TRISTAN

Together for RISc-V Technology and ApplicatioNs

WI 2.3.5 VLSI 32-bit RISC-V based on Bi-RISC-V

D2.2 Design and Implementation of RISC-V Cores and Extensions

Document Number	WI 2.3.5
Primary Author(s)	Timo Solla and Teppo Karema
Beneficiary/ies	VLSI
Document Date	11.10.2024
Document Version	FINAL
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

TRISTAN has received funding from the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement nr. 101095947. The KDT JU receives support from the European Union's Horizon Europe's research and innovation programme and Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain,

Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey

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/**. This core was at the time the project was prepared the simplest (and only) 32-bit RISC-V core that was able to boot the Linux operating system right away without any significant modifications. Initial evaluation of this core revealed weaknesses which are investigated and improved in Tristan project. Our deliverable includes a complete RISC-V core (VSRV1) including memory management and common AXI interface to LPDDR controller of the DDR memory. Also, the peripheral interface as well as three IO peripherals connected to it (UART, timer and interrupt handler) are included in the deliveries. FPGA users can make a complete processor system by using our deliverables right away because FPGAs typically have the required macro blocks available. IC users, however, will need to use their own libraries for the LPDDR control interface, memories and standard cells. Our in-house advanced peripherals (ethernet, SPI, fast buffered UART) will not be included in the deliveries either even though we will use them in WI6.2.3.

All HW description of VSRV1 are written in VHDL. This minimizes the cost of simulators for companies because mixed Verilog and VHDL simulation requires typically double the amount of licenses. Also, it makes the deliverable uniform.

1.2 Purpose and Scope

The document describes the design and implementation of 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
LEC	Conformal Logic Equivalence Checker
LPDDR	Low power double data rate random access memory
LSB	Least Significant Bit
MMU	Memory Management Unit
MSB	Most Significant Bit
OS	Operating System
RAM	Random Acceess Memory
SoC	System on a Chip
SRAM	Static Random Acceess Memory
UART	Universal Asynchronous Receiver Transmitter

2 Update on Architecture

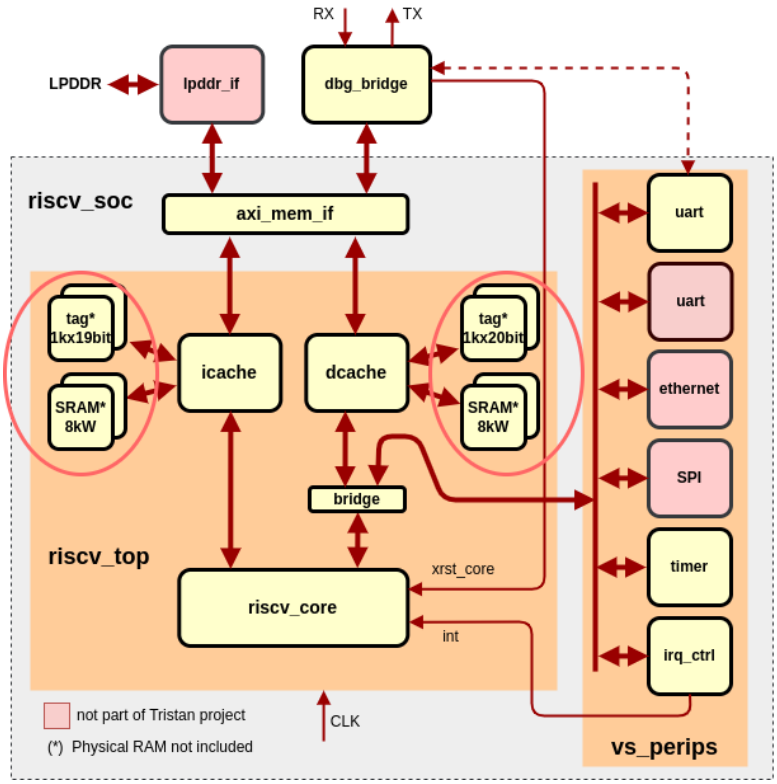

Figure 1: WI 2.3.5 - Architecture changes

The only architectural change of the VSRV1 processor core is the size of icache and dcache memories. The current implementation has double amount of SRAM in

caches compared to the initial plan in the D2.1 deliverable.Design and Implementation

3 Design and Implementation

3.1 Design Methodology

The target application of the VSRV1 processor is to be a peripheral protocol processor, executing for example network related packet communication under Linux operating system. For this application, it does not need to be very powerful. The design optimization objective is therefore to achieve Linux OS compatibility and optimize the ratio of performance and number of logic gates.

Two basic tests were used for the performance measure: NBench and Linux boot time. For consistent measures, the Linux boot was slightly modified by removing some randomized self-tests.

The architecture simulation and optimization were done by using ExactStep simulator (https://github.com/ultraembedded/exactstep). The source code of it was modified based on our architectural modification experiments. Also, the LPDDR as well as the cache and loop control were modelled and added to the ExactStep simulator to achieve almost cycle accurate results. Such accuracy is not achievable with most of the other simulators such as QEMU or Spike.

For the VSRV1 version, most of the improvements were selected based on the performance evaluation result being “mandatory” and “realistic to implement within one year”. The selection of the steps is explained in the table of Chapter 3.2.2.

The VSR1 was first partitioned and simulated as subblocks. Verified blocks were then included into the whole SoC system. Since VLSI Solution uses only in-house libraries and memory compilers the cache implementation was chosen such that silicon proven single port rams can be used

The peripheral bus was moved from the AXI bus to a memory mapped block similar to the one used in other VLSI's processors. This way VSRV1 can reuse all our peripheral libraries.

When system simulations showed correct functionality, the design was transferred to the FPGA. FPGA testing ensured that hardware-software co-operation well as LInux OS compatibility was guaranteed. It is also a lot faster to test a large quantity of software in FPGA than in simulations. The layout backend was done by using common commercial tools only.

3.2 Pin-list

3.2.1 Pin list of riscv_top module

PIN NAME	WIDTH	TYPE	NOTES
clk_i	1	in	Master clock
xrst_i	1	in	This is actually “enable”
xrst_cpu_i	1	in	
TEST			
scanena_mem_i	1	in	
scanin_mem_i	1	in	
scanout_mem_o	1	out	
RAM CONFIGURATION			
ram_delay_i	2	in	Select the rd/wr delay
ram_xpd_i	1	in	Power down the memory
MEM BIST INTERFACE (NOTE: mem count is 2x WAYS as there are tag/data rams) rd/wr select is 1-hot style in order like ...t1,d1,t0,d0			
mbist_ena_i	1	in	
imbist_rd_i	4	in	
imbist_wr_i	4	in	
dmbist_rd_i	4	in	
dmbist_wr_i	4	in	
mbist_addr_i	13	in	
mbist_data_i	32	in	
mbist_data_o	32	out	
AXI INTERFACE			
axi_i_awready_i	1	in	
axi_i_wready_i	1	in	
axi_i_bvalid_i	1	in	
axi_i_bresp_i	2	in	
axi_i_bid_i	4	in	
axi_i_arready_i	1	in	
axi_i_rvalid_i	1	in	
axi_i_rdata_i	32	in	
axi_i_rresp_i	2	in	
axi_i_rid_i	4	in	
axi_i_rlast_i	1	in	
axi_d_awready_i	1	in	
axi_d_wready_i	1	in	

axi_d_bvalid_i	1	in	
axi_d_bresp_i	2	in	
axi_d_bid_i	4	in	
axi_d_arready_i	1	in	
axi_d_rvalid_i	1	in	
axi_d_rdata_i	32	in	
axi_d_rresp_i	2	in	
axi_d_rid_i	4	in	
axi_d_rlast_i	1	in	
axi_i_awvalid_o	1	out	
axi_i_awaddr_o	32	out	
axi_i_awid_o	4	out	
axi_i_awlen_o	8	out	
axi_i_awburst_o	2		
axi_i_wvalid_o	1	out	
axi_i_wdata_o	32	out	
axi_i_wstrb_o	4	out	
axi_i_wlast_o	1	out	
axi_i_bready_o	1	out	
axi_i_arvalid_o	1	out	
axi_i_araddr_o	32	out	
axi_i_arid_o	4	out	
axi_i_arlen_o	8	out	
axi_i_arburst_o	2	out	
axi_i_rready_o	1	out	
axi_d_awvalid_o	1	out	
axi_d_awaddr_o	32	out	
axi_d_awid_o	4	out	
axi_d_awlen_o	8	out	
axi_d_awburst_o	2	out	
axi_d_wvalid_o	1	out	
axi_d_wdata_o	32	out	
axi_d_wstrb_o	4	out	
axi_d_wlast_o	1	out	
axi_d_bready_o	1	out	
axi_d_arvalid_o	1	out	
axi_d_araddr_o	32	out	
axi_d_arid_o	4	out	
axi_d_arlen_o	8	out	
axi_d_arburst_o	2	out	
axi_d_rready_o	1	out	

VSBUS			
pp_intr_i	1	in	
pp_addr_o	1	out	
pp_rd_o	1	out	
pp_rd_data_i	1	in	
pp_wr_o	1	out	
pp_wr_data_o	1	out	

Table 1: WI 2.3.5 - Pin list of riscv_top module

3.2.2 Pin list of VSRV1 module

PIN NAME	WIDTH	TYPE	NOTES
clk_i	1	in	
xrst_i	1	in	This is actually “enable”
xrst_cpu_i	1	in	
TEST			
tmode_i	1	in	
scanena_i	1	in	
scanin_i	1	in	NC in FPGA
scanout_o	1	out	NC in FPGA
scanena_mem_i	1	in	NC in FPGA
scanin_mem_i	1	in	NC in FPGA
scanout_mem_o		in	NC in FPGA
RAM CONFIGURATION			
ram_delay_i	2	in	Select the rd/wr delay
ram_xpd_i	1	in	Power down the memory
LPDDR INTERFACE			
axi_awready_i	1	in	
axi_wready_i	1	in	
axi_bvalid_i	1	in	
axi_bresp_i	2	in	
axi_bid_i	4	in	
axi_arready_i	1	in	
axi_rvalid_i	1	in	
axi_rdata_i	32	in	
axi_rresp_i	2	in	
axi_rid_i	4	in	

axi_rlast_i	1	in	
axi_awvalid_o	1	out	
axi_awaddr_o	32	out	
axi_awid_o	4	out	
axi_awlen_o	8	out	
axi_awburst_o	2	out	
axi_wvalid_o	1	out	
axi_wdata_o	32	out	
axi_wstrb_o	4	out	
axi_wlast_o	1	out	
axi_bready_o	1	out	
axi_arvalid_o	1	out	
axi_araddr_o	32	out	
axi_arid_o	4	out	
axi_arlen_o	8	out	
axi_arburst_o	2	out	
axi_rready_o	1	out	
ETHERNET INTERFACE			
CO-PROCESSOR IF			
vs_clk_i	1	in	
vs_xrst_i	1	in	
vs_rd_i	1	in	
vs_wr_i	1	in	
vs_ab_i	10	in	
vs_dbin_i	16	in	
vs_dbout_o	16	out	
vs_ethrx_intr_o	1	out	
vs_eth_memtest_i	2	in	
vs_rx_orun_o	1	out	
vs_rx_orun_rst_i	1	in	
vs_rxmem_i	1	in	
MDIO (configuration)			
MDC	1	out	
MDO	1	out	
MDI	1	in	
MDO_xena	1	out	
RGMII			
TXC	1	out	
TX_CTL	1	out	
TXD	4	out	
RXC	1	in	

RX_CTL	1	in	
RXD	4	in	
XRST_ETH	1	out	
UART			
rv_mux	1	in	
uart_txd_vs_i	1	in	
uart_txd_i	1	in	
uart_rxd_o	1	out	
uart2_txd_i	1	in	
uart2_rxd_o	1	out	
rx3_i	1	in	
tx3_o	1	out	
rx4_i	1	in	
tx4_o	1	out	
SPI			
spi_miso_i	1	in	
spi_clk_o	1	out	
spi_cs_o	1	out	
spi_mosi_o	1	out	
GPIO			
gpio_i	12	in	
gpio_oena_o	12	out	
gpio_o	12	out	
pp_rd_o	1	out	

Table 2: WI 2.3.5 - Pin list of VSRV1 module

3.3 Register Map

Registers in riscv_core are defined in Unprivileged Specification and Privileged Specification of RISC-V. This core supports extensions I, M, S_U, Zicsr and Zifence. I-extension is always enabled so the register file size is always 32 registers where register 0 is a zero vector. Zicsr is always enabled so all of the csr registers are present. When supervisor/user privilege levels (S_U) are enabled the privilege level registers for those are present. Otherwise only the machine mode registers are present. The MMU module can be excluded via parameter if address virtualization is not required.

The riscv_core has four user specified parameters (VHDL generics) that are set prior to synthesis/simulation. These user definable core parameters, their type and default values are

- BOOT_ADDR [31:0] = 0x80000000
- CPU_ID [31:0] = 0x00000000
- CACHE_ADDR_L [31:0] = 0x80000000 - cacheable address low limit
- CACHE_ADDR_H [31:0] = 0xFFFFFFFF - cacheable address high limit

Both caches have been set as associative type. They both have parameters to change their implementation and size. User parameters, type and default values are

- ICACHE_AXI_ID [3:0] = 0x8
- DCACHE_AXI_ID [3:0] = 0x4
- WAYS [1 to 8] = 2 - Number of ways (1,2,4,8 allowed)
- LINE_ADDR_W [6 to 12] = 10 - Line address(2^n words)
- LINE_SIZE_W [2 to 8] = 5 - Line size (2^n bytes)

3.3.1 Peripheral registers

The base address of peripheral registers is 0x10000000. Offset addresses relative to the base address are in the following tables.

IRQ Registers

Offset	Register	Bits	Field Name	Type	Description
0x1000_0000	IRQ_ISR	INTS_G-1:0	STATUS	RW	Pending interrupt (unmasked) bitmap
0x1000_0004	IRQ_IPR	INTS_G-1:0	PENDING	R	Pending interrupts (masked) bitmap
0x1000_0008	IRQ_IER	INTS_G-1:0	ENABLE	RW	Interrupt enable mask
0x1000_000C	IRQ_IAR	INTS_G-1:0	ACK	W	Bitmap of interrupts to acknowledge
0x1000_0010	IRQ_SIE	INTS_G-1:0	SET	W	Bitmap of interrupts to enable
0x1000_0014	IRQ_CIE	INTS_G-1:0	CLR	W	Bitmap of interrupts to disable
0x1000_0018	IRQ_IVR	31:0	VECTOR	RW	Highest priority active interrupt number
0x1000_001C	IRQ_MER	1	HIE	RW	Hardware interrupt enable
	IRQ_MER	0	ME	RW	Master enable

Table 3: WI 2.3.5 - IRQ Registers

Timer Registers

Offset	Timer Register	Bits	Field Name	Type	Description
0x1100_0008	TIMER_CTRL0	1	INTERRUPT	RW	Interrupt enable
	TIMER_CTRL0	2	ENABLE	RW	Timer enable§
0x1100_00	TIMER_CMP	31:0	VALUE	RW	Match value

0C	0				
0x1100_0010	TIMER_VAL0	31:0	CURRENT	RW	Current timer value
0x1100_0014	TIMER_CTRL1	1	INTERRUPT	RW	Interrupt enable
	TIMER_CTRL1	2	ENABLE	RW	Timer enable§
0x1100_0018	TIMER_CMP1	31:0	VALUE	RW	Match value
0x1100_001C	TIMER_VAL1	31:0	CURRENT	RW	Current timer value

Table 4: WI 2.3.5 - Timer Registers

UART Registers

Offset	Register	Bits	Field Name	Type	Description
0x1200_0000	ULITE_RX	7:0	DATA	R	Data byte
0x1200_0004	ULITE_TX	7:0	DATA	W	Data byte
0x1200_0008	ULITE_STATUS	4	IE	R	Interrupt enabled
	ULITE_STATUS	3	TXFULL	R	Transmit buffer full
	ULITE_STATUS	2	TXEMPTY	R	Transmit buffer empty
	ULITE_STATUS	1	RXFULL	R	Receive buffer full
	ULITE_STATUS	0	RXVALID	R	Receive buffer not empty
0x1200_000C	ULITE_CONTROL	4	IE	RW	Interrupt enable
	ULITE_CONTROL	1	RST_RX	RW	Flush Rx Buffer
	ULITE_CONTROL	0	RST_TX	RW	Flush Tx Buffer

Table 5: WI 2.3.5 - UART Registers

SPI Registers

Offset	Register	Bits	Name	Type	Description
0x1300_00	SPI_DGIE	31	GIE	RW	Global interrupt enable

1C	R				
0x1300_0020	SPI_IPISR	2	TX_EMPTY	RW	Tx FIFO empty interrupt status
0x1300_0028	SPI_IPIER	2	TX_EMPTY	RW	Tx FIFO interrupt enable
0x1300_0040,	SPI_SRR	31:0	RESET	RW	Software FIFO reset
	SPI_CR	0	LOOP	RW	Loopback enable (MOSI to MISO)
0x1300_0060	SPI_CR	1	SPE	RW	SPI Enable
	SPI_CR	2	MASTER	RW	Master mode (slave not supported)
	SPI_CR	3	CPOL	RW	Clock polarity
	SPI_CR	4	CPHA	RW	Clock phase
	SPI_CR	5	TXFIFO_RST	RW	Tx FIFO reset
	SPI_CR	6	RXFIFO_RST	RW	Rx FIFO reset
	SPI_CR	7	MANUAL_SS	RW	Manual chip select mode
	SPI_CR	8	TRANS_INHIBIT	RW	Transfer inhibit
	SPI_CR	9	LSB_FIRST	RW	Data LSB first (1) or MSB first (0)
	SPI_CR	0	RX_EMPTY	RW	Rx FIFO empty
0x1300_0064	SPI_SR	1	RX_FULL	R	Rx FIFO full
	SPI_SR	2	TX_EMPTY	R	Tx FIFO empty
	SPI_SR	3	TX_FULL	R	Tx FIFO full
0x1300_0068	SPI_DTR	7:0	DATA	RW	Data byte
0x1300_006C	SPI_DRR	7:0	DATA	R	Data byte
0x1300_0070	SPI_SSR	0	VALUE	RW	Chip select value

Table 6: WI 2.3.5 - SPI Registers

GPIO Registers

Offset	Register	Bits	Name	Type	Description
0x1400_0000	GPIO_DIRECTION	31:0	OUTPUT	RW	0 = Input; 1 = Output
0x1400_0004	GPIO_INPUT	31:0	VALUE	R	Raw input status
0x1400_0008	GPIO_OUTPUT	31:0	DATA	RW	GPIO output value
0x1400_00	GPIO_OUTPUT_	31:0	DATA	W	GPIO output mask - set

0C	SET				for high
0x1400_0010	GPIO_OUTPUT_CLR	31:0	DATA	W	GPIO output mask - set for low
0x1400_0014	GPIO_INT_MASK	31:0	ENABLE	RW	GPIO Interrupt Enable Mask
0x1400_0018	GPIO_INT_SET	31:0	SW_IRQ	W	Write 1 to assert an interrupt
0x1400_001C	GPIO_INT_CLR	31:0	ACK	W	Write 1 to clear an interrupt enable
0x1400_0020	GPIO_INT_STATUS	31:0	RAW	R	Set if interrupt active (regardless of INT_MASK)
0x1400_0024	GPIO_INT_LEVEL	31:0	ACTIVE_HIGH	RW	GPIO Interrupt Level : 1 = active high / rising edge; 0 = active low / falling edge
0x1400_0028	GPIO_INT_MODE	31:0	EDGE	RW	GPIO Interrupt Mode : 1 = edge / 0 = level

Table 7: WI 2.3.5 - GPIO Registers

3.3.2 Riscv_core registers

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 8: WI 2.3.5 - Riscv_core registers

3.4 Functional Description

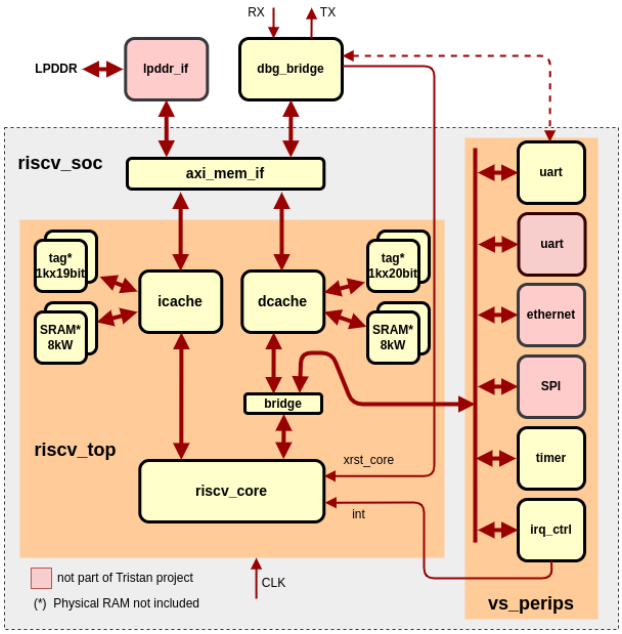

Figure 2: WI 2.3.5 - Architecture of the VSRV1 processor

Top hierarchy VSRV1 is divided as follows:

```
VSRV1                          (configuration/parameters file)
|--dbg_bridge                   : u_dbg
|  |--dbg_bridge_uart           : u_dbg/u_uart
|  |--dbg_bridge_fifo           : u_dbg/u_fifo_tx
|  |--dbg_bridge_fifo           : u_dbg/u_fifo_rx
|--riscv_soc                    : u_soc
   |--riscv_top                 : u_soc/u_core
   |  |--riscv_core             : u_soc/u_core/u_core
   |  |--icache                 : u_soc/u_core/u_icache
   |  |--dcache                 : u_soc/u_core/u_dcache
   |--vs_perips                 : u_soc/u_vs_perips
   |--axi_mem_if                : u_soc/u_axi_mem_if
      |--axi4_lite_tap          : u_soc/u_axi_mem_if/u_axi_tap
      |--axi4_arb               : u_soc/u_axi_mem_if/u_arb
      |  |--axi4_arb_onehot4    : u_soc/...../u_arb/u_rd_arb
      |  |--axi4_arb_onehot4    : u_soc/...../u_arb/u_wr_arb
      |--axi4_retime            : u_soc/.../u_retime
         |--axi4retime_fifo2x46 : u_soc/...../u_write_cmd_req
         |--axi4retime_fifo2x37 : u_soc/...../u_write_data_req
         |--axi4retime_fifo2x6  : u_soc/...../u_write_resp
         |--axi4retime_fifo2x46 : u_soc/...../u_read_req
         |--axi4retime_fifo2x39 : u_soc/...../u_read_resp
```

Figure 3: WI 2.3.5 - Top hierarchy of the VSRV1 processor

Referring to Figures 2 and 3, VSRV1 has two submodules: riscv_soc and dbg_bridge. The latter is a special UART that can be used to load the booth image and release the processor from the reset.

The riscv_soc submodule consists of three blocks:

External memory interface is implemented in axi_mem_if. It is a bridge between cache rams / debug UART and external memory. The memory can be any kind, but it must have an AXI4 interface.

Peripherals of the system are located in vs_perips submodule. At least interrupt controller (irq_ctrl) and UART are required when running Linux operating system. There can be also many other peripheral blocks such as SPI or ethernet. The bridge to the data bus is very simple. It is like a multiplexer but with some additional control.

The **processor core** (riscv_core) and Linux OS required memory management control for instructions (icache) and data (dcache) are in the riscv_top submodule.

```
                          (configuration/parameters file)
riscv_core                 :  /u_core
|--riscv_decode            :  /u_core/u_decode
|   |--riscv_decoder       :  /u_core/u_decode/u_dec
|--riscv_exec              :  /u_core/u_exec
|   |--riscv_alu           :  /u_core/u_exec/u_alu
|--riscv_multiplier        :  /u_core/u_mul
|--riscv_divider           :  /u_core/u_div
|--riscv_mmu               :  /u_core/u_mmu
|--riscv_lsu               :  /u_core/u_lsu
|-- |-riscv_lsu_fifo       :  /u_core/u_lsu/u_lsu_request
|--riscv_csr               :  /u_core/u_csr
|   |--riscv_csr_regfile   :  /u_core/u_csr/u_csrfile
|--riscv_issue             :  /u_core/u_issue
|   |--riscv_pipe_ctrl     :  /u_core/u_issue/u_pipe_ctrl
|   |--riscv_regfile       :  /u_core/u_issue/u_regfile
|--riscv_fetch             :  /u_core/u_fetch


icache (2x8kB)             :  /u_icache      (2x32x8912)
|--ram_itag                :  /u_icache/u_tag0   (18x1KW)
|--ram_itag                :  /u_icache/u_tag1   (18x1KW)
|--ram_idata               :  /u_icache/u_data0  (32x8KW)
|--ram_idata               :  /u_icache/u_data1  (32x8KW)


dcache                     : /u_dcache         (2x32x8912)
|--dcache_core             : /u_dcache/u_core
|  |--ram_dtag             : /u_dcache/u_core/u_tag0  (19x1KW)
|  |--ram_dtag             : /u_dcache/u_core/u_tag1  (19x1KW)
|  |--ram_ddata            : /u_dcache/u_core/u_data0 (8KW)
|  |--ram_ddata            : /u_dcache/u_core/u_data1 (8KW)
|--dcache_axi  (ID 4)      : /u_dcache/u_axi
   |--dcache_axi_fifo      : /u_dcache/u_axi/u_req
   |                           fifo size: 2 x (32+32+8+4+1)
   |--dcache_axi_axi       : /u_dcache/u_axi/u_axi
```

Figure 4: WI 2.3.5 - Hierarchy of the riscv_top submodule

3.4.1 System boot

A debug UART module is included to the design to upload the software to the main memory (LPDDR). There is no boot rom that would automatically initialize the system now. When system is in reset state the debug UART is enabled, and the riscv_core and caches stay in reset. Compiled elf file will be uploaded to the LPDDR via AXI4 bus interface and then a register controlling the reset of the riscv_core is cleared. This also switches the core's peripheral UART to the same pins that the debug UART was using. At boot-up the caches first initialize (flush) tag-memories and then the core can request first instruction. Once the first line is fetched from the memory the execution begins.

3.4.2 Riscv_core

The riscv_core fetches the instructions and data from the memories and executes them according to the 32-bit ISA instruction set.

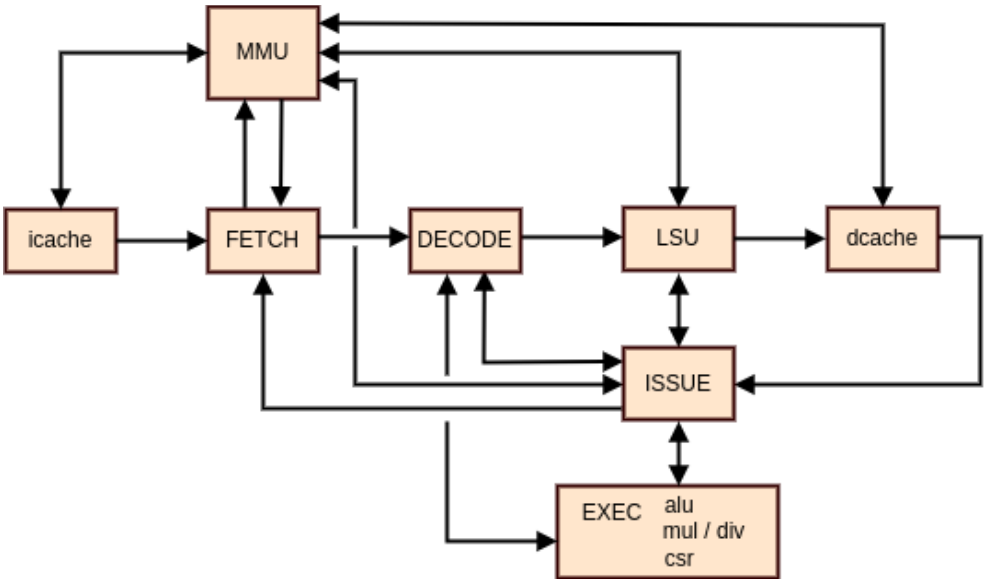

Figure 5: WI 2.3.5 - Architecture of the riscv_core submodule

3.4.3 Icache

Instruction cache submodule controls the access between the processor core and SRAM cache memories. It is a 32-bit set associative cache and has four parameters as VHDL generics:

Parameter	Type	Default value
AXI_ID	std_logic_vector	0x8
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

Table 9:WI 2.3.5 - Parameters as VHDL generics

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 are registered with clock edge). Memory sizes depend on the parameters and the default values make tag RAM as 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 type, the AXI4 write channel is omitted.

The detailed two-way cache implementation is shown in Figure 6: WI 2.3.5 - Two-way cache implementation below.

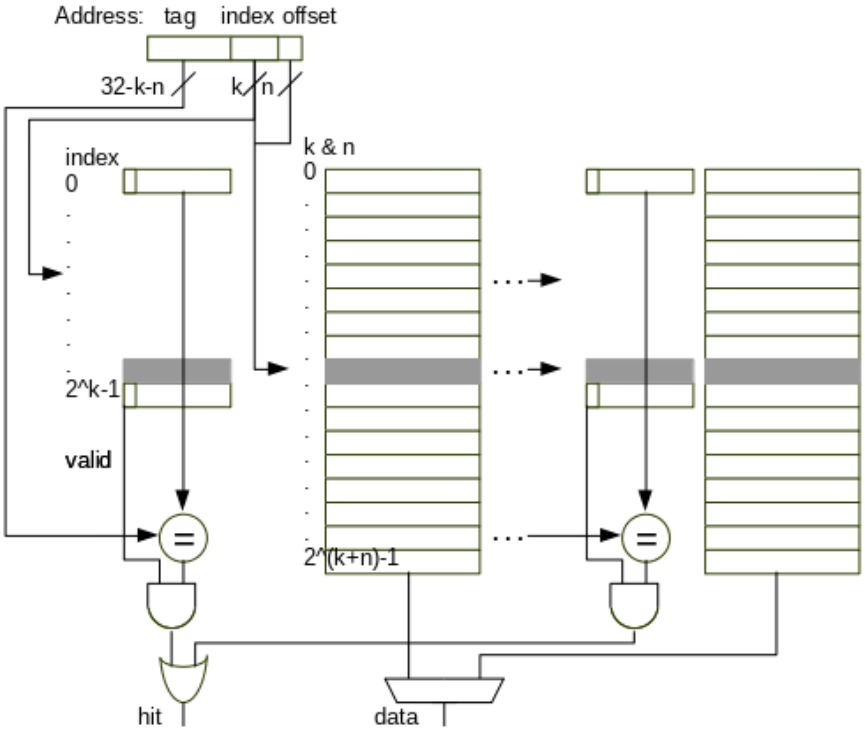

Figure 6: WI 2.3.5 - Two-way cache implementation

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 read to accept requests. The state diagram is shown below in Figure 7.

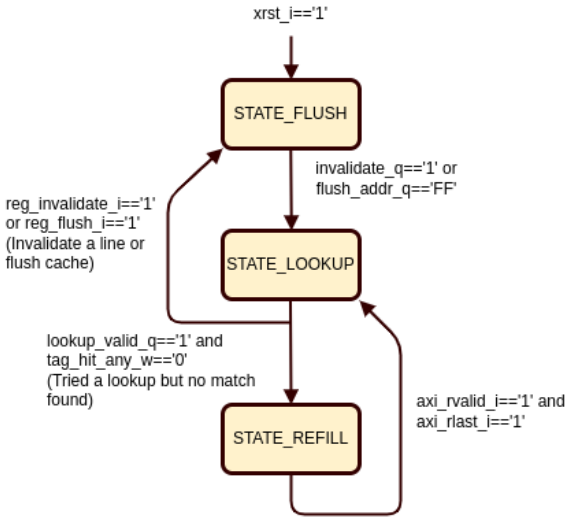

Figure 7: WI 2.3.5 - State diagram of icache

3.4.4 Dcache

Data cache is a set associative cache like icache and has 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 in Figure 8 below.

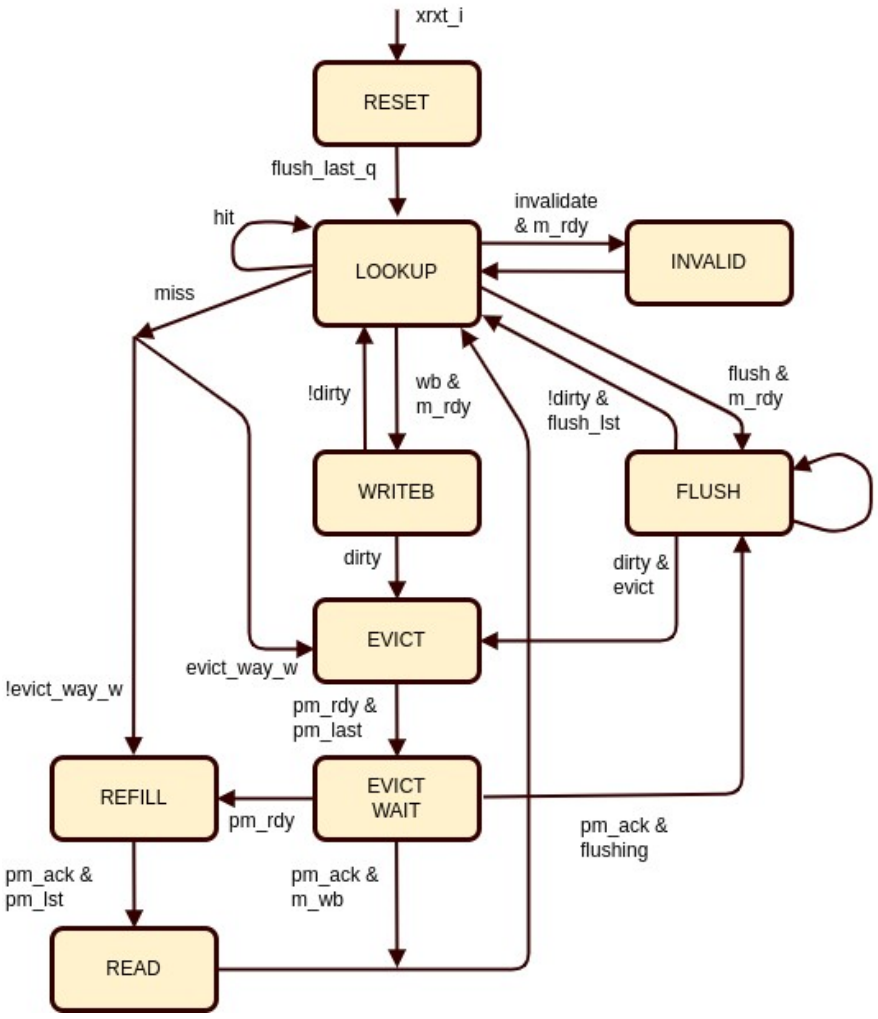

Figure 8: WI 2.3.5 - State diagram of dcache

3.4.5 Axi_mem_if

Icache and dcache interfaces use 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.

3.4.6 Vs_perips

For peripherals a very simple bus interface was 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.

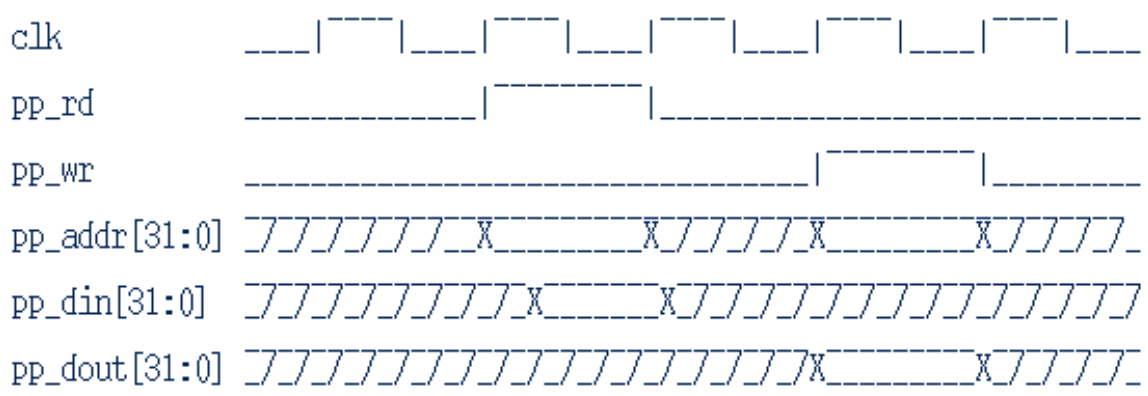

Figure 9: WI 2.3.5 - Timing diagram of vs_periph interface

The ricv_core block and data cache have small simple multiplexer kind of bridge to select the traffic alternatively to vs_perips submodule.

3.5 PPA Analysis

3.5.1 Gate count

Synthesis result of the risv_core block of VSRV1 is compared below with the synthesis result of core-v-wally core (both have similar architecture and configuration for synthesis)

Type	riscv_core (core + MMU of VSRV1)	Core-v-wally*
Number of registers	3471	5657
Number of nand equivalent logic gates	22,278	21,323

Table 10: WI 2.3.5 - Silicon area comparison of the VSRV1 core

* Modifications of the default Core-v-wally parameters are given below. Note that cache could not be made as big as it is in VSRV1 core.

Parameter	original default value	Updated value for synthesis
ZICBOM_SUPPORTED	0	1
ZICBOZ_SUPPORTED	0	1
ZICBOP_SUPPORTED	0	1

SVINVAL_SUPPORTED	0	1
SVADU_SUPPORTED	0	1
DCACHE_SUPPORTED	0	1
ICACHE_SUPPORTED	0	1
VIRTMEM_SUPPORTED	0	1
ITLB_ENTRIES	0	1
DTLB_ENTRIES	32'd0	32'd2
DCACHE_NUMWAYS	32'd4	32'd2
ICACHE_NUMWAYS	32'd4	32'd2
DTIM_SUPPORTED	1	0
IROM_SUPPORTED	1	0

Table 11: WI 2.3.5 - Modified parameters of Core-v-wally core for gate count comparison

3.5.2 Architecture development steps based on ExactStep simulator**

Linux boot time [s]	MEM	INT	FP	NOTE
250.413	0.027	0.038	0.042	Starting point: First Linux running version: 8KiB ICache, No DCache
76.921	0.095	0.156	0.157	4 KiB DCache
58.587	0.131	0.198	0.179	8 KiB DCache
42.321	0.162	0.229	0.220	25 MHz DDR
34.863	0.213	0.284	0.260	Added MMU
23.545	0.277	0.329	0.313	16+16 KiB I+D Cache
22.127	0.303	0.368	0.371	32+32 KiB I+D Cache
16.887	0.326	0.380	0.389	50 MHz LPDDR (=FPGA proto)
7.844	0.701	0.786	0.814	100 MHz CPU + LPDDR (=VSRV1 chip proto specification by using 110nm technology)

Table 12: WI 2.3.5 - Performance development improvement steps of the VSRV1 core.

*MEM, INT, FP: NBench Memory, Integer, and Floating-Point performance compared to ideal performance (1 operation per clock cycle) running at 100 MHz.

**ExactStep simulator original code was modified to include realistic model of the DDR memory, cache and pipeline control.

4 Verification and Validation

4.1 Verification Methodology

Since this project is in the research phase the verification concentrates on the functionality, performance and HW-SW co-operation verification. Blocks are first tested in functional test benches and then synthesized to FPGA where the focus is verification with the actual software. This also allows development of the software for the demo applications parallel with the hardware development and performance evaluations of the hardware together with the software. This way any hardware problem or performance issue can be addressed by the hardware and software developers together. The VSRV1 IP will be prototyped in a SoC chip in WI6.2.3, this will provide silicon proven IP with silicon proven performance. Target applications are commercial.

4.2 Testbench Architecture

The block level verification has the basic flow. The designer writes manually the input and the expected output and compares it with the simulated response. This was done in our case on VHDL level.

Additionally, we have used LEC to verify that our VHDL conversion from Verilog was correct.

4.3 Prototyping Architecture

The prototyping is mainly based on FPGA at the moment. In addition, we will develop a test chip in WI6.2.3. The test chip will be evaluated by using an evaluation board as well as by using loadboard and characterization program of a commercial ATE IC-tester.

4.4 Test Results

The FPGA based evaluation board boots Linux and works as expected in all tests we have done under Linux.

5 Tools

Tool	Purpose	Type
ExactStep	System level simulation and optimization	Open source
VHDL/Verilog simulator	RTL simulation	Commercial

Synthesis	Synthesis of VHDL to FPGA or std cell library	Commercial
Logic Equivalence Checker	Verify that conversion from Verilog to VHDL is equivalent with the original functionality	Commercial
Emacs	editor	Open source

Table 13: WI 2.3.5 - Tools

6 References

Requirements Specifications	WI2.3.5.xlsx
Architecture description and design specifications	TRISTAN_D2.1 Architecture Description and Design Specification_REPORT_Consortium Confidential v1.0.docx
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**
AMBA AXI Protocol Specification	https://developer.arm.com/documentation/ihi0022/latest/
QEMU simulator	https://www.qemu.org/
Spike simulator	https://github.com/riscv-software-src/riscv-isa-sim
Core-v-wally core	https://github.com/openhwgroup/cvw
Nbench test	https://github.com/petabridge/NBench