2****: WI2.3.5 minimal FPGA hardware*

(*) : FPGA IP block

(**): External DDR memory chip (DDR2, DDR3 etc.)

To run Linux OS at least interrupt controller and UART are needed. In this setup the core stays in reset after external reset is de-asserted. Dbg_uart acts as a third AXI4 master and it is used to upload the software to the DDR memory by using UART protocol. First a command is given (rd or wr) followed by data length, address and data. Once the software is uploaded to the DDR memory the core reset register in dbg_uart module is de-asserted with UART command and the core boots up.

2.2 Block Diagram

The vsriscv_top module consists of a 32-bit riscv core, instruction cache, data cache and peripheral bridge.

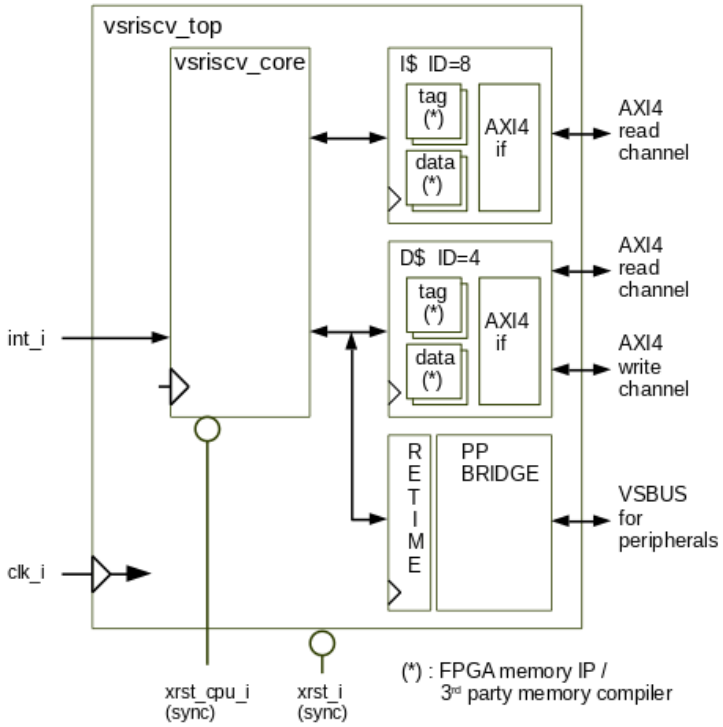

Figure ***3****. WI2.3.5 vsriscv_top module*

2.3 Interfaces

Icache and dcaches both interface to DDR memory via an AXI4 bus master.

Peripherals are connected to riscv core's data bus aside with data cache. All non-cacheable accesses are assumed to be peripheral transactions (outside the range of core parameters MEM_CACHE_ADDR_MIN and MEM_CACHE_ADDR_MAX. There could also be ROM memory if needed but the deliveries will have none.

2.3.1 AXI4 cache interfaces

Cache interfaces are a standard AXI4 bus except that icache only uses read channel signals (read only cache) as the dcache needs both the read and write channels.

2.3.2 VSBUS peripheral bus

For peripherals a very simple bus interface is added through a bridge. This interface is compliant with a bus used in vsdsp4 (Proprietary DSP processor of VLSI Solution).

This bus does single word in single cycle transactions only. The peripherals cannot halt the processor. The signals and bus interface protocol are shown below.

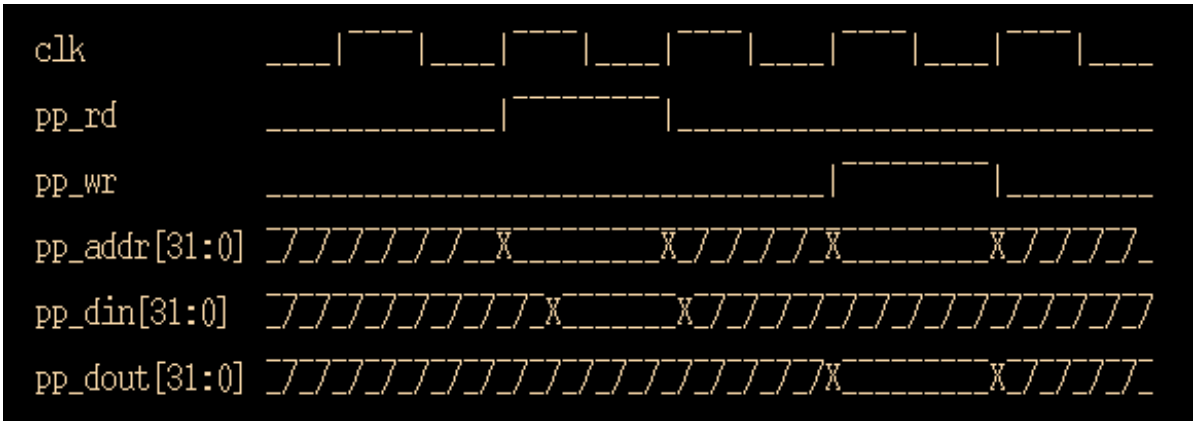

Figure **4***. WI2.3.5 signals and bus interface protocol*

Peripheral module's outputs can be connected to core pp_din bus using OR- or AND-resolution.

2.4 Sub-Modules

The design consists of three main modules being the vsriscv_core, icache and dcache.

2.4.1 Vsriscv_core

The core is designed to be simple, small, and efficient, with a focus on low-power and low-latency applications such as Internet of Things (IoT) devices. It has a single issue in-order pipeline that supports the basic RV32IM riscv ISA. It also suports Zicsr extension so operating systems like Linux can be used.

Extension definitions:

- I : Integer
- M : Integer multiplication and division
-Zicsr : Control and Status Register Access / Privileged Architecture

    The CSR access instructions as well as the exception and interrupt system

    - CSR access: csrrw, csrrs, csrrc, csrrwi, csrrsi, csrrci

- Environment: mret, wfi

The multiplier is a 32-bit multiplier, and it executes mul, mulh, mulhsu and mulhu instructions in one clock cycle. Divider executes div, divu, rem and remu instructions in 32 clock cycles.

Block diagram of the core is shown below

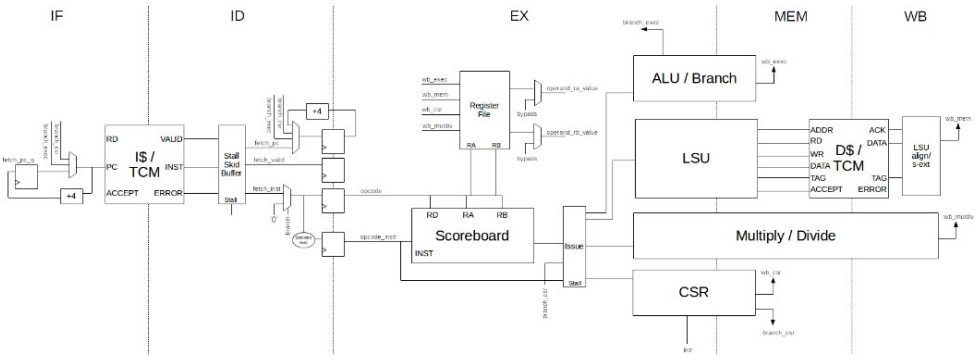

Figure ***5****: WI2.3.5 Block diagram of the core*

2.4.2 Icache

Instruction cache is a 32-bit set associative cache and has four parameters as VHDL generics:

```
generic (AXI_ID      : std_logic_vector      := X"8";
         WAYS        : natural range 1 to 8  := 2;
         LINE_ADDR_W : natural range 6 to 12 := 8;
         LINE_SIZE_W : natural range 2 to 8  := 5 );
```

Parameter default values make a 2-way cache with 2^8 lines and 2(5-2) words on a line. This makes a total of 16kB RAM. The VHDL code is written such that conventional single port RAMs can be used. In two way cache there are two tag RAM and two data RAMs. The RAMs must be semi-synchronous (address, data inputs, rd- and wr-enable registered). Memory sizes depend from parameters and default values make tag RAMas 20 bits (32 - LINE_ADDR_W – LINE_SIZE_W + 1 VALID bit) x 256 (2LINE_ADDR_W). The data RAM sizes are 32bits x 2048

(2(LINE_ADDR_w+LINE_SIZE_W-2)). As icache is a read only the AXI4 write channel is omitted. The cache implementation is shown in Figure 6:

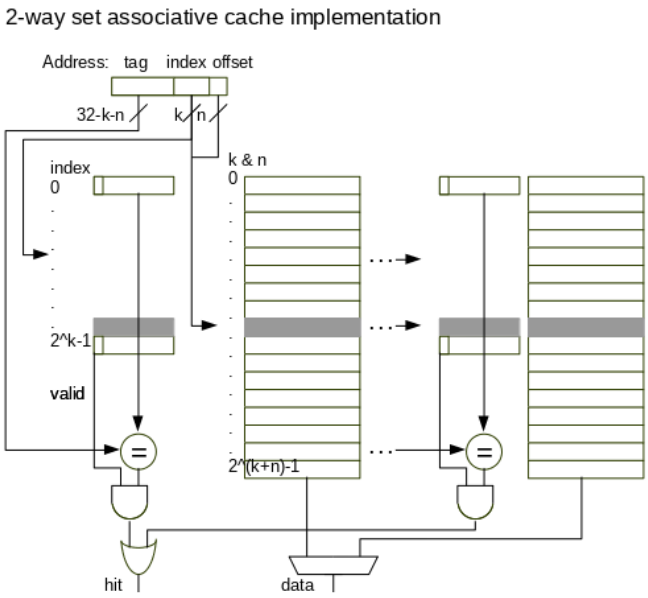

Figure 6: WI2.3.5 Cache implementation

2.4.3 Dcache

Data cache has the same parameters and same type of memories as icache except that the tag RAMs have one more bit as they also store the LINE DIRTY flag. Dcache also has AXI4 write channel to store dirty lines to main memory.

The cache implementation principle is identical with the icache (of the previous paragraph).

2.5 Clocking Strategy

All the modules in vsriscv_top are clocked with the same single clock source. The clock is externally generated with a PLL and can be set to any value which is less than the maximum operating frequency of the block. In ASIC synthesis the clock gating insertion can be enabled to save power.

2.6 Reset Strategy

Active low reset is synchronous, and it must be asserted and synchronized outside this block. Clock's falling edge can be used to de-assert the xrst_i signal. This ensures that no race situation can occur in any of the registers. Once the reset is de-asserted the caches first initialize their tag memories and stall riscv core until initialization is finished. After that core starts executing instruction from a given boot address (generic BOOT_ADDR, default 0x8000000). When reset is initialized, it is assumed that the rest of the system is already initialized (like DDR memory init done).

2.7 Power Management Strategy

Module itself has no power saving features. It is designed in a way that the clock can be halted, and the system maintains its state. The clock frequency can also be adjusted dynamically.

2.8 Debugging Strategy

Since the vsriscv core can be run in the FPGA, the debugging can be done in combination of VHDL simulation and FPGA prototyping.

3. Design Specifications

3.1 Pin-list

Vsriscv_top parameters and pins are listed in the VHDL entity declaration below

```
entity vsriscv_top is
  generic ( BOOT_ADDR         : slv_t := X"80000000";
            CPU_ID            : slv_t := X"00000000";  -- riscv core hart id
            ICACHE_AXI_ID     : slv_t := X"8";
            DCACHE_AXI_ID     : slv_t := X"4";
            MEM_CACHE_ADDR_MIN : slv_t := X"80000000";
            MEM_CACHE_ADDR_MAX : slv_t := X"8FFFFFFF" );
  port ( clk_i          : in  std_logic;
         xrst_i         : in  std_logic;  -- syncr reset
```

```
        xrst_cpu_i      : in  std_logic;
        -- icache AXI read channel
        axi_i_arready_i : in  std_logic;
        axi_i_rvalid_i  : in  std_logic;
        axi_i_rdata_i   : in  slv_31_00;
        axi_i_rresp_i   : in  slv_01_00;
        axi_i_rid_i     : in  slv_03_00;
        axi_i_rlast_i   : in  std_logic;
        axi_i_arvalid_o : out std_logic;
        axi_i_araddr_o  : out slv_31_00;
        axi_i_arid_o    : out slv_03_00;
        axi_i_arlen_o   : out slv_07_00;
        axi_i_arburst_o : out slv_01_00;
        axi_i_rready_o  : out std_logic;
        -- NOTE: NO AXI write channel in icache
        -- dcache AXI read channel, inputs
        axi_d_arready_i : in  std_logic;
        axi_d_rvalid_i  : in  std_logic;
        axi_d_rdata_i   : in  slv_31_00;
        axi_d_rresp_i   : in  slv_01_00;
        axi_d_rid_i     : in  slv_03_00;
        axi_d_rlast_i   : in  std_logic;
        axi_d_arvalid_o : out std_logic;
        axi_d_araddr_o  : out slv_31_00;
        axi_d_arid_o    : out slv_03_00;
        axi_d_arlen_o   : out slv_07_00;
        axi_d_arburst_o : out slv_01_00;
        axi_d_rready_o  : out std_logic;
        -- dcache AXI write channel
        axi_d_awready_i : in  std_logic;
        axi_d_wready_i  : in  std_logic;
        axi_d_bvalid_i  : in  std_logic;
        axi_d_bresp_i   : in  slv_01_00;
```

```
        axi_d_bid_i     : in  slv_03_00;
        axi_d_awvalid_o : out std_logic;
        axi_d_awaddr_o  : out slv_31_00;
        axi_d_awid_o    : out slv_03_00;
        axi_d_awlen_o   : out slv_07_00;
        axi_d_awburst_o : out slv_01_00;
        axi_d_wvalid_o  : out std_logic;
        axi_d_wdata_o   : out slv_31_00;
        axi_d_wstrb_o   : out slv_03_00;
        axi_d_wlast_o   : out std_logic;
        axi_d_bready_o  : out std_logic;
        --  VSBUS (peripheral bus)
        pp_intr_i       : in  std_logic;       -- perip interrupt, single pulse
        pp_addr_o       : out slv_31_00;
        pp_rd_o         : out std_logic;
        pp_rd_data_i    : in  slv_31_00;
        pp_wr_o         : out slv_03_00;
        pp_wr_data_o    : out slv_31_00 );
end entity riscv_top;
```

The module has three external interface buses:

- Icache AXI4 master read channel
- Dcache AXI4 master read/write channel
- VSBUS peripheral bus

3.2 3.2 Register Map

The icache and dcache do not have any configuration or status registers. In icache tag RAMs the MSB (19 in default configuration) bit is VALID bit and LSB bits (18:0) are TAG_ADDR bits. In dcache the tag RAMs MSB bit (20) is line DIRTY bit, MSB-1 is VALID bit and LSB bits (18:0) are TAG_ADDR bits.

Vsriscv_core has registers as defined in base ISA spec v2.1 and privileged ISA spec v1.11. The register file has 32 integer registers and it is located in

vsriscv_core/riscv_issue/ riscv_regfile/rfile_q[31:0][31:0]. Registers are listed in the next table.

rfile_q		**IP_REGS_BASE_ADDRESS + 0x0000**
Reset Value = 0x0000 0000		
Register	**ABI Name**	**Description (typical use)**
x28-x31	t3-6	Temporaries
x18-x27	s2-11	Saved registers
x12-x17	a2-7	Function arguments
x10-x11	a0-1	Function Arguments/return values
x9	s1	Saved register
x8	s0/fp	Saved register/Frame pointer
x5-x7	t0-2	Temporaries
x4	tp	Thread pointer
x3	gp	Global pointer
x2	sp	Stack pointer
x1	ra	Return address
x0	zero	Hard-wired zero

Table ***1****: WI2.3.5 RiscV register file*

Additionally, the core requires Control and Status Registers (CSRs). These registers control various aspects of the processor's behavior and store status information. These registers are listed in the next table.

CSR REGISTERS		**IP_REGS_BASE_ADDRESS + 0x0000**
Reset Value = 0x0000 0000		
Machine-Level CSRs (32-bit registers)		
Register	**Address**	**Description (typical use)**
MSTATUS	X"300"	Machine status register
MISA	X"301"	Machine ISA register, indicating supported ISA extensions

MIE	X"304"	Machine interrupt enable
MTVEC	X"305"	Machine trap vector
MSCRATCH	X"340"	Machine scratch register
MEPC	X"341"	Machine exception program counter
MCAUSE	X"342"	Machine exception cause
MTVAL	X"343"	Machine trap value (additional information about certain exceptions)
MIP	X"344"	Machine interrupt pending
MCYCLE	X"C00"	Machine performance counters
MTIME	X"C01"	Machine performance counters
MTIMEH	X"C81"	Upper 32 bits of performance counters
MHARTID	X"F14"	Machine hardware thread ID
MTIMECMP	X"7C0"	Timer compare register
MEDELEG	X"302"	Machine exception delegation register
MIDELEG	X"303"	Machine exception delegation register
Supervisor-Level CSRs (32-bit) (Some CSRs are shared with the Machine-Level)		
SSTATUS	X"100"	Supervisor status register
SIE	X"104"	Supervisor interrupt enable
STVEC	X"105"	Supervisor trap vector
SSCRATCH	X"140"	Supervisor scratch register
SEPC	X"141"	Supervisor exception program counter
SCAUSE	X"142"	Supervisor exception cause
STVAL	X"143"	Supervisor trap value
SIP	X"144"	Supervisor interrupt pending
SATP	X"180"	Supervisor Address Translation and Protection Register [31] MODE MODE=1 uses Sv32 Address Translation [30:22] ASID Address Space Identifier [21:0] PPN Physical Page Number of the root page table

DCACHE control CSRs		
DFLUSH	X"3A0"	Flush dcache
DWRITEBACK	X"3A1"	Writeback dcache line
DINVALIDATE	X"3A2"	Invalidate dcache line

*Table **2**: WI2.3.5 Riscv control and status registers*

3.3 Functional Description

3.3.1 Vsriscv_core

Vsriscv_core has the processor core. It fetches the instructions and data from the memories and executes them according to the 32-bit ISA instruction set.

3.3.2 Icache

Instruction cache state diagram is shown below.

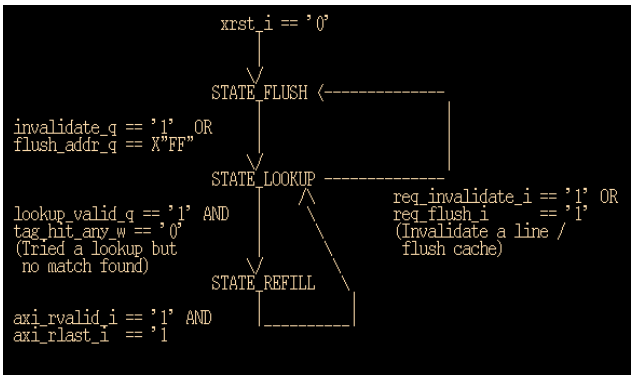

*Figure **7**: WI2.3.5 Instruction cache state diagram*

After reset the tag RAMs must first be initialized, and it takes 2LINE_SIZE_W clock cycles. After that the req_accept_o is set, and cache is ready to service any core read request. Individual lines can be invalidated by setting req_invalidate_i high for one clock cycle when req_accept_o in high. Similarly, the tag RAMs can be flushed by setting req_flush_i high for one clock cycle when cache is ready to accept requests.

3.3.3 Dcache

Data cache is a set associative cache like icache and has the same parameters and memory configurations. Since data bus is read and write bus there is one more bit in the tag RAMs though, namely the LINE_DIRTY flag. Dcache state machine is shown below.

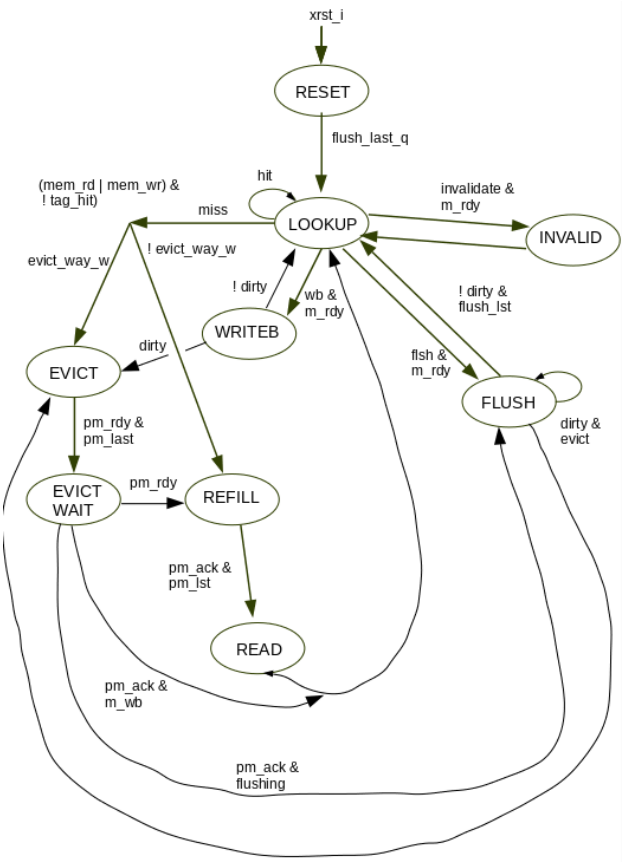

Figure **8***: WI2.3.5 Dcache state machine*

4. References

Bi-RISC-V core https://github.com/ultraembedded/

ISA spec v2.1	https://github.com/ultraembedded/riscv/tree/master/doc/riscv_isa_spec.pdf
privileged ISA spec v1.11.	https://github.com/ultraembedded/riscv/tree/master/doc/riscv_privileged_spec.pdf