CV32E40X User Manual

OpenHW Group

Oct 13, 2022

CONTENTS:

1 Changelog **1**
- 1.1 0.6.0 . . . 1
- 1.2 0.5.0 . . . 1
- 1.3 0.4.0 . . . 1
- 1.4 0.3.0 . . . 1
- 1.5 0.2.0 . . . 1
- 1.6 0.1.0 . . . 1

2 Introduction **3**
- 2.1 License . . . 3
- 2.2 Standards Compliance . . . 4
- 2.3 Synthesis guidelines . . . 6
  - 2.3.1 ASIC Synthesis . . . 6
  - 2.3.2 FPGA Synthesis . . . 6
- 2.4 Verification . . . 7
- 2.5 Contents . . . 7
- 2.6 History . . . 7
- 2.7 References . . . 7
- 2.8 Contributors . . . 8

3 Getting Started with CV32E40X **9**
- 3.1 Clock Gating Cell . . . 9

4 Core Integration **11**
- 4.1 Instantiation Template . . . 11
- 4.2 Parameters . . . 13
- 4.3 Interfaces . . . 15

5 Pipeline Details **17**
- 5.1 Multi- and Single-Cycle Instructions . . . 17
- 5.2 Hazards . . . 19

6 Instruction Fetch **21**
- 6.1 Misaligned Accesses . . . 22
- 6.2 Protocol . . . 22

7 Load-Store-Unit (LSU) **25**
- 7.1 Misaligned Accesses . . . 26
- 7.2 Protocol . . . 26
- 7.3 Write buffer . . . 26
- 7.4 Atomics . . . 31

8 Atomic instructions **33**
8.1 Load-Reserved/Store-Conditional Instructions . . . 33
8.2 Atomic Memory Operations . . . 33

9 Physical Memory Attribution (PMA) **35**
9.1 Address range . . . 35
9.2 Main memory vs I/O . . . 35
9.3 Bufferable and Cacheable . . . 36
9.4 Atomic operations . . . 36
9.5 Default attribution . . . 37

10 Register File **39**
10.1 General Purpose Register File . . . 39

11 eXtension Interface **41**
11.1 CORE-V-XIF . . . 41
11.1.1 Compressed interface . . . 41
11.1.2 Issue interface . . . 42
11.1.3 Commit interface . . . 42
11.1.4 Memory (request/response) interface . . . 43
11.1.5 Memory result interface . . . 43
11.1.6 Result interface . . . 43
11.2 Integration . . . 44
11.3 Timing . . . 44

12 Fence.i external handshake **47**

13 Sleep Unit **49**
13.1 Startup behavior . . . 49
13.2 WFI . . . 50
13.3 WFE . . . 50

14 Control and Status Registers **51**
14.1 CSR Map . . . 51
14.2 CSR Descriptions . . . 53
14.2.1 Jump Vector Table (`jvt`) . . . 54
14.2.2 Machine Status (`mstatus`) . . . 54
14.2.3 Machine ISA (`misa`) . . . 55
14.2.4 Machine Interrupt Enable Register (`mie`) - `SMCLIC` == 0 . . . 57
14.2.5 Machine Interrupt Enable Register (`mie`) - `SMCLIC` == 1 . . . 57
14.2.6 Machine Trap-Vector Base Address (`mtvec`) - `SMCLIC` == 0 . . . 57
14.2.7 Machine Trap-Vector Base Address (`mtvec`) - `SMCLIC` == 1 . . . 58
14.2.8 Machine Trap Vector Table Base Address (`mtvt`) . . . 58
14.2.9 Machine Status (`mstatush`) . . . 59
14.2.10 Machine Counter-Inhibit Register (`mcountinhibit`) . . . 59
14.2.11 Machine Performance Monitoring Event Selector (`mhpmevent3 .. mhpmevent31`) . . . 60
14.2.12 Machine Scratch (`mscratch`) . . . 60
14.2.13 Machine Exception PC (`mepc`) . . . 60
14.2.14 Machine Cause (`mcause`) - `SMCLIC` == 0 . . . 60
14.2.15 Machine Cause (`mcause`) - `SMCLIC` == 1 . . . 61
14.2.16 Machine Trap Value (`mtval`) . . . 61
14.2.17 Machine Interrupt Pending Register (`mip`) - `SMCLIC` == 0 . . . 61
14.2.18 Machine Interrupt Pending Register (`mip`) - `SMCLIC` == 1 . . . 62
14.2.19 Machine Next Interrupt Handler Address and Interrupt Enable (`mnxti`) . . . 62
14.2.20 Machine Interrupt Status (`mintstatus`) . . . 63

14.2.21 Machine Interrupt-Level Threshold (mintthresh) . . . 63
14.2.22 Machine Scratch Swap for Priv Mode Change (mscratchcsw) . . . 63
14.2.23 Machine Scratch Swap for Interrupt-Level Change (mscratchcswl) . . . 64
14.2.24 CLIC Base (mclicbase) . . . 64
14.2.25 Trigger Select Register (tselect) . . . 65
14.2.26 Trigger Data 1 (tdata1) . . . 65
14.2.27 Match Control Type 6 (mcontrol6) . . . 65
14.2.28 Exception Trigger (etrigger) . . . 66
14.2.29 Trigger Data 1 (tdata1) - disabled view . . . 67
14.2.30 Trigger Data Register 2 (tdata2) . . . 67
14.2.31 Trigger Data Register 3 (tdata3) . . . 67
14.2.32 Trigger Info (tinfo) . . . 67
14.2.33 Trigger Control (tcontrol) . . . 68
14.2.34 Debug Control and Status (dcsr) . . . 68
14.2.35 Debug PC (dpc) . . . 69
14.2.36 Debug Scratch Register 0/1 (dscratch0/1) . . . 69
14.2.37 Machine Cycle Counter (mcycle) . . . 69
14.2.38 Machine Instructions-Retired Counter (minstret) . . . 69
14.2.39 Machine Performance Monitoring Counter (mhpmcounter3 .. mhpmcounter31) . . . 70
14.2.40 Upper 32 Machine Cycle Counter (mcycleh) . . . 70
14.2.41 Upper 32 Machine Instructions-Retired Counter (minstreth) . . . 70
14.2.42 Upper 32 Machine Performance Monitoring Counter (mhpmcounter3h .. mhpmcounter31h) 70
14.2.43 Machine Vendor ID (mvendorid) . . . 71
14.2.44 Machine Architecture ID (marchid) . . . 71
14.2.45 Machine Implementation ID (mimpid) . . . 71
14.2.46 Hardware Thread ID (mhartid) . . . 72
14.2.47 Machine Configuration Pointer (mconfigptr) . . . 72
14.2.48 Cycle Counter (cycle) . . . 72
14.2.49 Time (time) . . . 72
14.2.50 Instructions-Retired Counter (instret) . . . 73
14.2.51 Performance Monitoring Counter (hpmcounter3 .. hpmcounter31) . . . 73
14.2.52 Upper 32 Cycle Counter (cycleh) . . . 73
14.2.53 Upper 32 Time (timeh) . . . 73
14.2.54 Upper 32 Instructions-Retired Counter (instreth) . . . 74
14.2.55 Upper 32 Performance Monitoring Counter (hpmcounter3h .. hpmcounter31h) . . . 74

15 Performance Counters **75**
15.1 Event Selector . . . 75
15.2 Controlling the counters from software . . . 76
15.3 Parametrization at synthesis time . . . 76

16 Exceptions and Interrupts **77**
16.1 Exceptions . . . 77
16.2 Non Maskable Interrupts . . . 78
16.3 CLINT Mode Interrupt Architecture . . . 79
16.3.1 Interrupt Interface . . . 79
16.3.2 Interrupts . . . 79
16.3.3 Nested Interrupt Handling . . . 81
16.4 CLIC Mode Interrupt Architecture . . . 81
16.4.1 Interrupt Interface . . . 81
16.4.2 Interrupts . . . 82
16.4.3 Nested Interrupt Handling . . . 82

17 Debug & Trigger **83**

17.1 Interface . . . 84
17.2 Core Debug Registers . . . 85
17.3 Debug state . . . 85
17.4 EBREAK Behavior . . . 88
17.4.1 Scenario 1 : Enter Exception . . . 88
17.4.2 Scenario 2 : Enter Debug Mode . . . 88
17.4.3 Scenario 3 : Exit Program Buffer & Restart Debug Code . . . 88

18 RISC-V Formal Interface **89**
18.1 New Additions . . . 89
18.2 Compatibility . . . 90
18.3 Trace output file . . . 94
18.4 Trace output format . . . 94

19 CORE-V Instruction Set Extensions **97**
19.1 Custom instructions . . . 97
19.2 Custom CSRs . . . 97

20 Core Versions and RTL Freeze Rules **99**
20.1 What happens after RTL Freeze? . . . 99
20.1.1 A bug is found . . . 99
20.1.2 RTL changes on non-verified yet parameters . . . 99
20.1.3 PPA optimizations and new features . . . 99
20.2 Released core versions . . . 100

21 Glossary **101**

Bibliography **103**

CHAPTER

ONE

CHANGELOG

1.1 0.6.0

Released on 2022-10-13 - GitHub

1.2 0.5.0

Released on 2022-08-26 - GitHub

1.3 0.4.0

Released on 2022-06-07 - GitHub

1.4 0.3.0

Released on 2022-03-29 - GitHub

1.5 0.2.0

Released on 2022-03-18 - GitHub

1.6 0.1.0

Released on 2022-02-16 - GitHub

CHAPTER

TWO

INTRODUCTION

CV32E40X is a 4-stage in-order 32-bit RISC-V processor core. Figure 2.1 shows a block diagram of the core.

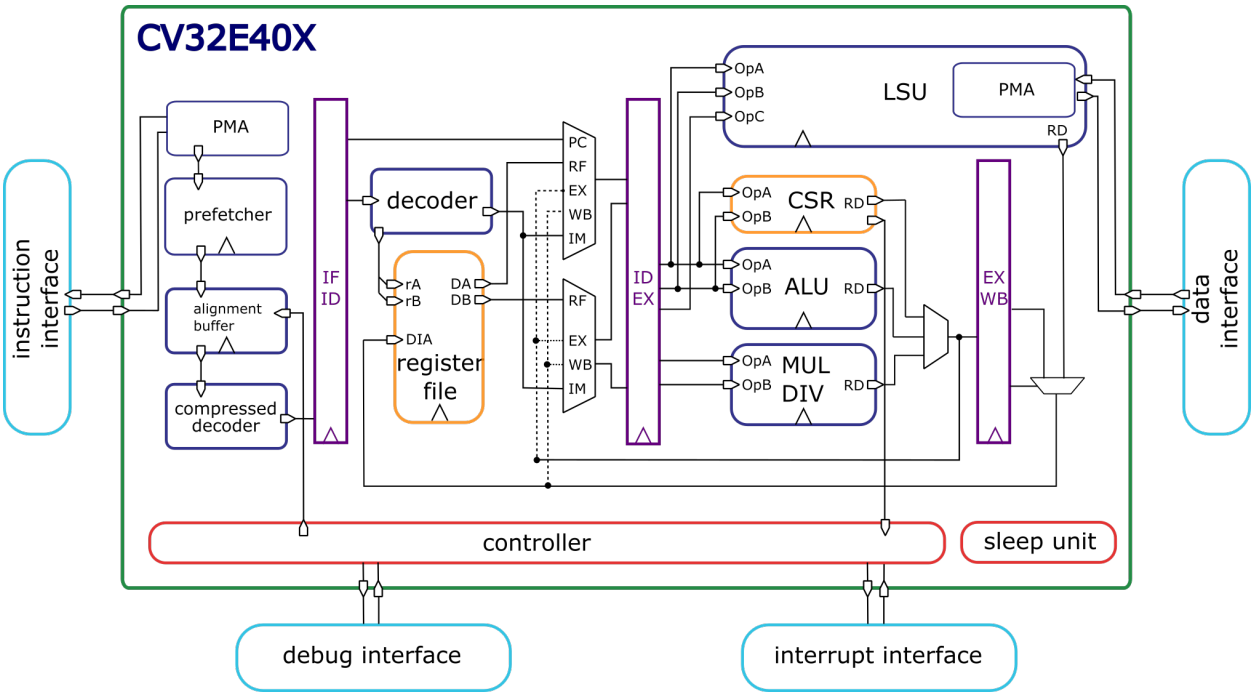


Figure 2.1: Block Diagram of CV32E40X RISC-V Core

2.1 License

Copyright 2020 OpenHW Group.

Copyright 2018 ETH Zurich and University of Bologna.

Copyright and related rights are licensed under the Solderpad Hardware License, Version 0.51 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://solderpad.org/licenses/SHL-0.51. Unless required by applicable law or agreed to in writing, software, hardware and materials distributed under this License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

2.2 Standards Compliance

CV32E40X is a standards-compliant 32-bit RISC-V processor. It follows these specifications:

Many features in the RISC-V specification are optional, and CV32E40X can be parameterized to enable or disable some of them.

CV32E40X supports one of the following base integer instruction sets from [RISC-V-UNPRIV].

Table 2.1: CV32E40X Base Instruction Set

Base Integer Instruction Set	Version	Configurability
RV32I: RV32I Base Integer Instruction Set	2.1	optionally enabled with the `RV32` parameter
RV32E: RV32E Base Integer Instruction Set	1.9 (not ratified yet)	optionally enabled with the `RV32` parameter

In addition, the following standard instruction set extensions are available from [RISC-V-UNPRIV], [RISC-V-ZBA_ZBB_ZBC_ZBS], [RISC-V-CRYPTO] and [RISC-V-ZCA_ZCB_ZCMP_ZCMT].

Table 2.2: CV32E40X Standard Instruction Set Extensions

Standard Extension	Version	Configurability
C: Standard Extension for Compressed Instructions	2.0	always enabled
M: Standard Extension for Integer Multiplication and Division	2.0	optionally enabled with the `M_EXT` parameter
Zicntr: Standard Extension for Base Counters and Timers	2.0	always enabled
Zihpm: Standard Extension for Hardware Performance Counters	2.0	always enabled
Zicsr: Control and Status Register Instructions	2.0	always enabled
Zifencei: Instruction-Fetch Fence	2.0	always enabled
Zca: Subset of the standard **Zc** Code-Size Reduction extension consisting of a subset of **C** with the FP load/stores removed.	v1.0.0-RC5.6 (not ratified yet; version will change)	always enabled
Zcb: Subset of the standard **Zc** Code-Size Reduction extension consisting of simple operations.	v1.0.0-RC5.6 (not ratified yet; version will change)	always enabled
Zcmp: Subset of the standard **Zc** Code-Size Reduction extension consisting of push/pop and double move which overlap with **c.fsdsp**.	v1.0.0-RC5.6 (not ratified yet; version will change)	always enabled
Zcmt: Subset of the standard **Zc** Code-Size Reduction extension consisting of table jump.	v1.0.0-RC5.6 (not ratified yet; version will change)	always enabled
A: Atomic Instructions	2.1	optionally enabled with the `A_EXT` parameter
Zba: Bit Manipulation Address calculation instructions	Version 1.0.0	optionally enabled with the `B_EXT` parameter
Zbb: Bit Manipulation Base instructions	Version 1.0.0	optionally enabled with the `B_EXT` parameter
Zbc: Bit Manipulation Carry-Less Multiply instructions	Version 1.0.0	optionally enabled with the `B_EXT` parameter
Zbs: Bit Manipulation Bit set, Bit clear, etc. instructions	Version 1.0.0	optionally enabled with the `B_EXT` parameter
Zkt: Data Independent Execution Latency	Version 1.0.0	always enabled
Zbkc: Constant time Carry-Less Multiply	Version 1.0.0	optionally enabled with the `B_EXT` parameter
Zmmul: Multiplication subset of the **M** extension	Version 0.1	optionally enabled with the `M_EXT` parameter

The following custom instruction set extensions are available.

Table 2.3: CV32E40X Custom Instruction Set Extensions

Custom Extension	Version	Configurability
Xif: eXtension Interface	0.1 (not finalized yet; version will change)	optionally enabled with the `X_EXT` parameter

Note: CV32E40X does not implement the **F** extension for single-precision floating-point instructions internal to the core. The **F** extension can be supported by interfacing the CV32E40X to an external FPU via the eXtension interface.

Most content of the RISC-V privileged specification is optional. CV32E40X supports the following features according to the RISC-V Privileged Specification [RISC-V-PRIV]:

- M-Mode
- All CSRs listed in *Control and Status Registers*
- Base Counters, Timers and Hardware Performance Counters as described in *Performance Counters* controlled by the `NUM_MHPMCOUNTERS` parameter
- Trap handling supporting direct mode or vectored mode as described at *Exceptions and Interrupts*
- Physical Memory Attribution (PMA) as described in *Physical Memory Attribution (PMA)*

CV32E40X supports the following ISA extensions from the RISC-V Debug Support specification [RISC-V-DEBUG]:

- **Sdext**: External Debug support. Always enabled.
- **Sdtrig**: Trigger Module. Optionally enabled with the `DBG_NUM_TRIGGERS` parameter.

2.3 Synthesis guidelines

The CV32E40X core is fully synthesizable. It has been designed mainly for ASIC designs, but FPGA synthesis is supported as well.

All the files in the `rtl` and `rtl/include` folders are synthesizable. The top level module is called `cv32e40x_core`.

The user must provide a clock-gating module that instantiates the clock-gating cells of the target technology. This file must have the same interface and module name of the one provided for simulation-only purposes at `bhv/cv32e40x_sim_clock_gate.sv` (see *Clock Gating Cell*).

The `constraints/cv32e40x_core.sdc` file provides an example of synthesis constraints. No synthesis scripts are provided.

2.3.1 ASIC Synthesis

ASIC synthesis is supported for CV32E40X. The whole design is completely synchronous and uses positive-edge triggered flip-flops. A technology specific implementation of a clock gating cell as described in *Clock Gating Cell* needs to be provided.

2.3.2 FPGA Synthesis

FPGA synthesis is supported for CV32E40X. The user needs to provide a technology specific implementation of a clock gating cell as described in *Clock Gating Cell*.

2.4 Verification

The verification environment (testbenches, testcases, etc.) for the CV32E40X core can be found at core-v-verif. It is recommended that you start by reviewing the CORE-V Verification Strategy.

2.5 Contents

- *Getting Started with CV32E40X* discusses the requirements and initial steps to start using CV32E40X.
- *Core Integration* provides the instantiation template and gives descriptions of the design parameters as well as the input and output ports.
- *CV32E40X Pipeline* described the overal pipeline structure.
- The instruction and data interfaces of CV32E40X are explained in *Instruction Fetch* and *Load-Store-Unit (LSU)*, respectively.
- *Physical Memory Attribution (PMA)* describes the Physical Memory Attribution (PMA) unit.
- The register-file is described in *Register File*.
- *eXtension Interface* describes the custom eXtension interface.
- *Sleep Unit* describes the Sleep unit.
- The control and status registers are explained in *Control and Status Registers*.
- *Performance Counters* gives an overview of the performance monitors and event counters available in CV32E40X.
- *Exceptions and Interrupts* deals with the infrastructure for handling exceptions and interrupts.
- *Debug & Trigger* gives a brief overview on the debug infrastructure.
- *RISC-V Formal Interface* gives a brief overview of the RVFI module.
- *Glossary* provides definitions of used terminology.

2.6 History

CV32E40X started its life as a fork of the CV32E40P from the OpenHW Group <https://www.openhwgroup.org>.

2.7 References

1. Gautschi, Michael, et al. "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices." in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2700-2713, Oct. 2017
2. Schiavone, Pasquale Davide, et al. "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications." 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS 2017)

2.8 Contributors

Andreas Traber (*atraber@iis.ee.ethz.ch*)

Michael Gautschi (*gautschi@iis.ee.ethz.ch*)

Pasquale Davide Schiavone (*pschiavo@iis.ee.ethz.ch*)

Arjan Bink (*arjan.bink@silabs.com*)

Paul Zavalney (*paul.zavalney@silabs.com*)

Micrel Lab and Multitherman Lab
University of Bologna, Italy

Integrated Systems Lab
ETH Zürich, Switzerland

CHAPTER

THREE

GETTING STARTED WITH CV32E40X

This page discusses initial steps and requirements to start using CV32E40X in your design.

3.1 Clock Gating Cell

CV32E40X requires clock gating cells. These cells are usually specific to the selected target technology and thus not provided as part of the RTL design. A simulation-only version of the clock gating cell is provided in `cv32e40x_sim_clock_gate.sv`. This file contains a module called `cv32e40x_clock_gate` that has the following ports:

- `clk_i`: Clock Input
- `en_i`: Clock Enable Input
- `scan_cg_en_i`: Scan Clock Gate Enable Input (activates the clock even though `en_i` is not set)
- `clk_o`: Gated Clock Output

And the following Parameters: * `LIB` : Standard cell library (semantics defined by integrator)

Inside CV32E40X, the clock gating cell is used in `cv32e40x_sleep_unit.sv`.

The `cv32e40x_sim_clock_gate.sv` file is not intended for synthesis. For ASIC synthesis and FPGA synthesis the manifest should be adapted to use a customer specific file that implements the `cv32e40x_clock_gate` module using design primitives that are appropriate for the intended synthesis target technology.

CHAPTER

FOUR

CORE INTEGRATION

The main module is named `cv32e40x_core` and can be found in `cv32e40x_core.sv`. Below, the instantiation template is given and the parameters and interfaces are described.

4.1 Instantiation Template

```
cv32e40x_core #(
    .LIB                        (         0 ),
    .RV32                       (     RV32I ),
    .A_EXT                      (         0 ),
    .B_EXT                      (    B_NONE ),
    .M_EXT                      (         M ),
    .X_EXT                      (         0 ),
    .X_NUM_RS                   (         2 ),
    .X_ID_WIDTH                 (         4 ),
    .X_MEM_WIDTH                (        32 ),
    .X_RFR_WIDTH                (        32 ),
    .X_RFW_WIDTH                (        32 ),
    .X_MISA                     (     32'h0 ),
    .X_ECS_XS                   (      2'b0 ),
    .DBG_NUM_TRIGGERS           (         1 ),
    .NUM_MHPMCOUNTERS           (         1 ),
    .PMA_NUM_REGIONS            (         1 ),
    .PMA_CFG                    ( PMA_CFG[] ),
    .SMCLIC                     (         0 ),
    .SMCLIC_ID_WIDTH            (         5 ),
    .SMCLIC_INTTHRESHBITS       (         8 )
) u_core (
    // Clock and reset
    .clk_i                  (),
    .rst_ni                 (),
    .scan_cg_en_i           (),

    // Configuration
    .boot_addr_i            (),
    .mtvec_addr_i           (),
    .dm_halt_addr_i         (),
    .dm_exception_addr_i    (),
    .mhartid_i              (),
```

(continues on next page)

(continued from previous page)

```
    .mimpid_patch_i          (),

    // Instruction memory interface
    .instr_req_o             (),
    .instr_gnt_i             (),
    .instr_addr_o            (),
    .instr_memtype_o         (),
    .instr_prot_o            (),
    .instr_dbg_o             (),
    .instr_rvalid_i          (),
    .instr_rdata_i           (),
    .instr_err_i             (),

    // Data memory interface
    .data_req_o              (),
    .data_gnt_i              (),
    .data_addr_o             (),
    .data_atop_o             (),
    .data_be_o               (),
    .data_memtype_o          (),
    .data_prot_o             (),
    .data_dbg_o              (),
    .data_wdata_o            (),
    .data_we_o               (),
    .data_rvalid_i           (),
    .data_rdata_i            (),
    .data_err_i              (),
    .data_exokay_i           (),

    // Cycle, Time
    .mcycle_o                (),
    .time_i                  (),

    // eXtension interface
    .xif_compressed_if       (),
    .xif_issue_if            (),
    .xif_commit_if           (),
    .xif_mem_if              (),
    .xif_mem_result_if       (),
    .xif_result_if           (),

     // Interrupt interface
    .irq_i                   (),

    .clic_irq_i              (),
    .clic_irq_id_i           (),
    .clic_irq_level_i        (),
    .clic_irq_priv_i         (),
    .clic_irq_shv_i          (),

    // Fencei flush handshake
    .fencei_flush_req_o      (),
```

(continues on next page)

(continued from previous page)

```
    .fencei_flush_ack_i       (),

    // Debug interface
    .debug_req_i              (),
    .debug_havereset_o        (),
    .debug_running_o          (),
    .debug_halted_o           (),
    .debug_pc_valid_o         (),
    .debug_pc_o               (),

    // Special control signals
    .fetch_enable_i           (),
    .core_sleep_o             (),
    .wu_wfe_i                 ()
);
```

4.2 Parameters

Note: All eXtension interface parameters (`X_NUM_RS`, `X_ID_WIDTH`, `X_MEM_WIDTH`, `X_RFR_WIDTH` and `X_RFW_WIDTH`) must be set with values matching the actual `if_xif` instance and the coprocessor/interconnect available outside of CV32E40X.

Name	Type/Range	Default	Description
LIB	int	0	Standard cell library (semantics defined by integrator)
RV32	rv32_e	RV32I	Base Integer Instruction Set. RV32 = RV32I: RV32I Base Integer Instruction Set. RV32 = RV32E: RV32E Base Integer Instruction Set.
A_EXT	bit	0	Enable Atomic Instruction (A) support (**not implemented yet**)
B_EXT	b_ext_e	B_NONE	Enable Bit Manipulation support. B_EXT = B_NONE: No Bit Manipulation instructions are supported. B_EXT = ZBA_ZBB: Zba and Zbb are supported. B_EXT = ZBA_ZBB_ZBS: Zba, Zbb and Zbs are supported. B_EXT = ZBA_ZBB_ZBC_ZBS: Zba, Zbb, Zbc and Zbs are supported.
M_EXT	m_ext_e	M	Enable Multiply / Divide support. M_EXT = M_NONE: No multiply / divide instructions are supported. M_EXT = ZMMUL: The multiplication subset of the M extension is supported. M_EXT = M: The M extension is supported.
X_EXT	bit	0	Enable eXtension Interface (X) support, see *eXtension Interface*
X_NUM_RS	int (2..3)	2	Number of register file read ports that can be used by the eXtension interface.
X_ID_WIDTH	int (3..32)	4	Identification width for the eXtension interface.
X_MEM_WIDTH	int (32 64, 128, 256)	32	Memory access width for loads/stores via the eXtension interface.
X_RFR_WIDTH	int (32, 64)	32	Register file read access width for the eXtension interface.
X_RFW_WIDTH	int (32, 64)	32	Register file write access width for the eXtension interface.
X_MISA	logic [31:0]	32'h0	MISA extensions implemented on the eXtension interface, see *Machine ISA (misa)*. X_MISA can only be used to set a subset of the following: {P, V, F, M}.
X_ECS_XS	logic [1:0]	2'b0	Default value for mstatus.XS if X_EXT = 1, see *Machine Status (mstatus)*.
NUM_MHPMCOUNTERS	int (0..29)	1	Number of MHPMCOUNTER performance counters, see *Performance Counters*
DBG_NUM_TRIGGERS	int (0..4)	1	Number of debug triggers, see *Debug & Trigger*
PMA_NUM_REGIONS	int (0..16)	0	Number of PMA regions
PMA_CFG[]	pma_cfg_t	PMA_R_DEFAULT	PMA configuration. Array of pma_cfg_t with PMA_NUM_REGIONS entries, see *Physical Memory Attribution (PMA)*
SMCLIC	int (0..1)	0	Is Smclic supported?
SMCLIC_ID_WIDTH	int (1..10)	5	Width of clic_irq_id_i and clic_irq_id_o. The maximum number of supported interrupts in CLIC mode is 2^SMCLIC_ID_WIDTH. Trap vector table alignment is restricted as described in *Machine Trap Vector Table Base Address (mtvt)*.
SMCLIC_INTTHRESHBITS	int (1..8)	8	Number of bits actually implemented in mintthresh.th field.

4.3 Interfaces

Signal(s)	Width	Dir	Description
clk_i	1	in	Clock signal
rst_ni	1	in	Active-low asynchronous reset
scan_cg_en_i	1	in	Scan clock gate enable. Design for test (DfT) related signal. Can be used during scan testing operation to force instantiated clock gate(s) to be enabled. This signal should be 0 during normal / functional operation.
boot_addr_i	32	in	Boot address. First program counter after reset = boot_addr_i. Must be word aligned. Do not change after enabling core via fetch_enable_i
mtvec_addr_i	32	in	mtvec address. Initial value for the address part of *Machine Trap-Vector Base Address (mtvec) - SMCLIC == 0*. Must be 128-byte aligned (i.e. mtvec_addr_i[6:0] = 0). Do not change after enabling core via fetch_enable_i
dm_halt_addr_i	32	in	Address to jump to when entering Debug Mode, see *Debug & Trigger*. Must be word aligned. Do not change after enabling core via fetch_enable_i
dm_exception_addr_i	32	in	Address to jump to when an exception occurs when executing code during Debug Mode, see *Debug & Trigger*. Must be word aligned. Do not change after enabling core via fetch_enable_i
mhartid_i	32	in	Hart ID, usually static, can be read from *Hardware Thread ID (mhartid)* CSR
mimpid_patch_i	4	in	Implementation ID patch. Must be static. Readable as part of *Machine Implementation ID (mimpid)* CSR.
instr_*	Instruction fetch interface, see *Instruction Fetch*		
data_*	Load-store unit interface, see *Load-Store-Unit (LSU)*		
mcycle_o	Cycle Counter Output		
time_i	Time input, see *Time (time)* CSR and *Upper 32 Time (timeh)* CSR		
irq_*	Interrupt inputs, see *Exceptions and Interrupts*		
clic_*_i	CLIC interface, see *Exceptions and Interrupts*		
debug_*	Debug interface, see *Debug & Trigger*		
fetch_enable_i	1	in	Enable the instruction fetch of CV32E40X. The first instruction fetch after reset de-assertion will not happen as long as this signal is 0. fetch_enable_i needs to be set to 1 for at least one cycle while not in reset to enable fetching. Once fetching has been enabled the value fetch_enable_i is ignored.
core_sleep_o	1	out	Core is sleeping, see *Sleep Unit*.
wu_wfe_i	1	in	Wake-up for wfe, see *Sleep Unit*.
xif_compressed_if	eXtension compressed interface, see *Compressed interface*		
xif_issue_if	eXtension issue interface, see *Issue interface*		
xif_commit_if	eXtension commit interface, see *Commit interface*		
xif_mem_if	eXtension memory interface, see *Memory (request/response) interface*		
xif_mem_result_if	eXtension memory result interface, see *Memory result interface*		
xif_result_if	eXtension result interface, see *Result interface*		

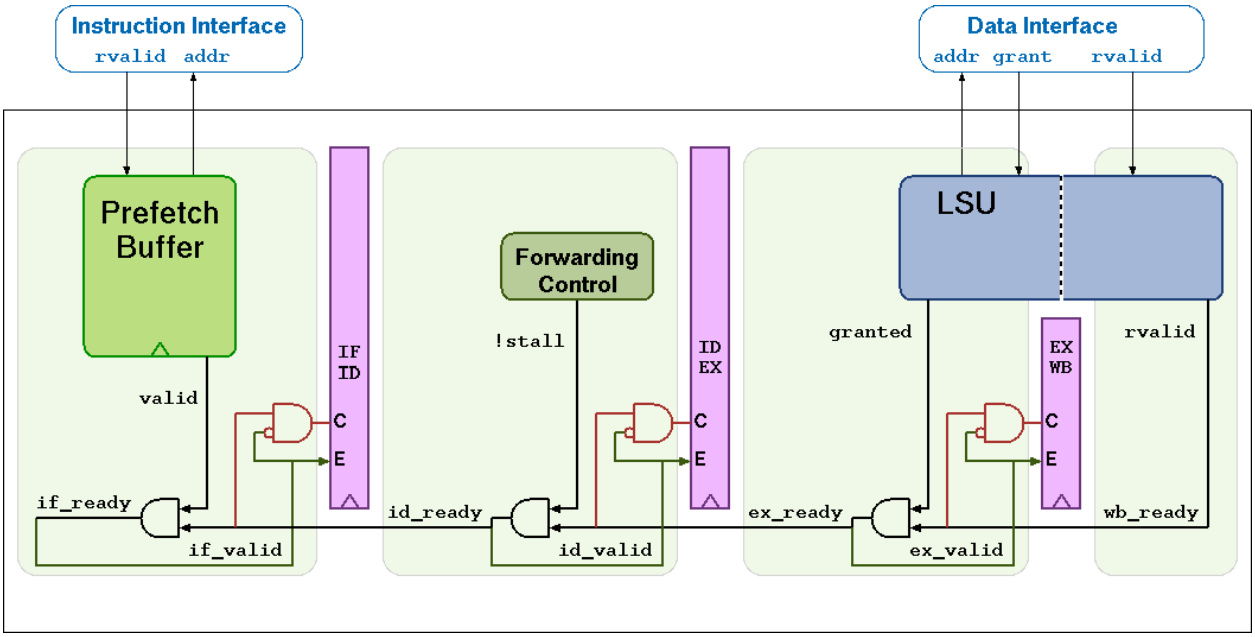


Figure 4.1: CV32E40X Pipeline

CHAPTER

FIVE

PIPELINE DETAILS

CV32E40X has a 4-stage in-order completion pipeline, the 4 stages are:

Instruction Fetch (IF)
: Fetches instructions from memory via an aligning prefetch buffer, capable of fetching 1 instruction per cycle if the instruction side memory system allows. The IF stage also pre-decodes RVC instructions into RV32I base instructions. See *Instruction Fetch* for details.

Instruction Decode (ID)
: Decodes fetched instruction and performs required register file reads. Jumps are taken from the ID stage.

Execute (EX)
: Executes the instructions. The EX stage contains the ALU, Multiplier and Divider. Branches (with their condition met) are taken from the EX stage. Multi-cycle instructions will stall this stage until they are complete. The address generation part of the load-store-unit (LSU) is contained in EX as well.

Writeback (WB)
: Writes the result of ALU, Multiplier, Divider, or Load instructions instructions back to the register file.

5.1 Multi- and Single-Cycle Instructions

Table 5.1 shows the cycle count per instruction type. Some instructions have a variable time, this is indicated as a range e.g. 1..32 means that the instruction takes a minimum of 1 cycle and a maximum of 32 cycles. The cycle counts assume zero stall on the instruction-side interface and zero stall on the data-side memory interface.

Table 5.1: Cycle counts per instruction type

Instruction Type	Cycles	Description
Integer Computational	1	Integer Computational Instructions are defined in the RISCV-V RV32I Base Integer Instruction Set.
CSR Access	4 (mstatus, mepc, mtvec, mcause, mcycle, minstret, mhpmcounter*, mcycleh, minstreth, mhpmcounter*h, mcountinhibit, mhpmevent*, dscr, dpc, dscratch0, dscratch1, privlv) 1 (all the other CSRs)	CSR Access Instruction are defined in 'Zicsr' of the RISC-V specification.
Load/Store	1 2 (non-word aligned word transfer) 2 (halfword transfer crossing word boundary)	Load/Store is handled in 1 bus transaction using both EX and WB stages for 1 cycle each. For misaligned word transfers and for halfword transfers that cross a word boundary 2 bus transactions are performed using EX and WB stages for 2 cycles each.
Multiplication	1 (mul) 4 (mulh, mulhsu, mulhu)	CV32E40X uses a single-cycle 32-bit x 32-bit multiplier with a 32-bit result. The multiplications with upper-word result take 4 cycles to compute.
Division Remainder	3 - 35 3 - 35	The number of cycles depends on the divider operand value (operand b), i.e. in the number of leading bits at 0. The minimum number of cycles is 3 when the divider has zero leading bits at 0 (e.g., 0x8000000). The maximum number of cycles is 35 when the divider is 0
Jump	2 3 (target is a non-word-	Jumps are performed in the ID stage. Upon a jump the IF stage (including prefetch buffer) is flushed. The new PC request will appear on the instruction-side memory interface the same cycle the jump instruction is in the ID stage.

5.2 Hazards

The CV32E40X experiences a 1 cycle penalty on the following hazards.

- Load data hazard (in case the instruction immediately following a load uses the result of that load)
- Jump register (jalr) data hazard (in case that a jalr depends on the result of an immediately preceding non-load instruction)

The CV32E40X experiences a 2 cycle penalty on the following hazards.

- Jump register (jalr) data hazard (in case that a jalr depends on the result of an immediately preceding load instruction)

CHAPTER

SIX

INSTRUCTION FETCH

The Instruction Fetch (IF) stage of the CV32E40X is able to supply one instruction to the Instruction Decode (ID) stage per cycle if the external bus interface is able to serve one instruction per cycle. In case of executing compressed instructions, on average less than one 32-bit instruction fetch will we needed per instruction in the ID stage.

For optimal performance and timing closure reasons, a prefetcher is used which fetches instructions via the external bus interface from for example an externally connected instruction memory or instruction cache.

The prefetch unit performs word-aligned 32-bit prefetches and stores the fetched words in an alignment buffer with three entries. As a result of this (speculative) prefetch, CV32E40X can fetch up to three words outside of the code region and care should therefore be taken that no unwanted read side effects occur for such prefetches outside of the actual code region.

Table 6.1 describes the signals that are used to fetch instructions. This interface is a simplified version of the interface that is used by the LSU, which is described in *Load-Store-Unit (LSU)*. The difference is that no writes are possible and thus it needs fewer signals.

Table 6.1: Instruction Fetch interface signals

Signal	Direction	Description
`instr_req_o`	output	Request valid, will stay high until `instr_gnt_i` is high for one cycle
`instr_gnt_i`	input	The other side accepted the request. `instr_addr_o`, `instr_memtype_o` and `instr_prot_o` may change in the next cycle.
`instr_addr_o[31:0]`	output	Address, word aligned
`instr_memtype_o[1:0]`	output	Memory Type attributes (cacheable, bufferable)
`instr_prot_o[2:0]`	output	Protection attributes
`instr_dbg_o`	output	Debug mode access
`instr_rvalid_i`	input	`instr_rdata_i` and `instr_err_i` are valid when `instr_rvalid_i` is high. This signal will be high for exactly one cycle per request.
`instr_rdata_i[31:0]`	input	Data read from memory
`instr_err_i`	input	An instruction interface error occurred

6.1 Misaligned Accesses

Externally, the IF interface performs word-aligned instruction fetches only. Misaligned instruction fetches are handled by performing two separate word-aligned instruction fetches. Internally, the core can deal with both word- and half-word-aligned instruction addresses to support compressed instructions. The LSB of the instruction address is ignored internally.

6.2 Protocol

The instruction bus interface is compliant to the OBI protocol (see [OPENHW-OBI] for detailed signal and protocol descriptions). The CV32E40X instruction fetch interface does not implement the following optional OBI signals: we, be, wdata, auser, wuser, aid, rready, ruser, rid. These signals can be thought of as being tied off as specified in the OBI specification. The CV32E40X instruction fetch interface can cause up to two outstanding transactions.

Figure 6.1 and Figure 6.3 show example timing diagrams of the protocol.

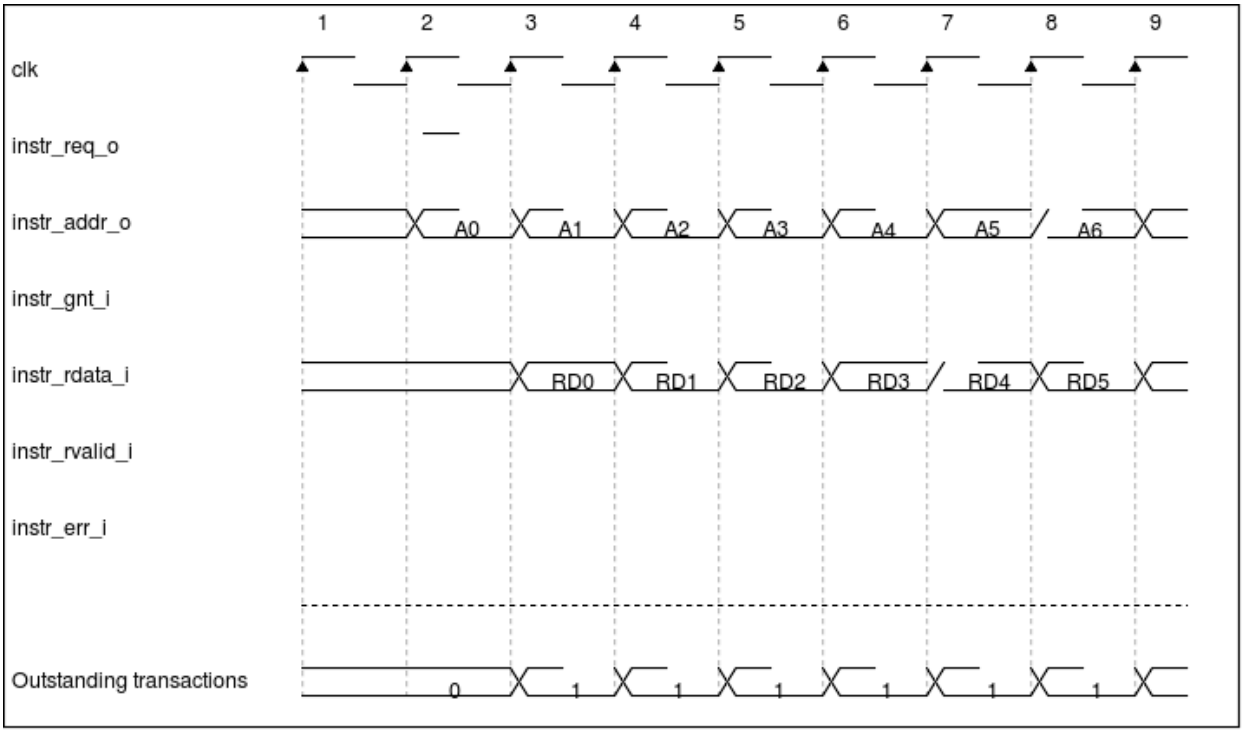


Figure 6.1: Back-to-back Memory Transactions

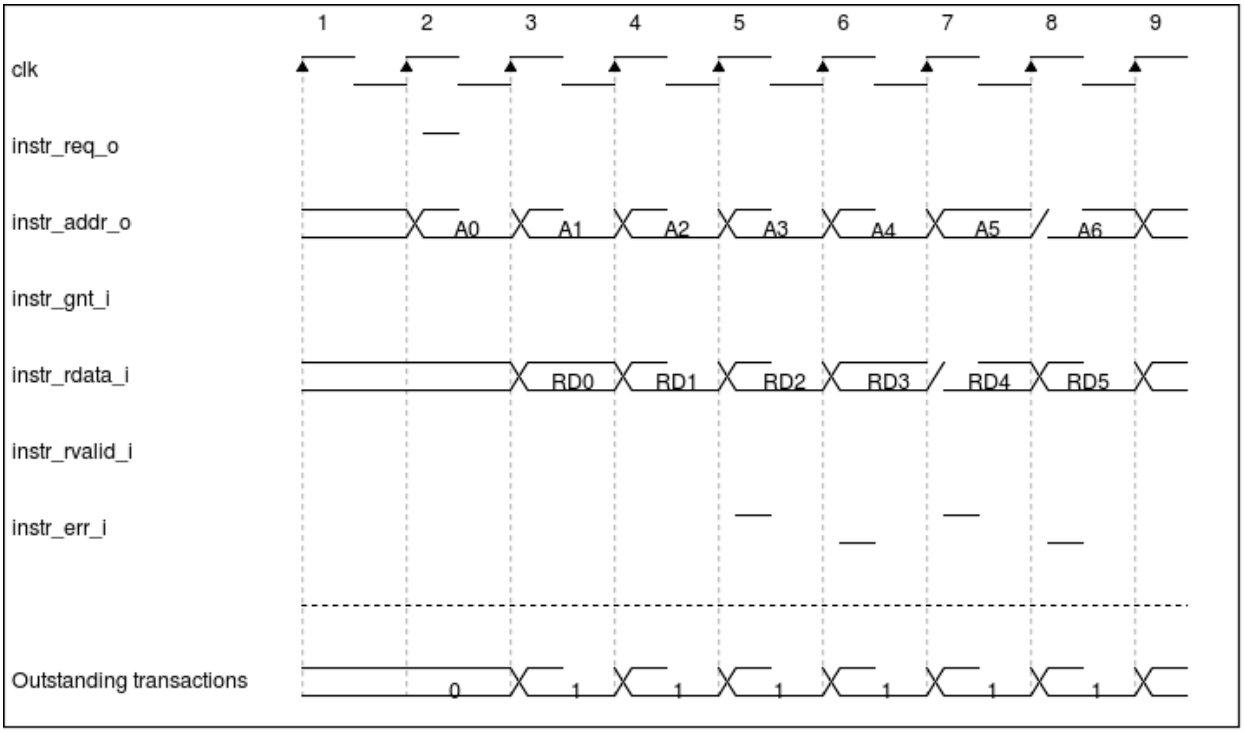


Figure 6.2: Back-to-back Memory Transactions with bus errors on A2/RD2 and A4/RD4

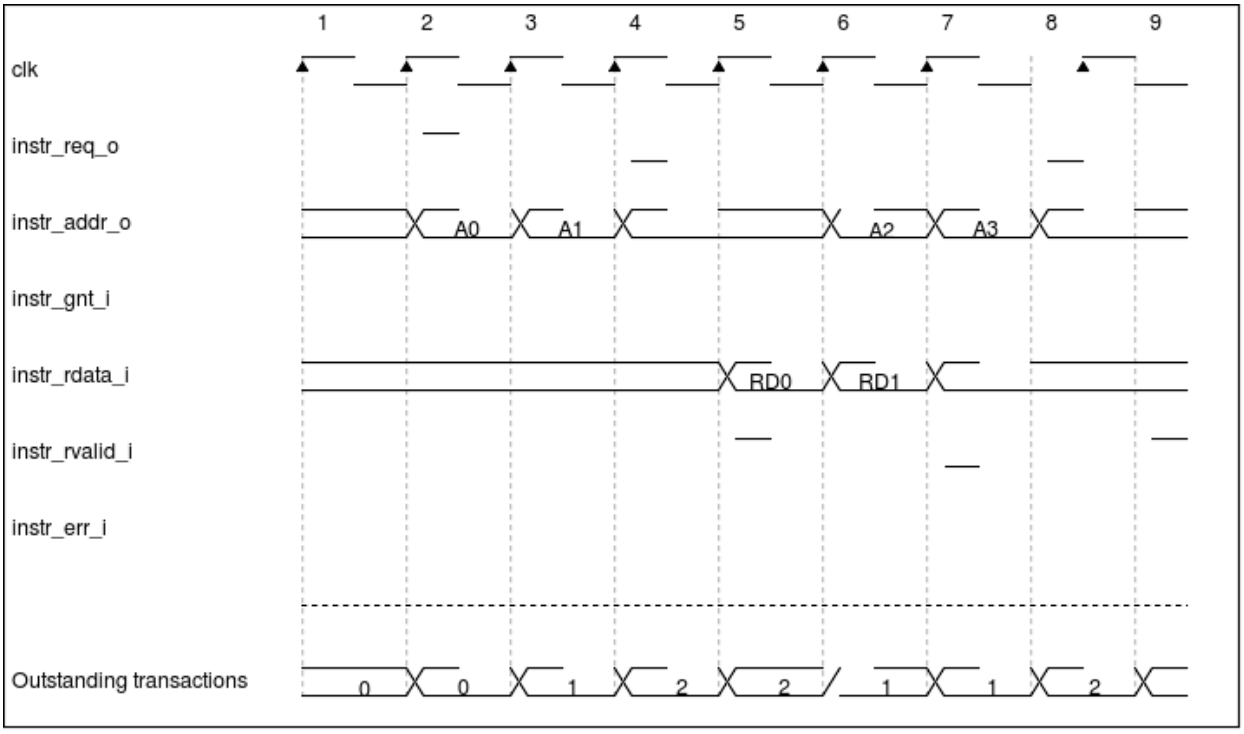


Figure 6.3: Multiple Outstanding Memory Transactions

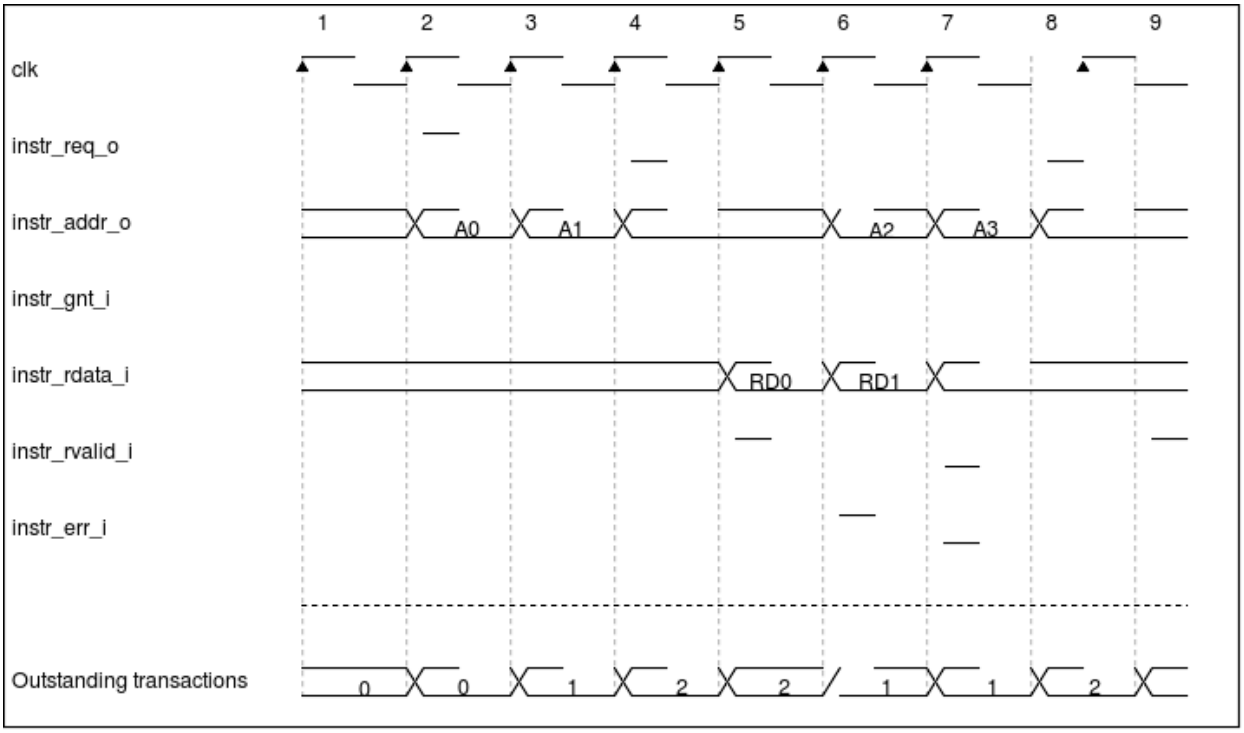


Figure 6.4: Multiple Outstanding Memory Transactions with bus error on A1/RD1

CHAPTER

SEVEN

LOAD-STORE-UNIT (LSU)

The Load-Store Unit (LSU) of the core takes care of accessing the data memory. Load and stores on words (32 bit), half words (16 bit) and bytes (8 bit) are supported.

Table 7.1 describes the signals that are used by the LSU.

Table 7.1: LSU interface signals

Signal	Direction	Description
data_req_o	output	Request valid, will stay high until data_gnt_i is high for one cycle
data_gnt_i	input	The other side accepted the request. data_addr_o, data_atop_o, data_be_o, data_memtype_o[2:0], data_prot_o, data_wdata_o, data_we_o may change in the next cycle.
data_addr_o[31:0]	output	Address, sent together with data_req_o.
data_atop_o[5:0]	output	Atomic attributes, sent together with data_req_o.
data_be_o[3:0]	output	Byte Enable. Is set for the bytes to write/read, sent together with data_req_o.
data_memtype_o[1:0]	output	Memory Type attributes (cacheable, bufferable), sent together with data_req_o.
data_prot_o[2:0]	output	Protection attributes, sent together with data_req_o.
data_dbg_o	output	Debug mode access, sent together with data_req_o.
data_wdata_o[31:0]	output	Data to be written to memory, sent together with data_req_o.
data_we_o	output	Write Enable, high for writes, low for reads. Sent together with data_req_o.
data_rvalid_i	input	data_rvalid_i will be high for exactly one cycle to signal the end of the response phase of for both read and write transactions. For a read transaction data_rdata_i holds valid data when data_rvalid_i is high.
data_rdata_i[31:0]	input	Data read from memory. Only valid when data_rvalid_i is high.
data_err_i	input	A data interface error occurred. Only valid when data_rvalid_i is high.
data_exokay_i	input	Exclusive transaction status. Only valid when data_rvalid_i is high.

7.1 Misaligned Accesses

Misaligned transaction are supported in hardware for Main memory regions, see *Physical Memory Attribution (PMA)*. For loads and stores in Main memory where the effective address is not naturally aligned to the referenced datatype (i.e., on a four-byte boundary for word accesses, and a two-byte boundary for halfword accesses) the load/store is performed as two bus transactions in case that the data item crosses a word boundary. A single load/store instruction is therefore performed as two bus transactions for the following scenarios:

- Load/store of a word for a non-word-aligned address
- Load/store of a halfword crossing a word address boundary

In both cases the transfer corresponding to the lowest address is performed first. All other scenarios can be handled with a single bus transaction.

Misaligned transactions are not supported in I/O regions and will result in an exception trap when attempted, see *Exceptions and Interrupts*.

7.2 Protocol

The data bus interface is compliant to the OBI protocol (see [OPENHW-OBI] for detailed signal and protocol descriptions). The CV32E40X data interface does not implement the following optional OBI signals: auser, wuser, aid, rready, ruser, rid. These signals can be thought of as being tied off as specified in the OBI specification. The CV32E40X data interface can cause up to two outstanding transactions.

The OBI protocol that is used by the LSU to communicate with a memory works as follows.

The LSU provides a valid address on `data_addr_o`, control information on `data_we_o`, `data_be_o` (as well as write data on `data_wdata_o` in case of a store) and sets `data_req_o` high. The memory sets `data_gnt_i` high as soon as it is ready to serve the request. This may happen at any time, even before the request was sent. After a request has been granted the address phase signals (`data_addr_o`, `data_we_o`, `data_be_o` and `data_wdata_o`) may be changed in the next cycle by the LSU as the memory is assumed to already have processed and stored that information. After granting a request, the memory answers with a `data_rvalid_i` set high if `data_rdata_i` is valid. This may happen one or more cycles after the request has been granted. Note that `data_rvalid_i` must also be set high to signal the end of the response phase for a write transaction (although the `data_rdata_i` has no meaning in that case). When multiple granted requests are outstanding, it is assumed that the memory requests will be kept in-order and one `data_rvalid_i` will be signalled for each of them, in the order they were issued.

Figure 7.1, Figure 7.2, Figure 7.3 and Figure 7.4 show example timing diagrams of the protocol.

7.3 Write buffer

CV32E40X contains a a single entry write buffer that is used for bufferable transfers. A bufferable transfer is a write transfer originating from a store instruction, where the write address is inside a bufferable region defined by the PMA (*Physical Memory Attribution (PMA)*). Note that Store Conditional (SC) and Atomic Memory Operation (AMO) instructions will not utilize the write buffer.

The write buffer (when not full) allows CV32E40X to proceed executing instructions without having to wait for `data_gnt_i` = 1 and `data_rvalid_i` = 1 for these bufferable transers.

Note: On the OBI interface `data_gnt_i` = 1 and `data_rvalid_i` = 1 still need to be signaled for every transfer (as specified in [OPENHW-OBI]), also for bufferable transfers.

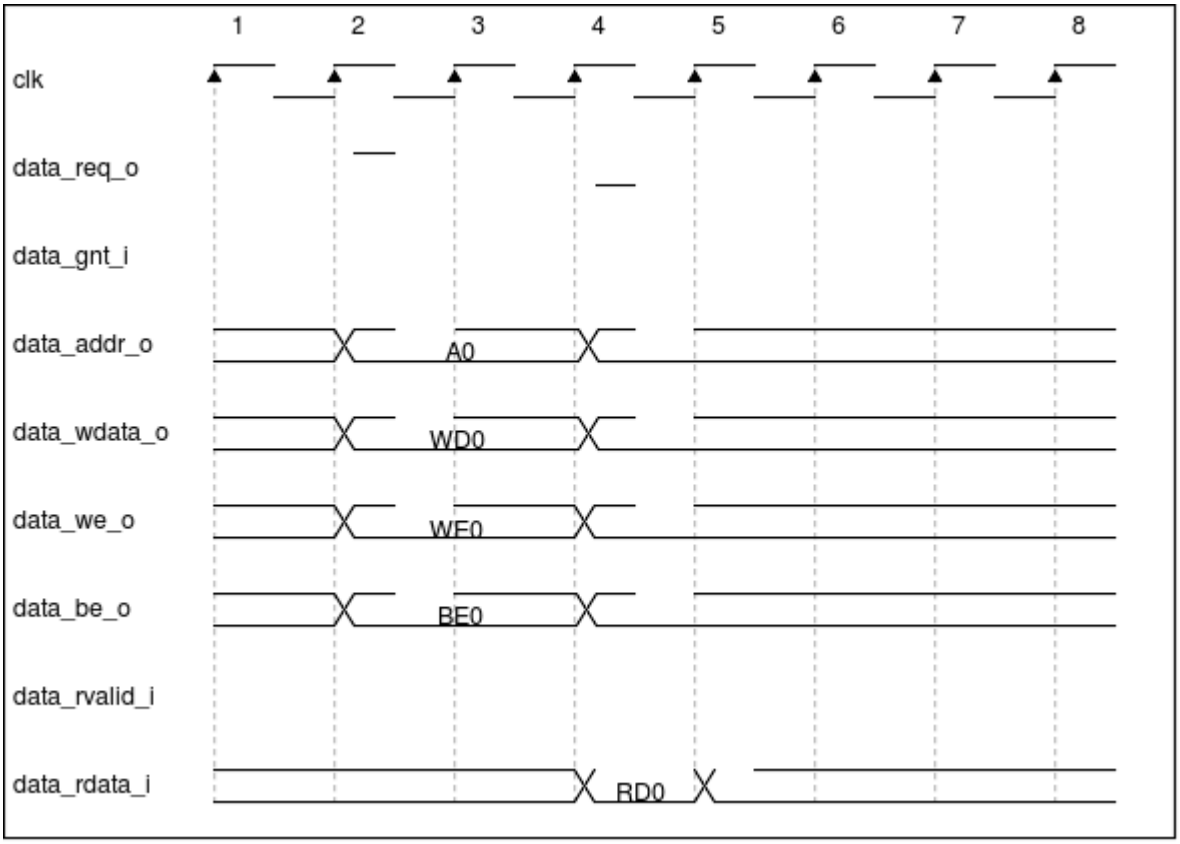


Figure 7.1: Basic Memory Transaction

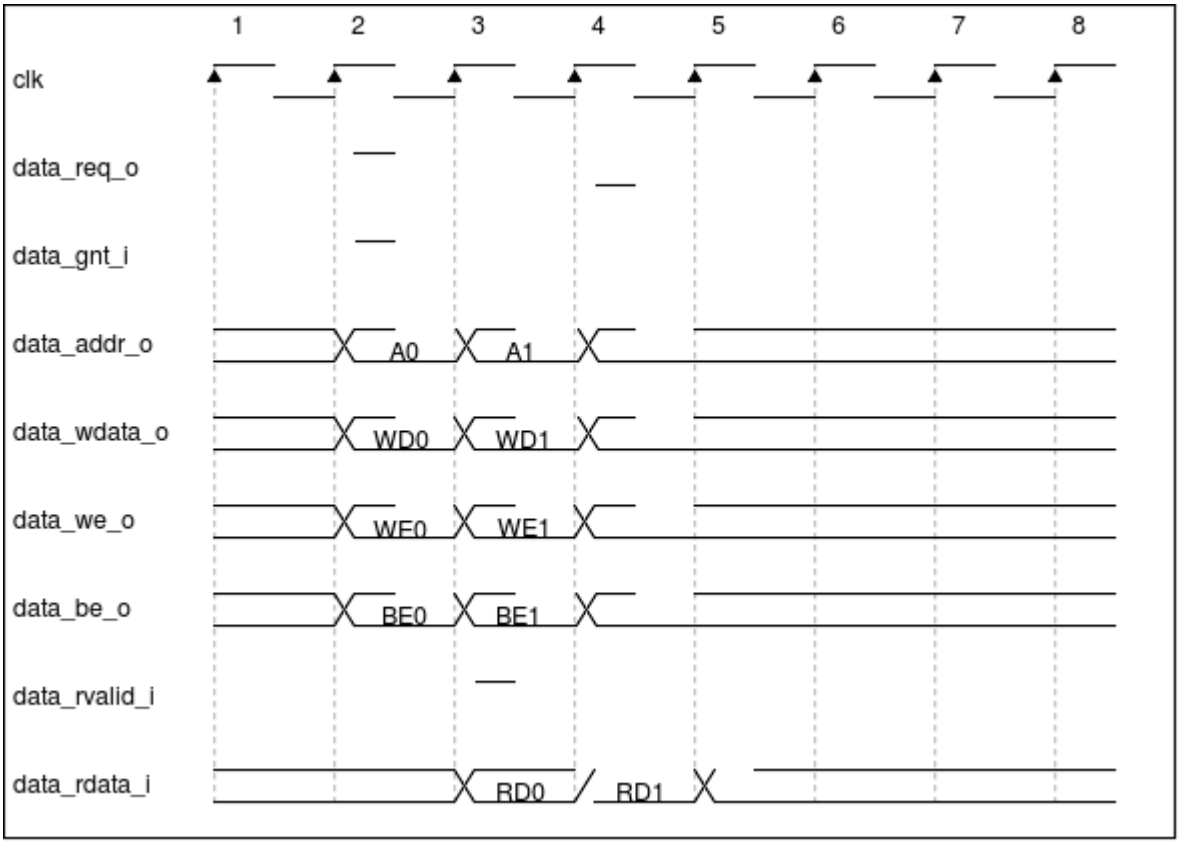


Figure 7.2: Back-to-back Memory Transactions

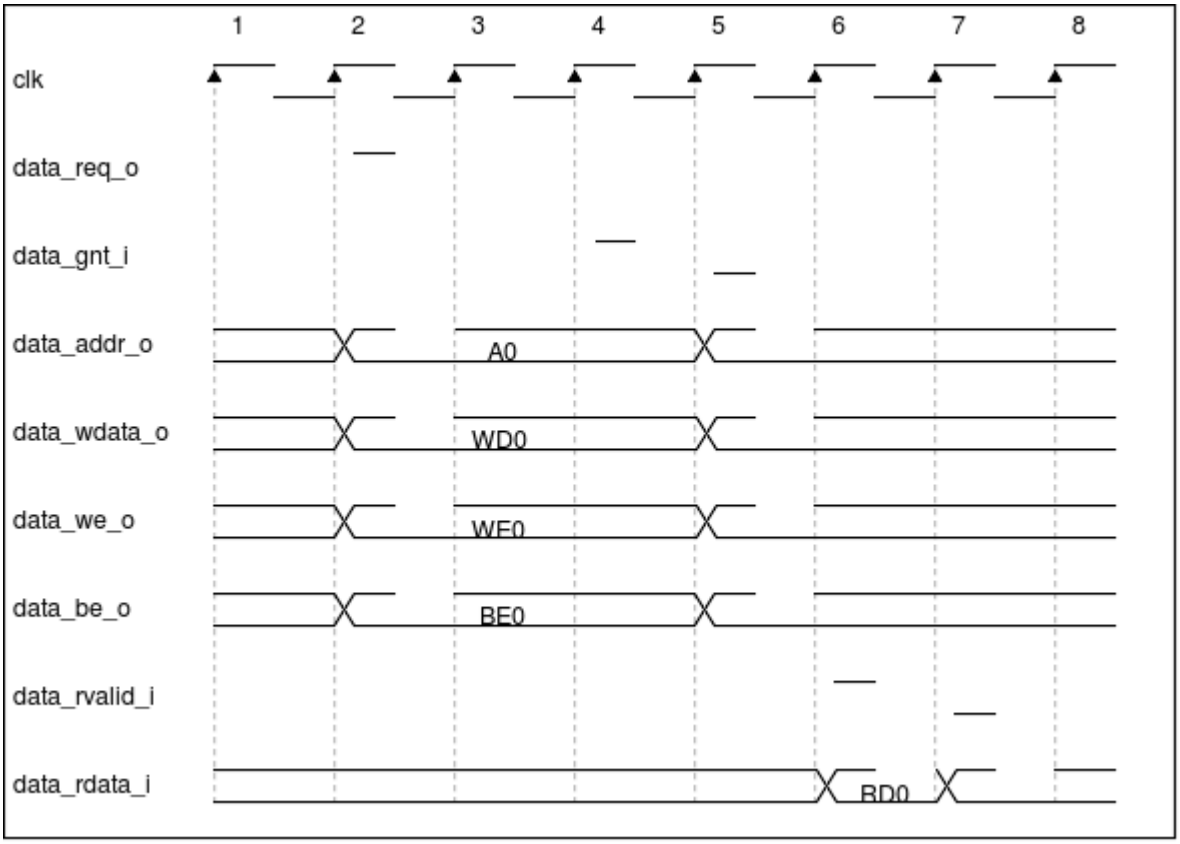


Figure 7.3: Slow Response Memory Transaction

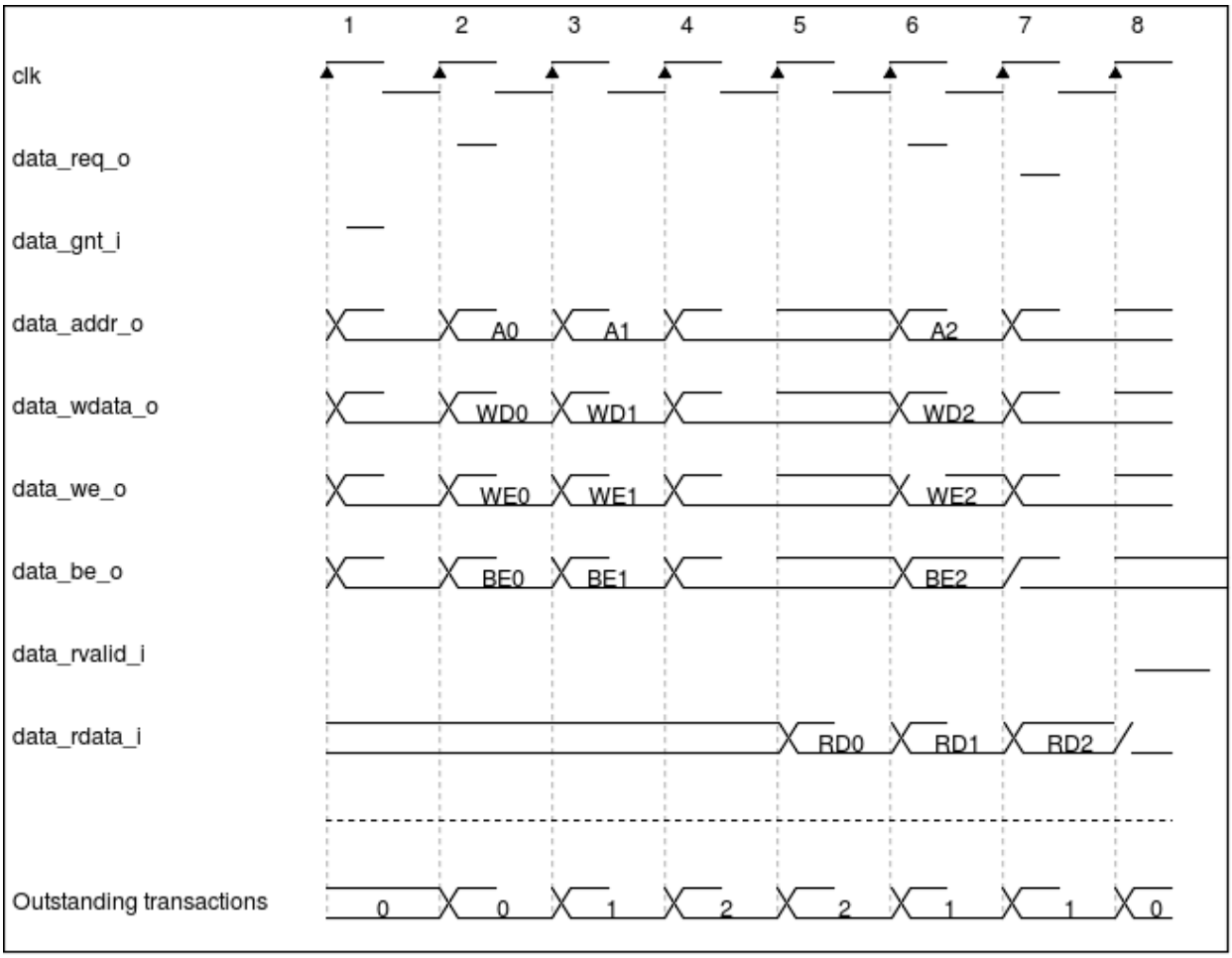


Figure 7.4: Multiple Outstanding Memory Transactions

Bus transfers will occur in program order, no matter if transfers are bufferable and non-bufferable. Transactions in the write buffer must be completed before the CV32E40X is able to:

- Retire a `fence` instruction
- Retire a `fence.i` instruction
- Enter SLEEP mode

7.4 Atomics

CV32E40X supports exclusive transactions and atomic transactions if `A_EXT` = 1. For atomic transactions CV32E40X does however **not** provide a full implementation of the `A` extension as it is assumed that CV32E40X is used in combination with an external adapter that transforms the OBI transactions (see [OPENHW-OBI]) into the required *read-modify-write* sequences. For more information about Atomic instructions, see *Atomic instructions*.

CHAPTER

EIGHT

ATOMIC INSTRUCTIONS

CV32E40X supports exclusive transactions and atomic transactions if `A_EXT` = 1.

8.1 Load-Reserved/Store-Conditional Instructions

The `lr.w` and `sc.w` instructions are supported if `A_EXT` = 1. These instructions perform exclusive transactions via the data OBI interface (i.e. `data_atop_o[5]` = 1). The `data_atop_o` signal will indicate the type of exclusive transaction as specified in [OPENHW-OBI].

The definition of the related reservation set as well as registering or invalidating a reservation is outside the scope of CV32E40X.

Exclusive transaction success of `lr.w` and `sc.w` instructions is signaled with `data_err_i` = 0 and `data_exokay_i` = 1. Exclusive transaction failure of `lr.w` and `sc.w` instructions is signaled with `data_err_i` = 0 and `data_exokay_i` = 0. Bus errors for `lr.w` and `sc.w` instructions are signaled with `data_err_i` = 1 and `data_exokay_i` = 0.

If a `sc.w` succeeds CV32E40X writes 0 `rd`. If a `sc.w` fails CV32E40X writes a nonzero value (1) to `rd`. CV32E40X ignores the `data_exokay_i` signal for `lr.w` instructions and will therefore **not** detect the failure of `lr.w` instructions. If a `lr.w` fails because it is attempted on a region without support for exclusive transactions, then a following `sc.w` will fail as well. The PMA's `atomic` attribute can be used to detect attempts to perform any type of atomic transaction (including `lr.w` and `sc.w`) on regions not supporting atomic transactions.

8.2 Atomic Memory Operations

The `amoswap.w`, `amoadd.w`, `amoand.w`, `amoor.w`, `amoxor.w`, `amomax[u].w` and `amomin[u].w` instructions are supported if `A_EXT` = 1. These instructions perform atomic memory operations (AMOs).

Atomic memory operation (AMO) instructions perform read-modify-write operations for multiprocessor synchronization. They atomically load a data value from the address in `rs1`, place the value into register `rd`, apply a binary operator to the loaded value and the original value in `rs2`, then store the result back to the address in `rs1`.

CV32E40X does however **not** provide a full implementation of these instructions as it is assumed that CV32E40X is used in combination with an external adapter that transforms the related OBI transactions into the required *read-modify-write* sequences.

CV32E40X will use the data OBI interface as follows for AMOs:

- `data_addr_o` is used to signal the address in `rs1`.
- `data_atop_o` is used to signal the AMO as specified in [OPENHW-OBI].
- `data_wdata_o` is used to signal the original value in `rs2`.
- `data_we_o` will be 1.

- data_rdata_i is used to receive the value that is then placed into register rd.

The environment of CV32E40X is expected to do the following for AMOs:

- Atomically load a data value from the address data_addr_o, return it on data_rdata_i (even though data_we_o = 1 for this transaction), apply a binary operator as specified via data_atop_o to the loaded value and data_wdata_o and write the result to address data_addr_o.

The timing and validity of the data_rdata_i and data_wdata_o signals are the same as for non-AMOs.

CHAPTER

NINE

PHYSICAL MEMORY ATTRIBUTION (PMA)

The CV32E40X includes a Physical Memory Attribution (PMA) unit that allows compile time attribution of the physical memory map. The PMA is configured through the top level parameters `PMA_NUM_REGIONS` and `PMA_CFG[]`. The number of PMA regions is configured through the `PMA_NUM_REGIONS` parameter. Valid values are 0-16. The configuration array, `PMA_CFG[]`, must consist of `PMA_NUM_REGIONS` entries of the type `pma_cfg_t`, defined in `cv32e40x_pkg.sv`:

```
typedef struct packed {
  logic [31:0] word_addr_low;
  logic [31:0] word_addr_high;
  logic        main;
  logic        bufferable;
  logic        cacheable;
  logic        atomic;
} pma_cfg_t;
```

In case of address overlap between PMA regions, the region with the lowest index in `PMA_CFG[]` will have priority. The PMA can be deconfigured by setting `PMA_NUM_REGIONS=0`. When doing this, `PMA_CFG[]` should be left unconnected.

9.1 Address range

The address boundaries of a PMA region are set in `word_addr_low/word_addr_high`. These contain bits 33:2 of 34-bit, word aligned addresses. To get an address match, the transfer address `addr` must be in the range `{word_addr_low, 2'b00} <= addr[33:0] < {word_addr_high, 2'b00}`. Note that `addr[33:32] = 2'b00` as the CV32E40X does not support Sv32.

If `X_EXT` = 1, then the address boundaries shall be configured to be `X_MEM_WIDTH` bit aligned.

9.2 Main memory vs I/O

Memory ranges can be defined as either main (`main=1`) or I/O (`main=0`).

Code execution is allowed from main memory and main memory is considered to be idempotent. Non-aligned transactions are supported in main memory. Modifiable transactions are supported in main memory.

Code execution is not allowed from I/O regions and an instruction access fault (exception code 1) is raised when attempting to execute from such regions. I/O regions are considered to be non-idempotent and therefore the PMA will prevent speculative accesses to such regions. Non-aligned transactions are not supported in I/O regions. An attempt to perform a non-naturally aligned load access to an I/O region causes a precise load access fault (exception code 5). An attempt to perform a non-naturally aligned store access to an I/O region causes a precise store access fault (exception code 7). Modifiable/modified transactions are not supported in I/O regions. An attempt to perform a

modifiable/modified load access to an I/O region causes a precise load access fault (exception code 5). An attempt to perform a modifiable/modified store access to an I/O region causes a precise store access fault (exception code 7).

Note: The [RISC-V-ZCA_ZCB_ZCMP_ZCMT] specification leaves it to the core implementation whether `cm.push`, `cm.pop`, `cm.popret` and `cm.popretz` instructions are supported to non-idempotent memories or not. In CV32E40X the `cm.push`, `cm.pop`, `cm.popret` and `cm.popretz` instructions are **not** allowed to perform their load or store acceses to non-idempotent memories (I/O) and a load access fault (exception code 5) or store access fault (exception code 7) will occur upon the first such load or store access violating this requirement (meaning that the related `pop` or `push` might become partially executed).

Note: Modifiable transactions are transactions which allow transformations as for example merging or splitting. For example, a misaligned store word instruction that is handled as two subword transactions on the data interface is considered to use modified transactions.

Note: As execution based debug is used, the Debug Module (with code entry points defined by `dm_halt_addr_i` and `dm_exception_addr_i`) needs to be located in a memory region that supports code execution, i.e. in a region defined as main.

9.3 Bufferable and Cacheable

Accesses to regions marked as bufferable (`bufferable=1`) will result in the OBI `mem_type[0]` bit being set, except if the access was an instruction fetch, a load, or part of an atomic memory operation. Bufferable stores will utilize the write buffer, see *Write buffer*.

Accesses to regions marked as cacheable (`cacheable=1`) will result in the OBI `mem_type[1]` bit being set.

Note: The PMA must be configured such that accesses to the external debug module are non-cacheable, to enable its program buffer to function correctly.

9.4 Atomic operations

Regions supporting atomic operations can be defined by setting `atomic=1`. An attempt to perform a Load-Reserved to a region in which Atomic operations are not allowed will cause a precise load access fault (exception code 5). An attempt to perform a Store-Conditional or Atomic Memory Operation (AMO) to a region in which Atomic operations are not allowed will cause a precise store/AMO access fault (exception code 7). Note that the `atomic` attribute is only used when the RV32A extension is included.

9.5 Default attribution

If the PMA is deconfigured (`PMA_NUM_REGIONS=0`), the entire memory range will be treated as main memory (`main=1`), non-bufferable (`bufferable=0`), non-cacheable (`cacheable=0`) and atomics will be supported (`atomic=1`).

If the PMA is configured (`PMA_NUM_REGIONS > 0`), memory regions not covered by any PMA regions are treated as I/O memory (`main=0`), non-bufferable (`bufferable=0`), non-cacheable (`cacheable=0`) and atomics will not be supported (`atomic=0`).

Every instruction fetch, load and store will be subject to PMA checks and failed checks will result in an exception. PMA checks cannot be disabled. See *Exceptions and Interrupts* for details.

CHAPTER

TEN

REGISTER FILE

Source file: `rtl/cv32e40x_register_file.sv`

CV32E40X has 31 32-bit wide registers which form registers `x1` to `x31`. Register `x0` is statically bound to 0 and can only be read, it does not contain any sequential logic.

The number of read ports and the number of write ports of the register file depends on the parameter settings of CV32E40X. The register file has two read ports and one write port for the default parameter settings. If `X_EXT = 1`, then depending on the other eXtension interface parameters up to three read ports and two write ports can be instantiated. Register file reads are performed in the ID stage. Register file writes are performed in the WB stage.

10.1 General Purpose Register File

The general purpose register file is flip-flop-based. It uses regular, positive-edge-triggered flip-flops to implement the registers.

CHAPTER

ELEVEN

EXTENSION INTERFACE

The eXtension interface, also called `CORE-V-XIF`, enables extending CV32E40X with (custom or standardized) instructions without the need to change the RTL of CV32E40X itself. Extensions can be provided in separate modules external to CV32E40X and are integrated at system level by connecting them to the eXtension interface.

The eXtension interface provides low latency (tightly integrated) read and write access to the CV32E40X register file. All opcodes which are not used (i.e. considered to be invalid) by CV32E40X can be used for extensions. It is recommended however that custom instructions do not use opcodes that are reserved/used by RISC-V International.

The eXtension interface enables extension of CV32E40X with:

- Custom ALU type instructions.
- Custom load/store type instructions.
- Custom CSRs and related instructions.

Control-Tranfer type instructions (e.g. branches and jumps) are not supported via the eXtension interface.

Note: CV32E40X does for example not implement the **F** (single-precision floating-point), **P** (Packed SIMD) or **V** (Vector) extensions internal to the core. Such extensions are considered good candidates to be implemented as external coprocessor functionality connected via the eXtension interface.

11.1 CORE-V-XIF

The eXtension interface of complies to the [OPENHW-XIF] specification. The reader is deferred to [OPENHW-XIF] for explanation of the interface protocol and semantics. Here we only list the top level interface pins to clarify the mapping of CV32E40X's SystemVerilog interfaces to CV32E40X signals.

11.1.1 Compressed interface

Table 11.1 describes the compressed interface signals.

Table 11.1: Compressed interface signals

Signal	Type	Direction	Description
xif_compressed_if.compressed_valid	logic	output	Compressed request valid. Request to uncompress a compressed instruction.
xif_compressed_if.compressed_ready	logic	input	Compressed request ready. The transactions signaled via compressed_req and compressed_resp are accepted when compressed_valid and compressed_ready are both 1.
xif_compressed_if.compressed_req	x_compressed_req_t	output	Compressed request packet.
xif_compressed_if.compressed_resp	x_compressed_resp_t	input	Compressed response packet.

11.1.2 Issue interface

Table 11.2 describes the issue interface signals.

Table 11.2: Issue interface signals

Signal	Type	Direction	Description
xif_issue_if.issue_valid	logic	output	Issue request valid. Indicates that CV32E40X wants to offload an instruction.
xif_issue_if.issue_ready	logic	input	Issue request ready. The transaction signaled via issue_req and issue_resp is accepted when issue_valid and issue_ready are both 1.
xif_issue_if.issue_req	x_issue_req_t	output	Issue request packet.
xif_issue_if.issue_resp	x_issue_resp_t	input	Issue response packet.

11.1.3 Commit interface

Table 11.3 describes the commit interface signals.

Table 11.3: Commit interface signals

Signal	Type	Direction	Description
xif_commit_if.commit_valid	logic	output	Commit request valid. Indicates that CV32E40X has valid commit or kill information for an offloaded instruction. There is no corresponding ready signal (it is implicit and assumed 1). The coprocessor shall be ready to observe the commit_valid and commit_kill signals at any time coincident or after an issue transaction initiation.
xif_commit_if.commit	x_commit_t	output	Commit packet.

11.1.4 Memory (request/response) interface

Table 11.4 describes the memory (request/response) interface signals.

Table 11.4: Memory (request/response) interface signals

Signal	Type	Direction	Description
xif_mem_if.mem_valid	logic	input	Memory (request/response) valid. Indicates that the coprocessor wants to perform a memory transaction for an offloaded instruction.
xif_mem_if.mem_ready	logic	output	Memory (request/response) ready. The memory (request/response) signaled via mem_req is accepted by CV32E40X when mem_valid and mem_ready are both 1.
xif_mem_if.mem_req	x_mem_req_t	input	Memory request packet.
xif_mem_if.mem_resp	x_mem_resp_t	output	Memory response packet. Response to memory request (e.g. PMA check response). Note that this is not the memory result.

11.1.5 Memory result interface

Table 11.5 describes the memory result interface signals.

Table 11.5: Memory result interface signals

Signal	Type	Direction	Description
xif_mem_result_if.mem_result_valid	logic	output	Memory result valid. Indicates that CV32E40X has a valid memory result for the corresponding memory request. There is no corresponding ready signal (it is implicit and assumed 1). The coprocessor must be ready to accept mem_result whenever mem_result_valid is 1.
xif_mem_result_if.mem_result	x_mem_result_t	output	Memory result packet.

11.1.6 Result interface

Table 11.6 describes the result interface signals.

Table 11.6: Result interface signals

Signal	Type	Direction	Description
xif_result_if.result_valid	logic	input	Result request valid. Indicates that the coprocessor has a valid result (write data or exception) for an offloaded instruction.
xif_result_if.result_ready	logic	output	Result request ready. The result signaled via result is accepted by the core when result_valid and result_ready are both 1.
xif_result_if.result	x_result_t	input	Result packet.

11.2 Integration

When integrating the eXtension interface, all parameters used by both CV32E40X, the SystemVerilog interface and the coprocessor/interconnect must match. Parameters or localparams should be used at the hierarchy level above CV32E40X as shown in Figure 11.1.

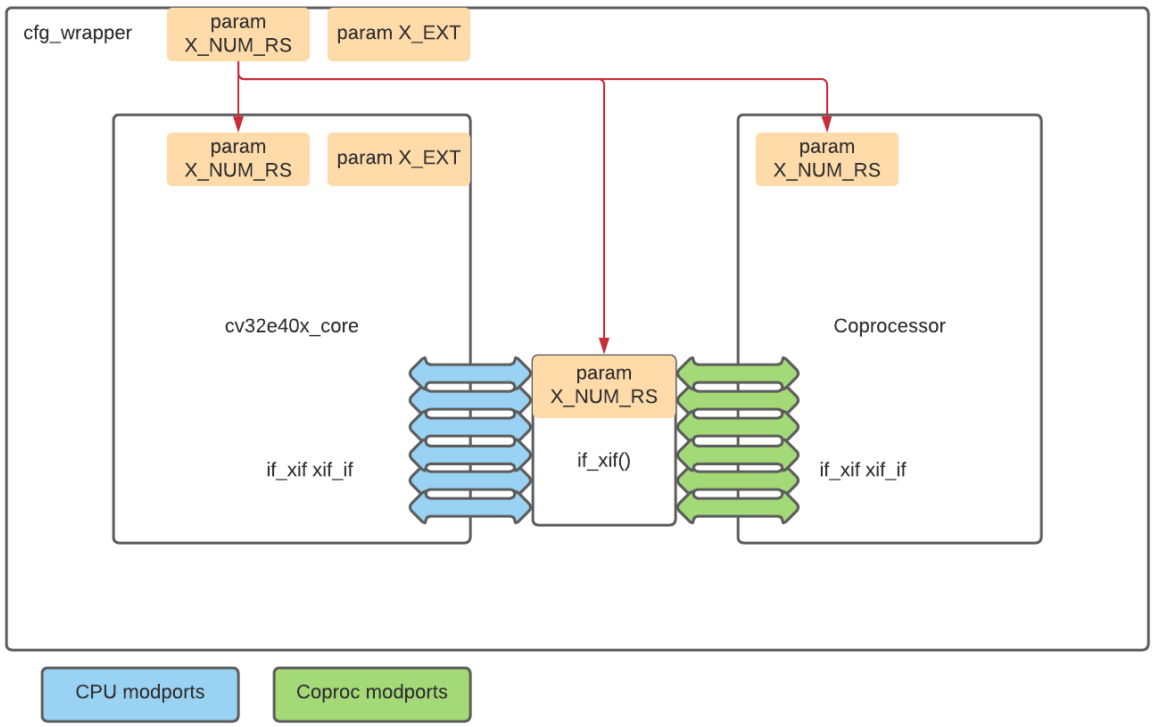


Figure 11.1: eXtenstion interface integration

11.3 Timing

For optimal system level performance CV32E40X, the coprocessor(s) and the optional interconnect are advised to adhere to the timing budgets shown in Figure 11.2.

All eXtension interface signals not explicitly covered in Figure 11.2 should follow the generic timing budget that is outlined - 20% for the processor, 20% for the interconnect and 60% for the coprocessor.

The CV32E40X github repository contains a constraints file as seen from the processor: cv32e40x_core.sdc

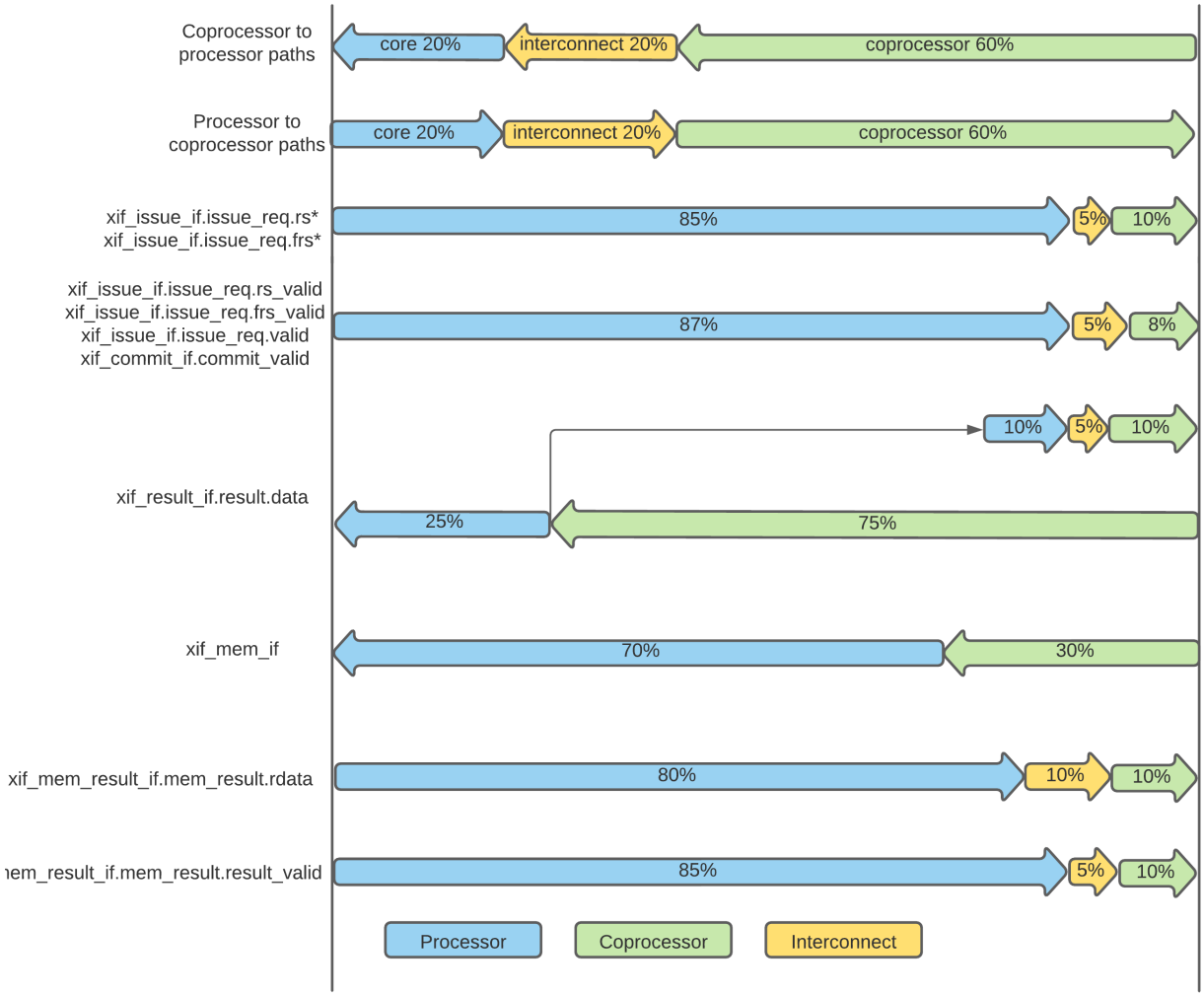


Figure 11.2: eXtenstion interface timing budgets

CHAPTER

TWELVE

FENCE.I EXTERNAL HANDSHAKE

CV32E40X includes an external handshake that will be exercised upon execution of the fence.i instruction. The handshake is composed of the signals `fencei_flush_req_o` and `fencei_flush_ack_i` and can for example be used to flush an externally connected cache.

The `fencei_flush_req_o` signal will go high upon executing a `fence.i` instruction ([RISC-V-UNPRIV]) once possible earlier store instructions have fully completed (including emptying of the the write buffer). The `fencei_flush_req_o` signal will go low again the cycle after sampling both `fencei_flush_req_o` and `fencei_flush_ack_i` high. Once `fencei_flush_req_o` has gone low again a branch will be taken to the instruction after the `fence.i` thereby flushing possibly prefetched instructions.

Fence instructions are not impacted by the distinction between main and I/O regions (defined in *Physical Memory Attribution (PMA)*) and execute as a conservative fence on all operations, ignoring the predecessor and successor fields.

Note: If the `fence.i` external handshake is not used by the environment of CV32E40X, then it is recommended to tie the `fencei_flush_ack_i` to 1 in order to avoid stalling `fence.i` instructions indefinitely.

CHAPTER

THIRTEEN

SLEEP UNIT

Source File: `rtl/cv32e40x_sleep_unit.sv`

The Sleep Unit contains and controls the instantiated clock gate, see *Clock Gating Cell*, that gates `clk_i` and produces a gated clock for use by the other modules inside CV32E40X. The Sleep Unit is the only place in which `clk_i` itself is used; all other modules use the gated version of `clk_i`.

The clock gating in the Sleep Unit is impacted by the following:

- `rst_ni`
- `fetch_enable_i`
- **wfi** instruction
- **wfe** instruction

Table 13.1 describes the Sleep Unit interface.

Table 13.1: Sleep Unit interface signals

Signal	Direction	Description
`core_sleep_o`	output	Core is sleeping because of a **wfi** or **wfe** instruction. If `core_sleep_o` = 1, then `clk_i` is gated off internally and it is allowed to gate off `clk_i` externally as well. See *WFI* and *WFE* for details.
`wu_wfe_i`	input	Wake-up signal for custom **wfe** instruction. See *WFE* for details.

13.1 Startup behavior

`clk_i` is internally gated off (while signaling `core_sleep_o` = 0) during CV32E40X startup:

- `clk_i` is internally gated off during `rst_ni` assertion
- `clk_i` is internally gated off from `rst_ni` deassertion until `fetch_enable_i` = 1

After initial assertion of `fetch_enable_i`, the `fetch_enable_i` signal is ignored until after a next reset assertion.

13.2 WFI

The **wfi** instruction can under certain conditions be used to enter sleep mode awaiting a locally enabled interrupt to become pending. The operation of **wfi** is unaffected by the global interrupt bits in **mstatus**.

A **wfi** will not enter sleep mode, but will be executed as a regular **nop**, if any of the following conditions apply:

- `debug_req_i` = 1 or a debug request is pending
- The core is in debug mode
- The core is performing single stepping (debug)
- The core has a trigger match (debug)

If a **wfi** causes sleep mode entry, then `core_sleep_o` is set to 1 and `clk_i` is gated off internally. `clk_i` is allowed to be gated off externally as well in this scenario. A wake-up can be triggered by any of the following:

- A locally enabled interrupt is pending
- A debug request is pending
- Core is in debug mode

Upon wake-up `core_sleep_o` is set to 0, `clk_i` will no longer be gated internally, must not be gated off externally, and instruction execution resumes.

If one of the above wake-up conditions coincides with the **wfi** instruction, then sleep mode is not entered and `core_sleep_o` will not become 1.

Figure 13.1 shows an example waveform for sleep mode entry because of a **wfi** instruction.

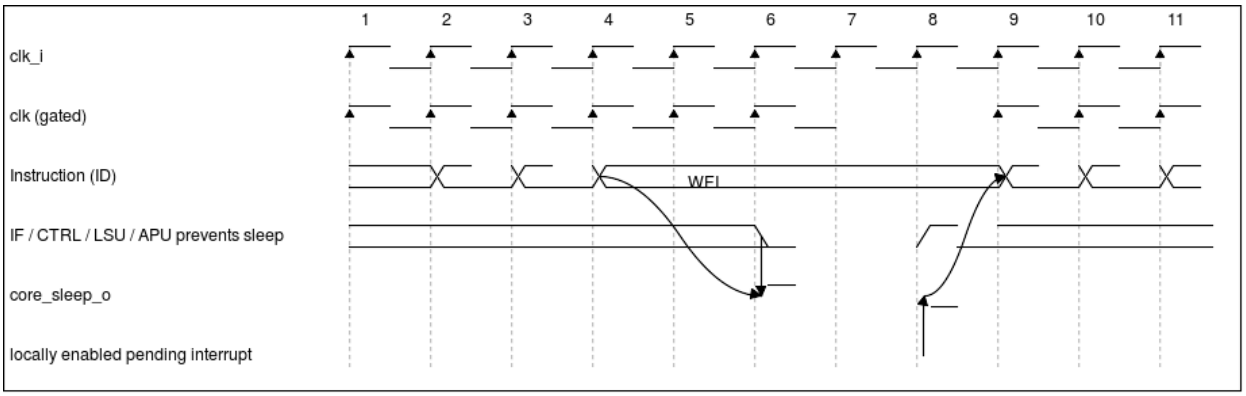


Figure 13.1: **wfi** example

13.3 WFE

The custom **wfe** instruction behaves exactly as the **wfi** instruction, except that a wake-up can additionally be triggered by asserting `wu_wfe_i`.

The **wfe** instruction is encoded as a custom SYSTEM instruction with opcode `0x8C00_0073`.

CHAPTER

FOURTEEN

CONTROL AND STATUS REGISTERS

14.1 CSR Map

Table 14.1 lists all implemented CSRs. To columns in Table 14.1 may require additional explanation:

The **Parameter** column identifies those CSRs that are dependent on the value of specific compile/synthesis parameters. If these parameters are not set as indicated in Table 14.1 then the associated CSR is not implemented. If the parameter column is empty then the associated CSR is always implemented.

The **Privilege** column indicates the access mode of a CSR. The first letter indicates the lowest privilege level required to access the CSR. Attempts to access a CSR with a higher privilege level than the core is currently running in will throw an illegal instruction exception. This is largely a moot point for the CV32E40X as it only supports machine and debug modes. The remaining letters indicate the read and/or write behavior of the CSR when accessed by the indicated or higher privilge level:

- **RW**: CSR is **read-write**. That is, CSR instructions (e.g. csrrw) may write any value and that value will be returned on a subsequent read (unless a side-effect causes the core to change the CSR value).
- **RO**: CSR is **read-only**. Writes by CSR instructions raise an illegal instruction exception.

Writes of a non-supported value to **WLRL** bitfields of a **RW** CSR do not result in an illegal instruction exception. The exact bitfield access types, e.g. **WLRL** or **WARL**, can be found in the RISC-V privileged specification.

Reads or writes to a CSR that is not implemented will result in an illegal instruction exception.

Table 14.1: Control and Status Register Map

CSR Address	Name	Privilege	Parameter	Description
Machine CSRs				
0x300	`mstatus`	MRW		Machine Status (lower 32 bits).
0x301	`misa`	MRW		Machine ISA
0x304	`mie`	MRW		Machine Interrupt Enable Register
0x305	`mtvec`	MRW		Machine Trap-Handler Base Address
0x307	`mtvt`	MRW	`SMCLIC` = 1	Machine Trap-Handler Vector Table Base Address
0x310	`mstatush`	MRW		Machine Status (upper 32 bits).
0x320	`mcountinhibit`	MRW		(HPM) Machine Counter-Inhibit Register
0x323	`mhpmevent3`	MRW		(HPM) Machine Performance-Monitoring Event Selector 3
. . . .				

continues on next page

Table 14.1 – continued from previous page

CSR Address	Name	Privilege	Parameter	Description
0x33F	mhpmevent31	MRW		(HPM) Machine Performance-Monitoring Event Selector 31
0x340	mscratch	MRW		Machine Scratch
0x341	mepc	MRW		Machine Exception Program Counter
0x342	mcause	MRW		Machine Trap Cause
0x343	mtval	MRW		Machine Trap Value
0x344	mip	MRW		Machine Interrupt Pending Register
0x345	mnxti	MRW	SMCLIC = 1	Interrupt handler address and enable modifier
0x346	mintstatus	MRW	SMCLIC = 1	Current interrupt levels
0x347	mintthresh	MRW	SMCLIC = 1	Interrupt-level threshold
0x348	mscratchcsw	MRW	SMCLIC = 1	Conditional scratch swap on priv mode change
0x349	mscratchcswl	MRW	SMCLIC = 1	Conditional scratch swap on level change
0x34A	mclicbase	MRW	SMCLIC = 1	CLIC Base Register
0x7A0	tselect	MRW	DBG_NUM_TRIGGERS > 0	Trigger Select Register
0x7A1	tdata1	MRW	DBG_NUM_TRIGGERS > 0	Trigger Data Register 1
0x7A2	tdata2	MRW	DBG_NUM_TRIGGERS > 0	Trigger Data Register 2
0x7A3	tdata3	MRW	DBG_NUM_TRIGGERS > 0	Trigger Data Register 3
0x7A4	tinfo	MRW	DBG_NUM_TRIGGERS > 0	Trigger Info
0x7A5	tcontrol	MRW	DBG_NUM_TRIGGERS > 0	Trigger Control
0x7B0	dcsr	DRW		Debug Control and Status
0x7B1	dpc	DRW		Debug PC
0x7B2	dscratch0	DRW		Debug Scratch Register 0
0x7B3	dscratch1	DRW		Debug Scratch Register 1
0xB00	mcycle	MRW		(HPM) Machine Cycle Counter
0xB02	minstret	MRW		(HPM) Machine Instructions-Retired Counter
0xB03	mhpmcounter3	MRW		(HPM) Machine Performance-Monitoring Counter 3
. . . .				
0xB1F	mhpmcounter31	MRW		(HPM) Machine Performance-Monitoring Counter 31
0xB80	mcycleh	MRW		(HPM) Upper 32 Machine Cycle Counter
0xB82	minstreth	MRW		(HPM) Upper 32 Machine Instructions-Retired Counter
0xB83	mhpmcounterh3	MRW		(HPM) Upper 32 Machine Performance-Monitoring Counter 3
. . . .				
0xB9F	mhpmcounterh31	MRW		(HPM) Upper 32 Machine Performance-Monitoring Counter 31
0xF11	mvendorid	MRO		Machine Vendor ID
0xF12	marchid	MRO		Machine Architecture ID
0xF13	mimpid	MRO		Machine Implementation ID
0xF14	mhartid	MRO		Hardware Thread ID

continues on next page

Table 14.1 – continued from previous page

CSR Address	Name	Privilege	Parameter	Description
0xF15	mconfigptr	MRO		Machine Configuration Pointer

Table 14.2: Control and Status Register Map (Unprivileged and User-Level CSRs)

CSR Address	Name	Privilege	Parameter	Description
Unprivileged and User-Level CSRs				
0x017	jvt	URW		Table jump base vector and control register

Table 14.3: Control and Status Register Map (additional CSRs for Zicntr)

CSR Address	Name	Privilege	Parameter	Description
User CSRs				
0xC00	cycle	URO		Cycle Counter
0xC01	time	URO		Time
0xC02	instret	URO		Instructions-Retired Counter
0xC80	cycleh	URO		Upper 32 Cycle Counter
0xC81	timeh	URO		Upper 32 Time
0xC82	instreth	URO		Upper 32 Instructions-Retired Counter

Table 14.4: Control and Status Register Map (additional CSRs for Zihpm)

CSR Address	Name	Privilege	Parameter	Description
User CSRs				
0xC03	hpmcounter3	URO		(HPM) Performance-Monitoring Counter 3
. . . . .				
0xC1F	hpmcounter31	URO		(HPM) Performance-Monitoring Counter 31
0xC83	hpmcounterh3	URO		(HPM) Upper 32 Performance-Monitoring Counter 3
. . . . .				
0xC9F	hpmcounterh31	URO		(HPM) Upper 32 Performance-Monitoring Counter 31

14.2 CSR Descriptions

What follows is a detailed definition of each of the CSRs listed above. The **R/W** column defines the access mode behavior of each bit field when accessed by the privilege level specified in Table 14.1 (or a higher privilege level):

- **R**: **read** fields are not affected by CSR write instructions. Such fields either return a fixed value, or a value determined by the operation of the core.
- **RW**: **read/write** fields store the value written by CSR writes. Subsequent reads return either the previously written value or a value determined by the operation of the core.
- **WARL**: **write-any-read-legal** fields store only legal values written by CSR writes. The WARL keyword can optionally be followed by a legal value (or comma separated list of legal values) enclosed in brackets. If the legal value(s) are not specified, then all possible values are considered valid. For example, a WARL (0x0) field supports only the value 0x0. Any value may be written, but all reads would return 0x0 regardless of the value

being written to it. A WARL field may support more than one value. If an unsupported value is (attempted to be) written to a WARL field, the value marked with an asterix (the so-called resolution value) is written. If there is no such predefined resolution value, then the original (legal) value of the bitfield is preserved.

- **WPRI**: Software should ignore values read from these fields, and preserve the values when writing.

Note: The **R/W** information does **not** impact whether CSR accesses result in illegal instruction exceptions or not.

14.2.1 Jump Vector Table (`jvt`)

CSR Address: 0x017

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:10	WARL	**BASE[31:10]**: Table Jump Base Address, 1024 byte aligned.
9:6	WARL (0x0)	**BASE[9:6]**: Table Jump Base Address, 1024 byte aligned. `jvt[9:6]` is hard-wired to 0x0.
5:0	WARL (0x0)	**MODE**: Jump table mode

Table jump base vector and control register

Note: Memory writes to the `jvt` based vector table require an instruction barrier (`fence.i`) to guarantee that they are visible to the instruction fetch (see *Fence.i external handshake* and [RISC-V-UNPRIV]).

14.2.2 Machine Status (`mstatus`)

CSR Address: 0x300

Reset Value: defined (based on *X_EXT*`, `X_ECS_XS)

Bit #	R/W	Description
31	R	**SD**: State Dirty. SD = ((FS == 0x3) OR (XS == 0x3) OR (VS == 0x3)).
30:23	WPRI (0x0)	Reserved. Hardwired to 0.
22	WARL (0x0)	**TSR**. Hardwired to 0.
21	WARL (0x0)	**TW**. Hardwired to 0.
20	WARL (0x0)	**TVM**. Hardwired to 0.
19	R (0x0)	**MXR**. Hardwired to 0.
18	R (0x0)	**SUM**. Hardwired to 0.
17	R (0x0)	**MPRV**. Hardwired to 0.
16:15	R / R (0x0)	**XS**: Other Extension Context Status. R with reset value defined by X_ECS_XS if X_EXT == 1, hardwired to 0 otherwise.
14:13	RW / WARL (0x0)	**FS**: FPU Extension Context Status. RW if X_EXT == 1, hardwired to 0 otherwise.
12:11	WARL (0x3)	**MPP**: Machine Previous Priviledge mode. Hardwired to 0x3.
10:9	RW / WPRI (0x0)	**VS**: Vector Extension Context Status. RW if X_EXT == 1, hardwired to 0 otherwise.
8	WARL (0x0)	**SPP**. Hardwired to 0.
7	RW	**MPIE**. When an exception is encountered, MPIE will be set to MIE. When the mret instruction is executed, the value of MPIE will be stored to MIE.
6	WARL (0x0)	**UBE**. Hardwired to 0.
5	R (0x0)	**SPIE**. Hardwired to 0.
4	WPRI (0x0)	Reserved. Hardwired to 0.
3	RW	**MIE**: If you want to enable interrupt handling in your exception handler, set the Interrupt Enable MIE to 1 inside your handler code.
2	WPRI (0x0)	Reserved. Hardwired to 0.
1	R (0x0)	**SIE**. Hardwired to 0.
0	WPRI (0x0)	Reserved. Hardwired to 0

14.2.3 Machine ISA (misa)

CSR Address: 0x301

Reset Value: defined (based on RV32, A_EXT, M_EXT, X_EXT, X_MISA)

Detailed:

Bit #	R/W	Description
31:30	WARL (0x1)	**MXL** (Machine XLEN).
29:26	WARL (0x0)	(Reserved).
25	WARL (0x0)	**Z** (Reserved).
24	WARL (0x0)	**Y** (Reserved).
23	WARL (0x1)	**X** (Non-standard extensions present).
22	WARL (0x0)	**W** (Reserved).
21	WARL	**V** (Tentatively reserved for Vector extension).
20	WARL (0x0)	**U** (User mode implemented).
19	WARL (0x0)	**T** (Tentatively reserved for Transactional Memory extension).
18	WARL (0x0)	**S** (Supervisor mode implemented).
17	WARL (0x0)	**R** (Reserved).
16	WARL (0x0)	**Q** (Quad-precision floating-point extension).
15	WARL	**P** (Packed-SIMD extension).
14	WARL (0x0)	**O** (Reserved).
13	WARL (0x0)	**N**
12	WARL	**M** (Integer Multiply/Divide extension).
11	WARL (0x0)	**L** (Tentatively reserved for Decimal Floating-Point extension).
10	WARL (0x0)	**K** (Reserved).
9	WARL (0x0)	**J** (Tentatively reserved for Dynamically Translated Languages extension).
8	WARL	**I** (RV32I/64I/128I base ISA).
7	WARL (0x0)	**H** (Hypervisor extension).
6	WARL (0x0)	**G** (Additional standard extensions present).
5	WARL	**F** (Single-precision floating-point extension).
4	WARL	**E** (RV32E base ISA).
3	WARL (0x0)	**D** (Double-precision floating-point extension).
2	WARL (0x1)	**C** (Compressed extension).
1	WARL (0x0)	**B** Reserved.
0	WARL	**A** (Atomic extension).

All bitfields in the `misa` CSR read as 0 except for the following:

- **A** = 1 if `A_EXT` == 1
- **C** = 1
- **I** = 1 if `RV32` == RV32I
- **E** = 1 if `RV32` == RV32E
- **M** = 1 if `M_EXT` == M
- **MXL** = 1 (i.e. XLEN = 32)
- **X** = 1
- If `X_EXT` == 1, then the value of `X_MISA` is ORed into the `misa` CSR.

Note: The `WARL` in above table is depending on *X_EXT*`. If `X_EXT` == 1, then some of the `misa` bits can read values depending on the value of `X_MISA`.

14.2.4 Machine Interrupt Enable Register (mie) - SMCLIC == 0

CSR Address: 0x304

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:16	RW	Machine Fast Interrupt Enables: Set bit x to enable interrupt irq_i[x].
15:12	WARL (0x0)	Reserved. Hardwired to 0.
11	RW	**MEIE**: Machine External Interrupt Enable, if set, irq_i[11] is enabled.
10	WARL (0x0)	Reserved. Hardwired to 0.
9	WARL (0x0)	**SEIE**. Hardwired to 0
8	WARL (0x0)	Reserved. Hardwired to 0.
7	RW	**MTIE**: Machine Timer Interrupt Enable, if set, irq_i[7] is enabled.
6	WARL (0x0)	Reserved. Hardwired to 0.
5	WARL (0x0)	**STIE**. Hardwired to 0.
4	WARL (0x0)	Reserved. Hardwired to 0.
3	RW	**MSIE**: Machine Software Interrupt Enable, if set, irq_i[3] is enabled.
2	WARL (0x0)	Reserved. Hardwired to 0.
1	WARL (0x0)	**SSIE**. Hardwired to 0.
0	WARL (0x0)	Reserved. Hardwired to 0.

14.2.5 Machine Interrupt Enable Register (mie) - SMCLIC == 1

CSR Address: 0x304

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:0	WARL (0x0)	Reserved. Hardwired to 0.

Note: In CLIC mode the `mie` CSR is replaced by separate memory-mapped interrupt enables (`clicintie`).

14.2.6 Machine Trap-Vector Base Address (mtvec) - SMCLIC == 0

CSR Address: 0x305

Reset Value: Defined

Detailed:

Bit #	R/W	Description
31:7	WARL	**BASE[31:7]**: Trap-handler base address, always aligned to 128 bytes.
6:2	WARL (0x0)	**BASE[6:2]**: Trap-handler base address, always aligned to 128 bytes. `mtvec[6:2]` is hardwired to 0x0.
1:0	WARL (0x0, 0x1)	**MODE**: Interrupt handling mode. 0x0 = non-vectored CLINT mode, 0x1 = vectored CLINT mode.

The initial value of `mtvec` is equal to {**mtvec_addr_i[31:7]**, 5'b0, 2'b01}.

When an exception or an interrupt is encountered, the core jumps to the corresponding handler using the content of the `mtvec[31:7]` as base address. Both non-vectored CLINT mode and vectored CLINT mode are supported.

Upon an NMI in non-vectored CLINT mode the core jumps to **mtvec[31:7]**, 5'h0, 2'b00} (i.e. index 0). Upon an NMI in vectored CLINT mode the core jumps to **mtvec[31:7]**, 5'hF, 2'b00} (i.e. index 15).

Note: For NMIs the exception codes in the `mcause` CSR do not match the table index as for regular interrupts.

Note: Memory writes to the `mtvec` based vector table require an instruction barrier (`fence.i`) to guarantee that they are visible to the instruction fetch (see *Fence.i external handshake* and [RISC-V-UNPRIV]).

14.2.7 Machine Trap-Vector Base Address (`mtvec`) - `SMCLIC == 1`

CSR Address: 0x305

Reset Value: Defined

Detailed:

Bit #	R/W	Description
31:7	WARL	**BASE[31:7]**: Trap-handler base address, always aligned to 128 bytes.
6	WARL (0x0)	**BASE[6]**: Trap-handler base address, always aligned to 128 bytes. `mtvec[6]` is hardwired to 0x0.
5:2	WARL (0x0)	**SUBMODE**: Sub mode. Reserved for future use.
1:0	WARL (0x3)	**MODE**: Interrupt handling mode. Always CLIC mode.

The initial value of `mtvec` is equal to {**mtvec_addr_i[31:7]**, 1'b0, 6'b000011}.

Upon an NMI in CLIC mode the core jumps to **mtvec[31:7]**, 5'h0, 2'b00} (i.e. index 0).

Note: Memory writes to the `mtvec` based vector table require an instruction barrier (`fence.i`) to guarantee that they are visible to the instruction fetch (see *Fence.i external handshake* and [RISC-V-UNPRIV]).

14.2.8 Machine Trap Vector Table Base Address (`mtvt`)

CSR Address: 0x307

Reset Value: 0x0000_0000

Include Condition: `SMCLIC` = 1

Detailed:

Bit #	R/W	Description
31:N	RW	**BASE[31:N]**: Trap-handler vector table base address. N = maximum(6, 2+SMCLIC_ID_WIDTH). See note below for alignment restrictions.
N-1:6	WARL (0x0)	**BASE[N-1:6]**: Trap-handler vector table base address. This field is only defined if N > 6. N = maximum(6, 2+SMCLIC_ID_WIDTH). `mtvt[N-1:6]` is hardwired to 0x0. See note below for alignment restrictions.
5:0	R (0x0)	Reserved. Hardwired to 0.

Note: The `mtvt` CSR holds the base address of the trap vector table, which has its alignment restricted to both at least 64-bytes and to `2^(2+SMCLIC_ID_WIDTH)` bytes or greater power-of-two boundary. For example if `SMCLIC_ID_WIDTH` = 8, then 256 CLIC interrupts are supported and the trap vector table is aligned to 1024 bytes, and therefore **BASE[9:6]** will be WARL (0x0).

Note: Memory writes to the `mtvt` based vector table require an instruction barrier (`fence.i`) to guarantee that they are visible to the instruction fetch (see *Fence.i external handshake* and [RISC-V-UNPRIV]).

14.2.9 Machine Status (`mstatush`)

CSR Address: 0x310

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Definition
31:6	WPRI (0x0)	Reserved. Hardwired to 0.
5	WARL (0x0)	**MBE**. Hardwired to 0.
4	WARL (0x0)	**SBE**. Hardwired to 0.
3:0	WPRI (0x0)	Reserved. Hardwired to 0.

14.2.10 Machine Counter-Inhibit Register (`mcountinhibit`)

CSR Address: 0x320

Reset Value: Defined

The performance counter inhibit control register. The default value is to inihibit all implemented counters out of reset. The bit returns a read value of 0 for non implemented counters.

Detailed:

Bit#	R/W	Description
31:3	WARL	`mhpmcounter3` - `mhpmcounter31` inhibits. Depends on `NUM_MHPMCOUNTERS` (i.e. bits related to non-implemented counters always read as 0).
2	WARL	**IR**: `minstret` inhibit
1	WARL (0x0)	Hardwired to 0.
0	WARL	**CY**: `mcycle` inhibit

14.2.11 Machine Performance Monitoring Event Selector (mhpmevent3 .. mhpmevent31)

CSR Address: 0x323 - 0x33F

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Definition
31:16	WARL (0x0)	Hardwired to 0.
15:0	WARL	**SELECTORS**. Each bit represents a unique event to count.

The event selector fields are further described in Performance Counters section. Non implemented counters always return a read value of 0.

14.2.12 Machine Scratch (mscratch)

CSR Address: 0x340

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:0	RW	Scratch value

14.2.13 Machine Exception PC (mepc)

CSR Address: 0x341

Reset Value: 0x0000_0000

Bit #	R/W	Description
31:1	WARL	Machine Expection Program Counter 31:1
0	WARL (0x0)	Hardwired to 0.

When an exception is encountered, the current program counter is saved in MEPC, and the core jumps to the exception address. When a mret instruction is executed, the value from MEPC replaces the current program counter.

14.2.14 Machine Cause (mcause) - SMCLIC == 0

CSR Address: 0x342

Reset Value: 0x0000_0000

Bit #	R/W	Description
31	RW	**INTERRUPT**. This bit is set when the exception was triggered by an interrupt.
30:11	WLRL (0x0)	**EXCCODE[30:11]**. Hardwired to 0.
10:0	WLRL	**EXCCODE[10:0]**. See note below.

Note: Software accesses to *mcause[10:0]* must be sensitive to the WLRL field specification of this CSR. For example, when *mcause[31]* is set, writing 0x1 to *mcause[1]* (Supervisor software interrupt) will result in UNDEFINED behavior.

14.2.15 Machine Cause (mcause) - SMCLIC == 1

CSR Address: 0x342

Reset Value: 0x3000_0000

Bit #	R/W	Description
31	RW	**INTERRUPT**. This bit is set when the exception was triggered by an interrupt.
30	RW	**MINHV**. Set by hardware at start of hardware vectoring, cleared by hardware at end of successful hardware vectoring.
29:28	WARL (0x3)	**MPP**: Previous privilege mode. Same as `mstatus.MPP`
27	RW	**MPIE**: Previous interrupt enable. Same as `mstatus.MPIE`
26:24	RW	Reserved. Hardwired to 0.
23:16	RW	**MPIL**: Previous interrupt level.
15:12	WARL (0x0)	Reserved. Hardwired to 0.
11	WLRL (0x0)	**EXCCODE[11]**
10:0	WLRL	**EXCCODE[10:0]**

Note: `mcause.MPP` and `mstatus.MPP` mirror each other. `mcause.MPIE` and `mstatus.MPIE` mirror each other. Reading or writing the fields `MPP`/`MPIE` in `mcause` is equivalent to reading or writing the homonymous field in `mstatus`.

14.2.16 Machine Trap Value (mtval)

CSR Address: 0x343

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:0	WARL (0x0)	Hardwired to 0.

14.2.17 Machine Interrupt Pending Register (mip) - SMCLIC == 0

CSR Address: 0x344

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:16	R	Machine Fast Interrupt Enables: Interrupt irq_i[x] is pending.
15:12	WARL (0x0)	Reserved. Hardwired to 0.
11	R	**MEIP**: Machine External Interrupt Enable, if set, irq_i[11] is pending.
10	WARL (0x0)	Reserved. Hardwired to 0.
9	WARL (0x0)	**SEIP**. Hardwired to 0
8	WARL (0x0)	Reserved. Hardwired to 0.
7	R	**MTIP**: Machine Timer Interrupt Enable, if set, irq_i[7] is pending.
6	WARL (0x0)	Reserved. Hardwired to 0.
5	WARL (0x0)	**STIP**. Hardwired to 0.
4	WARL (0x0)	Reserved. Hardwired to 0.
3	R	**MSIP**: Machine Software Interrupt Enable, if set, irq_i[3] is pending.
2	WARL (0x0)	Reserved. Hardwired to 0.
1	WARL (0x0)	**SSIP**. Hardwired to 0.
0	WARL (0x0)	Reserved. Hardwired to 0.

14.2.18 Machine Interrupt Pending Register (`mip`) - `SMCLIC == 1`

CSR Address: 0x344

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:0	WARL (0x0)	Reserved. Hardwired to 0.

Note: In CLIC mode the `mip` CSR is replaced by separate memory-mapped interrupt enables (`clicintip`).

14.2.19 Machine Next Interrupt Handler Address and Interrupt Enable (`mnxti`)

CSR Address: 0x345

Reset Value: 0x0000_0000

Include Condition: `SMCLIC` = 1

Detailed:

Bit #	R/W	Description
31:0	RW	**MNXTI**: Machine Next Interrupt Handler Address and Interrupt Enable.

This register can be used by the software to service the next interrupt when it is in the same privilege mode, without incurring the full cost of an interrupt pipeline flush and context save/restore.

Note: Use of `mnxti` with non-zero `uimm` values for bits 0, 2, and 4 are reserved for future use. CV32E40X will treat such instructions as illegal instructions.

14.2.20 Machine Interrupt Status (mintstatus)

CSR Address: 0x346

Reset Value: 0x0000_0000

Include Condition: SMCLIC = 1

Detailed:

Bit #	R/W	Description
31:24	R	**MIL**: Machine Interrupt Level
23:16	R (0x0)	Reserved. Hardwired to 0.
15: 8	R (0x0)	**SIL**: Supervisor Interrupt Level, hardwired to 0.
7: 0	R (0x0)	**UIL**: User Interrupt Level, hardwired to 0.

This register holds the active interrupt level for each privilege mode. Only Machine Interrupt Level is supported.

14.2.21 Machine Interrupt-Level Threshold (mintthresh)

CSR Address: 0x347

Reset Value: 0x0000_0000

Include Condition: SMCLIC = 1

Detailed:

Bit #	R/W	Description
31:8	R (0x0)	Reserved. Hardwired to 0.
7:0	WARL	**TH**: Threshold

This register holds the machine mode interrupt level threshold.

Note: The SMCLIC_INTTHRESHBITS parameter specifies the number of bits actually implemented in the mintthresh.th field. The implemented bits are kept left justified in the most-significant bits of the 8-bit field, with the lower unimplemented bits treated as hardwired to 1.

14.2.22 Machine Scratch Swap for Priv Mode Change (mscratchcsw)

CSR Address: 0x348

Reset Value: 0x0000_0000

Include Condition: SMCLIC = 1

Detailed:

Bit #	R/W	Description
31:0	RW	**MSCRATCHCSW**: Machine scratch swap for privilege mode change

Scratch swap register for multiple privilege modes.

Note: Only the read-modify-write (swap/CSRRW) operation is useful for `mscratchcsw`. The behavior of the non-CSRRW variants (i.e. CSRRS/C, CSRRWI, CSRRS/CI) and CSRRW variants with **rd = x0** or **rs1 = x0** on `mscratchcsw` are implementation-defined. CV32E40X will treat such instructions as illegal instructions.

14.2.23 Machine Scratch Swap for Interrupt-Level Change (`mscratchcswl`)

CSR Address: 0x349

Reset Value: 0x0000_0000

Include Condition: `SMCLIC` = 1

Detailed:

Bit #	R/W	Description
31:0	RW	**MSCRATCHCSWL**: Machine Scratch Swap for Interrupt-Level Change

Scratch swap register for multiple interrupt levels.

Note: Only the read-modify-write (swap/CSRRW) operation is useful for `mscratchcswl`. The behavior of the non-CSRRW variants (i.e. CSRRS/C, CSRRWI, CSRRS/CI) and CSRRW variants with **rd = x0** or **rs1 = x0** on `mscratchcswl` are implementation-defined. CV32E40X will treat such instructions as illegal instructions.

14.2.24 CLIC Base (`mclicbase`)

CSR Address: 0x34A

Note: The address for the `mclicbase` CSR has not been defined yet in [RISC-V-SMCLIC]. The used address is therefore likely to change. Also it is likely that the `mclicbase` CSR will be removed all together.

Reset Value: 0x0000_0000

Include Condition: `SMCLIC` = 1

Detailed:

Bit #	R/W	Description
31:12	R (0x0)	**MCLICBASE**: CLIC Base
11: 0	R (0x0)	Reserved. Hardwired to 0.

CLIC base register.

Note: Currently `mclicbase` CSR is simply hardwired to 0x0 and will therefore likely not reflect the actual CLIC base. This CSR will likely be removed. The [RISC-V-SMCLIC] specification does not specify its address yet and therefore no further attempt is made to further implement this in CV32E40X.

14.2.25 Trigger Select Register (tselect)

CSR Address: 0x7A0

Reset Value: 0x0000_0000

Bit #	R/W	Description
31:0	WARL (0x0 - (DBG_NUM_TRIGGERS-1))	CV32E40X implements 0 to DBG_NUM_TRIGGERS triggers. Selects which trigger CSRs are accessed through the tdata* CSRs.

14.2.26 Trigger Data 1 (tdata1)

CSR Address: 0x7A1

Reset Value: 0x6800_1000

Bit#	R/W	Description
31:28	WARL (0x5, 0x6, 0xF)	**TYPE**. 0x5 (etrigger), 0x6 (mcontrol6), 0xF (disabled).
27	WARL (0x1)	**DMODE**. Only debug mode can write tdata registers.
26:0	WARL	**DATA**. Trigger data depending on type

Note: The WARL behavior of tdata1.DATA depends on the value of tdata1.TYPE as described in *Match Control Type 6 (mcontrol6)*, *Exception Trigger (etrigger)* and *Trigger Data 1 (tdata1) - disabled view*.

14.2.27 Match Control Type 6 (mcontrol6)

CSR Address: 0x7A1 (mcontrol6 is accessible as tdata1 when tdata1.TYPE is 0x6)

Reset Value: Not applicable

Bit#	R/W	Description
31:28	WARL (0x6)	**TYPE**. 6 = Address match trigger.
27	WARL (0x1)	**DMODE**. Only debug mode can write `tdata` registers.
26:25	WARL (0x0)	Hardwired to 0.
24	WARL (0x0)	**VS**. Hardwired to 0.
23	WARL (0x0)	**VU**. Hardwired to 0.
22	WARL (0x0)	**HIT**. Hardwired to 0.
21	WARL (0x0)	**SELECT**. Only address matching is supported.
20	WARL (0x0)	**TIMING**. Break before the instruction at the specified address.
19:16	WARL (0x0)	**SIZE**. Match accesses of any size.
15:12	WARL (0x1)	**ACTION**. Enter debug mode on match.
11	WARL (0x0)	**CHAIN**. Hardwired to 0.
10:7	WARL (0x0*, 0x2, 0x3)	**MATCH**. 0: Address matches *tdata2*, 2: Address is greater than or equal to *tdata2*, 3: Address is less than *tdata2*.
6	WARL	**M**. Match in machine mode.
5	WARL (0x0)	Hardwired to 0.
4	WARL (0x0)	**S**. Hardwired to 0.
3	WARL (0x0)	**U**. Hardwired to 0.
2	WARL	**EXECUTE**. Enable matching on instruction address.
1	WARL	**STORE**. Enable matching on store address.
0	WARL	**LOAD**. Enable matching on load address.

14.2.28 Exception Trigger (`etrigger`)

CSR Address: 0x7A1 (`etrigger` is accessible as `tdata1` when `tdata1.TYPE` is 0x5)

Reset Value: Not applicable

Bit#	R/W	Description
31:28	WARL (0x5)	**TYPE**. 5 = Exception trigger.
27	WARL (0x1)	**DMODE**. Only debug mode can write `tdata` registers.
26	WARL (0x0)	**HIT**. Hardwired to 0.
25:13	WARL (0x0)	Hardwired to 0.
12	WARL (0x0)	**VS**. Hardwired to 0.
11	WARL (0x0)	**VU**. Hardwired to 0.
10	WARL (0x0)	Hardwired to 0.
9	WARL	**M**. Match in machine mode.
8	WARL (0x0)	Hardwired to 0.
7	WARL (0x0)	**S**. Hardwired to 0.
6	WARL (0x0)	**U**. Hardwired to 0.
5:0	WARL (0x1)	**ACTION**. Enter debug mode on match.

14.2.29 Trigger Data 1 (tdata1) - disabled view

CSR Address: 0x7A1 (tdata1 view when tdata1.TYPE is 0xF)

Reset Value: Not applicable

Bit#	R/W	Description
31:28	WARL (0xF)	**TYPE**. 0xF (disabled).
27	WARL (0x1)	**DMODE**. Only debug mode can write tdata registers.
26:0	WARL (0x0)	**DATA**.

14.2.30 Trigger Data Register 2 (tdata2)

CSR Address: 0x7A2

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	**DATA**

Accessible in Debug Mode or M-Mode, depending on **tdata1.dmode**. This register stores the instruction address, load address or store address to match against for a breakpoint trigger or the currently selected exception codes for an exception trigger.

14.2.31 Trigger Data Register 3 (tdata3)

CSR Address: 0x7A3

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	WARL (0x0)	Hardwired to 0.

Accessible in Debug Mode or M-Mode. CV32E40X does not support the features requiring this register. CSR is hardwired to 0.

14.2.32 Trigger Info (tinfo)

CSR Address: 0x7A4

Reset Value: 0x0000_8060

Detailed:

Bit#	R/W	Description
31:16	WARL (0x0)	Hardwired to 0.
15:0	R (0x8060)	**INFO**. Types 0x5, 0x6 and 0xF are supported.

The **info** field contains one bit for each possible *type* enumerated in *tdata1*. Bit N corresponds to type N. If the bit is set, then that type is supported by the currently selected trigger. If the currently selected trigger does not exist, this field contains 1.

Accessible in Debug Mode or M-Mode.

14.2.33 Trigger Control (`tcontrol`)

CSR Address: 0x7A5

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:8	WARL (0x0)	Hardwired to 0.
7	WARL (0x0)	**MPTE**. Hardwired to 0.
6:4	WARL (0x0)	Hardwired to 0.
3	WARL (0x0)	**MTE**. Hardwired to 0.
2:0	WARL (0x0)	Hardwired to 0.

CV32E40X does not support the features requiring this register. CSR is hardwired to 0.

14.2.34 Debug Control and Status (`dcsr`)

CSR Address: 0x7B0

Reset Value: 0x4000_0413

Detailed:

Bit #	R/W	Description
31:28	R (0x4)	**XDEBUGVER**. External debug support exists as described in [RISC-V-DEBUG].
27:18	WARL (0x0)	Reserved
17	WARL (0x0)	**EBREAKVS**. Hardwired to 0
16	WARL (0x0)	**EBREAKVU**. Hardwired to 0.
15	RW	**EBREAKM**. Set to enter debug mode on `ebreak` during machine mode.
14	WARL (0x0)	Hardwired to 0.
13	WARL (0x0)	**EBREAKS**. Hardwired to 0.
12	WARL (0x0)	**EBREAKU**. Hardwired to 0.
11	WARL	**STEPIE**. Set to enable interrupts during single stepping.
10	WARL	**STOPCOUNT**.
9	WARL (0x0)	**STOPTIME**. Hardwired to 0.
8:6	R	**CAUSE**. Return the cause of debug entry.
5	WARL (0x0)	**V**. Hardwired to 0.
4	WARL (0x1)	**MPRVEN**. Hardwired to 1.
3	R	**NMIP**. If set, an NMI is pending
2	RW	**STEP**. Set to enable single stepping.
1:0	WARL (0x3)	**PRV**. Returns the privilege mode before debug entry.

14.2.35 Debug PC (dpc)

CSR Address: 0x7B1

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:0	RW	**DPC**. Debug PC

When the core enters in Debug Mode, DPC contains the virtual address of the next instruction to be executed.

14.2.36 Debug Scratch Register 0/1 (dscratch0/1)

CSR Address: 0x7B2/0x7B3

Reset Value: 0x0000_0000

Detailed:

Bit #	R/W	Description
31:0	RW	DSCRATCH0/1

14.2.37 Machine Cycle Counter (mcycle)

CSR Address: 0xB00

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	The lower 32 bits of the 64 bit machine mode cycle counter.

14.2.38 Machine Instructions-Retired Counter (minstret)

CSR Address: 0xB02

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	The lower 32 bits of the 64 bit machine mode instruction retired counter.

14.2.39 Machine Performance Monitoring Counter (mhpmcounter3 .. mhpmcounter31)

CSR Address: 0xB03 - 0xB1F

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	Machine performance-monitoring counter

The lower 32 bits of the 64 bit machine performance-monitoring counter(s). The number of machine performance-monitoring counters is determined by the parameter NUM_MHPMCOUNTERS with a range from 0 to 29 (default value of 1). Non implemented counters always return a read value of 0.

14.2.40 Upper 32 Machine Cycle Counter (mcycleh)

CSR Address: 0xB80

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	The upper 32 bits of the 64 bit machine mode cycle counter.

14.2.41 Upper 32 Machine Instructions-Retired Counter (minstreth)

CSR Address: 0xB82

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	The upper 32 bits of the 64 bit machine mode instruction retired counter.

14.2.42 Upper 32 Machine Performance Monitoring Counter (mhpmcounter3h .. mhpmcounter31h)

CSR Address: 0xB83 - 0xB9F

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	RW	Machine performance-monitoring counter

The upper 32 bits of the 64 bit machine performance-monitoring counter(s). The number of machine performance-monitoring counters is determined by the parameter NUM_MHPMCOUNTERS with a range from 0 to 29 (default value of 1). Non implemented counters always return a read value of 0.

14.2.43 Machine Vendor ID (mvendorid)

CSR Address: 0xF11

Reset Value: 0x0000_0602

Detailed:

Bit #	R/W	Description
31:7	R (0xC)	Number of continuation codes in JEDEC manufacturer ID.
6:0	R (0x2)	Final byte of JEDEC manufacturer ID, discarding the parity bit.

The `mvendorid` encodes the OpenHW JEDEC Manufacturer ID, which is 2 decimal (bank 13).

14.2.44 Machine Architecture ID (marchid)

CSR Address: 0xF12

Reset Value: 0x0000_0014

Detailed:

Bit #	R/W	Description
31:0	R (0x14)	Machine Architecture ID of CV32E40X is 0x14 (decimal 20)

14.2.45 Machine Implementation ID (mimpid)

CSR Address: 0xF13

Reset Value: Defined

Detailed:

Bit #	R/W	Description
31:20	R (0x0)	Hardwired to 0.
19:16	R (0x0)	**MAJOR**.
15:12	R (0x0)	Hardwired to 0.
11:8	R (0x0)	**MINOR**.
7:4	R (0x0)	Hardwired to 0.
3:0	R	**PATCH**. **mimpid_patch_i**, see *Core Integration*

The Machine Implementation ID uses a Major, Minor, Patch versioning scheme. The **PATCH** bitfield is defined and set by the integrator and shall be set to 0 when no patches are applied. It is made available as **mimpid_patch_i** on the boundary of CV32E40X such that it can easily be changed by a metal layer only change.

14.2.46 Hardware Thread ID (mhartid)

CSR Address: 0xF14

Reset Value: Defined

Bit #	R/W	Description
31:0	R	Machine Hardware Thread ID **mhartid_i**, see *Core Integration*

14.2.47 Machine Configuration Pointer (mconfigptr)

CSR Address: 0xF15

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Definition
31:0	R (0x0)	Reserved

14.2.48 Cycle Counter (cycle)

CSR Address: 0xC00

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the lower 32 bits of the 64 bit machine mode cycle counter.

14.2.49 Time (time)

CSR Address: 0xC01

Reset Value: defined (based on `time_i`)

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the lower 32 bits of the 64 bit time counter. A read of the `time` CSR value returns the value present on the `time_i[31:0]` pins.

14.2.50 Instructions-Retired Counter (`instret`)

CSR Address: 0xC02

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the lower 32 bits of the 64 bit machine mode instruction retired counter.

14.2.51 Performance Monitoring Counter (`hpmcounter3 .. hpmcounter31`)

CSR Address: 0xC03 - 0xC1F

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the lower 32 bits of the 64 bit machine mode performance counter. Non implemented counters always return a read value of 0.

14.2.52 Upper 32 Cycle Counter (`cycleh`)

CSR Address: 0xC80

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the upper 32 bits of the 64 bit machine mode cycle counter.

14.2.53 Upper 32 Time (`timeh`)

CSR Address: 0xC81

Reset Value: defined (based on `time_i`)

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the upper 32 bits of the 64 bit time counter. A read of the `time` CSR value returns the value present on the `time_i[63:32]` pins.

14.2.54 Upper 32 Instructions-Retired Counter (instreth)

CSR Address: 0xC82

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the upper 32 bits of the 64 bit machine mode instruction retired counter.

14.2.55 Upper 32 Performance Monitoring Counter (hpmcounter3h .. hpmcounter31h)

CSR Address: 0xC83 - 0xC9F

Reset Value: 0x0000_0000

Detailed:

Bit#	R/W	Description
31:0	R	

Read-only unprivileged shadow of the upper 32 bits of the 64 bit machine mode performance counter. Non implemented counters always return a read value of 0.

CHAPTER

FIFTEEN

PERFORMANCE COUNTERS

CV32E40X implements performance counters according to [RISC-V-PRIV]. The performance counters are placed inside the Control and Status Registers (CSRs) and can be accessed with the `CSRRW(I)` and `CSRRS/C(I)` instructions.

CV32E40X implements the clock cycle counter `mcycle(h)`, the retired instruction counter `minstret(h)`, as well as the parameterizable number of event counters `mhpmcounter3(h)` - `mhpmcounter31(h)` and the corresponding event selector CSRs `mhpmevent3` - `mhpmevent31`, and the `mcountinhibit` CSR to individually enable/disable the counters. `mcycle(h)` and `minstret(h)` are always available.

All counters are 64 bit wide.

The number of event counters is determined by the parameter `NUM_MHPMCOUNTERS` with a range from 0 to 29 (default value of 1).

Unimplemented counters always read 0.

Note: All performance counters are using the gated version of `clk_i`. The **wfi** instruction impact the gating of `clk_i` as explained in *Sleep Unit* and can therefore affect the counters.

15.1 Event Selector

The following events can be monitored using the performance counters of CV32E40X.

Bit #	Event Name	
0	CYCLES	Number of cycles
1	INSTR	Number of instructions retired
2	COMP_INSTR	Number of compressed instructions retired
3	JUMP	Number of jumps (unconditional)
4	BRANCH	Number of branches (conditional)
5	BRANCH_TAKEN	Number of branches taken (conditional)
6	INTR_TAKEN	Number of taken interrupts (excluding NMI)
7	DATA_READ	Number of read transactions on the OBI data interface.
8	DATA_WRITE	Number of write transactions on the OBI data interface.
9	IF_INVALID	Number of cycles that the IF stage causes ID stage underutilization
10	ID_INVALID	Number of cycles that the ID stage causes EX stage underutilization
11	EX_INVALID	Number of cycles that the EX stage causes WB stage underutilization
12	WB_INVALID	Number of cycles that the WB stage causes register file write port underutilization
13	LD_STALL	Number of stall cycles caused by load use hazards
14	JMP_STALL	Number of stall cycles caused by jump register hazards
15	WB_DATA_STALL	Number of stall cycles caused in the WB stage by loads/stores.

The event selector CSRs mhpmevent3 - mhpmevent31 define which of these events are counted by the event counters mhpmcounter3(h) - mhpmcounter31(h). If a specific bit in an event selector CSR is set to 1, this means that events with this ID are being counted by the counter associated with that selector CSR. If an event selector CSR is 0, this means that the corresponding counter is not counting any event.

Note: At most 1 bit should be set in an event selector. If multiple bits are set in an event selector, then the operation of the associated counter is undefined.

15.2 Controlling the counters from software

By default, all available counters are disabled after reset in order to provide the lowest power consumption.

They can be individually enabled/disabled by overwriting the corresponding bit in the mcountinhibit CSR at address 0x320 as described in [RISC-V-PRIV]. In particular, to enable/disable mcycle(h), bit 0 must be written. For minstret(h), it is bit 2. For event counter mhpmcounterX(h), it is bit X.

The lower 32 bits of all counters can be accessed through the base register, whereas the upper 32 bits are accessed through the h-register. Reads of all these registers are non-destructive.

15.3 Parametrization at synthesis time

The mcycle(h) and minstret(h) counters are always available and 64 bit wide.

The number of available event counters mhpmcounterX(h) can be controlled via the NUM_MHPMCOUNTERS parameter. By default NUM_MHPCOUNTERS set to 1.

An increment of 1 to the NUM_MHPCOUNTERS results in the addition of the following:

- 64 flops for mhpmcounterX
- 15 flops for *mhpmeventX*
- 1 flop for *mcountinhibit[X]*
- Adder and event enablement logic

CHAPTER

SIXTEEN

EXCEPTIONS AND INTERRUPTS

CV32E40X supports one of two interrupt architectures. If the `SMCLIC` parameter is set to 0, then the CLINT mode interrupt architecture is supported (see *CLINT Mode Interrupt Architecture*). If the `SMCLIC` parameter is set to 1, then the CLIC mode interrupt architecture is supported (see *CLIC Mode Interrupt Architecture*).

16.1 Exceptions

CV32E40X can trigger the following exceptions as reported in `mcause`:

Interrupt	Exception Code	Description	Scenario(s)
0	1	Instruction access fault	Execution attempt from I/O region.
0	2	Illegal instruction	
0	3	Breakpoint	Environment break.
0	5	Load access fault	Non-naturally aligned load access attempt to an I/O region. Modified load access attempt to an I/O region. Load-Reserved attempt to region without atomic support.
0	7	Store/AMO access fault	Non-naturally aligned store access attempt to an I/O region. Modified store access attempt to an I/O region. Store-Conditional or Atomic Memory Operation (AMO) attempt to region without atomic support.
0	11	Environment call from M-Mode (ECALL)	
0	24	Instruction bus fault	`instr_err_i` = 1 and `instr_rvalid_i` = 1 for instruction fetch

If an instruction raises multiple exceptions, the priority, from high to low, is as follows:

- `instruction access fault (1)`
- `instruction bus fault (24)`
- `illegal instruction (2)`
- `environment call from M-Mode (11)`
- `environment break (3)`
- `store/AMO access fault (7)`

- load access fault (5)

Exceptions in general cannot be disabled and are always active. All exceptions are precise. Whether the PMA will actually cause exceptions depends on its configuration. CV32E40X raises an illegal instruction exception for any instruction in the RISC-V privileged and unprivileged specifications that is explicitly defined as being illegal according to the ISA implemented by the core, as well as for any instruction that is left undefined in these specifications unless the instruction encoding is configured as a custom CV32E40X instruction for specific parameter settings as defined in (see *CORE-V Instruction Set Extensions*). An instruction bus error leads to a precise instruction interface bus fault if an attempt is made to execute the instruction that has an associated bus error. Similarly an instruction fetch with a failing PMA check only leads to an instruction access exception if an actual execution attempt is made for it.

16.2 Non Maskable Interrupts

Non Maskable Interrupts (NMIs) update `mepc`, `mcause` and `mstatus` similar to regular interrupts. However, as the faults that result in NMIs are imprecise, the contents of `mepc` is not guaranteed to point to the instruction after the faulted load or store.

Note: Specifically `mstatus.mie` will get cleared to 0 when an (unrecoverable) NMI is taken. [RISC-V-PRIV] does not specify the behavior of `mstatus` in response to NMIs, see https://github.com/riscv/riscv-isa-manual/issues/756. If this behavior is specified at a future date, then we will reconsider our implementation.

NMIs have higher priority than other interrupts for both the CLINT mode interrupt architecture and the CLIC mode interrupt architecture.

If `SMCLIC` == 0, then the NMI vector location is as follows:

- Upon an NMI in non-vectored CLINT mode the core jumps to **mtvec[31:7]**, 5'h0, 2'b00} (i.e. index 0).
- Upon an NMI in vectored CLINT mode the core jumps to **mtvec[31:7]**, 5'hF, 2'b00} (i.e. index 15).

If `SMCLIC` == 1, then the NMI vector location is as follows:

- Upon an NMI in CLIC mode the core jumps to **mtvec[31:7]**, 5'h0, 2'b00} (i.e. index 0).

Note: For NMIs the exception codes in the `mcause` CSR do not match the table index as for regular interrupts.

An NMI will occur when a load or store instruction experiences a bus fault. The fault resulting in an NMI is handled in an imprecise manner, meaning that the instruction that causes the fault is allowed to retire and the associated NMI is taken afterwards. NMIs are never masked by the `MIE` bit. NMIs are masked however while in debug mode or while single stepping with `STEPIE` = 0 in the `dcsr` CSR. This means that many instructions may retire before the NMI is visible to the core if debugging is taking place. Once the NMI is visible to the core, at most two instructions will retire before the NMI is taken.

If an NMI becomes pending while in debug mode as described above, the NMI will be taken immediately after debug mode has been exited.

In case of bufferable stores, the NMI is allowed to become visible an arbitrary time after the instruction retirement. As for the case with debugging, this can cause several instructions to retire before the NMI becomes visible to the core.

When a data bus fault occurs, the first detected fault will be latched and used for `mcause` when the NMI is taken. Any new data bus faults occuring while an NMI is pending will be discarded. When the NMI handler is entered, new data bus faults may be latched.

While an NMI is pending, `DCSR.nmip` will be 1. Note that this CSR is only accessible from debug mode, and is thus not visible for machine mode code.

16.3 CLINT Mode Interrupt Architecture

If SMCLIC == 0, then CV32E40X supports the CLINT mode interrupt architecture as defined in [RISC-V-PRIV]. In this configuration only the CLINT mode interrupt handling modes (non-vectored CLINT mode and vectored CLINT mode) can be used. The irq_i[31:16] interrupts are a custom extension that can be used with the CLINT mode interrupt architecture.

When entering an interrupt/exception handler, the core sets the mepc CSR to the current program counter and saves mstatus.MIE to mstatus.MPIE. All exceptions cause the core to jump to the base address of the vector table in the mtvec CSR. Interrupts are handled in either non-vectored CLINT mode or vectored CLINT mode depending on the value of mtvec.MODE. In non-vectored CLINT mode the core jumps to the base address of the vector table in the mtvec CSR. In vectored CLINT mode the core jumps to the base address plus four times the interrupt ID. Upon executing an MRET instruction, the core jumps to the program counter previously saved in the mepc CSR and restores mstatus.MPIE to mstatus.MIE.

The base address of the vector table must be aligned to 128 bytes and can be programmed by writing to the mtvec CSR (see *Machine Trap-Vector Base Address (mtvec) - SMCLIC == 0*).

16.3.1 Interrupt Interface

Table 16.1 describes the interrupt interface used for the CLINT mode interrupt architecture.

Table 16.1: CLINT mode interrupt architecture interface signals

Signal	Direction	Description
irq_i[31:16]	input	Active high, level sensistive interrupt inputs. Custom extension.
irq_i[15:12]	input	Reserved. Tie to 0.
irq_i[11]	input	Active high, level sensistive interrupt input. Referred to as Machine External Interrupt (MEI), but integrator can assign a different purpose if desired.
irq_i[10:8]	input	Reserved. Tie to 0.
irq_i[7]	input	Active high, level sensistive interrupt input. Referred to as Machine Timer Interrupt (MTI), but integrator can assign a different purpose if desired.
irq_i[6:4]	input	Reserved. Tie to 0.
irq_i[3]	input	Active high, level sensistive interrupt input. Referred to as Machine Software Interrupt (MSI), but integrator can assign a different purpose if desired.
irq_i[2:0]	input	Reserved. Tie to 0.

Note: The clic_*_i pins are ignored in CLINT mode and should be tied to 0.

16.3.2 Interrupts

The irq_i[31:0] interrupts are controlled via the mstatus, mie and mip CSRs. CV32E40X uses the upper 16 bits of mie and mip for custom interrupts (irq_i[31:16]), which reflects an intended custom extension in the RISC-V CLINT mode interrupt architecture. After reset, all interrupts, except for NMIs, are disabled. To enable any of the irq_i[31:0] interrupts, both the global interrupt enable (MIE) bit in the mstatus CSR and the corresponding individual interrupt enable bit in the mie CSR need to be set. For more information, see the *Control and Status Registers* documentation.

If multiple interrupts are pending, they are handled in the fixed priority order defined by [RISC-V-PRIV]. The highest priority is given to the interrupt with the highest ID, except for the Machine Timer Interrupt, which has the lowest

priority. So from high to low priority the interrupts are ordered as follows:

- store bus fault NMI (1025)
- load bus fault NMI (1024)
- irq_i[31]
- irq_i[30]
- …
- irq_i[16]
- irq_i[11]
- irq_i[3]
- irq_i[7]

The irq_i[31:0] interrupt lines are level-sensitive. The NMIs are triggered by load/store bus fault events. To clear the irq_i[31:0] interrupts at the external source, CV32E40X relies on a software-based mechanism in which the interrupt handler signals completion of the handling routine to the interrupt source, e.g., through a memory-mapped register, which then deasserts the corresponding interrupt line.

In Debug Mode, all interrupts are ignored independent of mstatus.MIE and the content of the mie CSR.

CV32E40X can trigger the following interrupts as reported in mcause:

Interrupt	Exception Code	Description	Scenario(s)
1	3	Machine Software Interrupt (MSI)	irq_i[3]
1	7	Machine Timer Interrupt (MTI)	irq_i[7]
1	11	Machine External Interrupt (MEI)	irq_i[11]
1	31-16	Machine Fast Interrupts	irq_i[31]-irq_i[16]
1	1024	Load bus fault NMI (imprecise)	data_err_i = 1 and data_rvalid_i = 1 for load
1	1025	Store bus fault NMI (imprecise)	data_err_i = 1 and data_rvalid_i = 1 for store

Note: Load bus fault and store bus fault are handled as imprecise non-maskable interrupts (as opposed to precise exceptions).

Note: The NMI vector location is at index 15 of the machine trap vector table for both non-vectored CLINT mode and vectored CLINT mode (i.e. at {**mtvec[31:7]**, 5'hF, 2'b00}). The NMI vector location therefore does **not** match its exception code.

16.3.3 Nested Interrupt Handling

Within the CLINT mode interrupt architecture there is no hardware support for nested interrupt handling. Nested interrupt handling can however still be supported via software.

The hardware automatically disables interrupts upon entering an interrupt/exception handler. Otherwise, interrupts during the critical part of the handler, i.e. before software has saved the `mepc` and `mstatus` CSRs, would cause those CSRs to be overwritten. If desired, software can explicitly enable interrupts by setting `mstatus`.MIE to 1 from within the handler. However, software should only do this after saving `mepc` and `mstatus`. There is no limit on the maximum number of nested interrupts. Note that, after enabling interrupts by setting `mstatus`.MIE to 1, the current handler will be interrupted also by lower priority interrupts. To allow higher priority interrupts only, the handler must configure `mie` accordingly.

16.4 CLIC Mode Interrupt Architecture

If `SMCLIC` == 1, then CV32E40X supports the Core-Local Interrupt Controller (CLIC) Privileged Architecture Extension defined in [RISC-V-SMCLIC]. In this configuration only the CLIC interrupt handling mode can be used (i.e. `mtvec[1:0]` = 0x3).

The CLIC implementation is split into a part internal to the core (containing CSRs and related logic) and a part external to the core (containing memory mapped registers and arbitration logic). CV32E40X only provides the core internal part of CLIC. The external part can be added on the interface described in *Interrupt Interface*. CLIC provides low-latency, vectored, pre-emptive interrupts.

16.4.1 Interrupt Interface

Table 16.2 describes the interrupt interface used for the CLIC interrupt architecture.

Table 16.2: CLIC mode interrupt architecture interface signals

Signal	Direction	Description
`clic_irq_i`	input	Is there any pending-and-enabled interrupt?
`clic_irq_id_i[SMCLIC_ID_WIDTH-1:0]`	input	Index of the most urgent pending-and-enabled interrupt.
`clic_irq_level_i[7:0]`	input	Interrupt level of the most urgent pending-and-enabled interrupt.
`clic_irq_priv_i[1:0]`	input	Associated privilege mode of the most urgent pending-and-enabled interrupt. Only machine-mode interrupts are supported.
`clic_irq_shv_i`	input	Selective hardware vectoring enabled for the most urgent pending-and-enabled interrupt?

The term *pending-and-enabled* interrupt in above table refers to *pending-and-locally-enabled*, i.e. based on the `CLICINTIP` and `CLICINTIE` memory mapped registers from [RISC-V-SMCLIC].

Note: Edge triggered interrupts are not supported.

Note: `clic_irq_shv_i` shall be 0 if `cliccfg.nvbits` of the externl CLIC module is 0.

Note: `clic_irq_priv_i[1:0]` shall be tied to 2'b11 (machine).

Note: The `irq_i[31:0]` pins are ignored in CLIC mode and should be tied to 0.

16.4.2 Interrupts

Although the [RISC-V-SMCLIC] specification supports up to 4096 interrupts, CV32E40X itself supports at most 1024 interrupts. The maximum number of supported CLIC interrupts is equal to `2^SMCLIC_ID_WIDTH`, which can range from 2 to 1024. The `SMCLIC_ID_WIDTH` parameter also impacts the alignment requirement for the trap vector table, see *Machine Trap Vector Table Base Address (mtvt)*.

Interrupt prioritization is mostly performed in the part of CLIC that is external to the core, with the exception that CV32E40X prioritizes all NMIs above interrupts received via `clic_irq_i`.

16.4.3 Nested Interrupt Handling

CV32E40X offers hardware support for nested interrupt handling when `SMCLIC` == 1.

CLIC extends interrupt preemption to support up to 256 interrupt levels for each privilege mode, where higher-numbered interrupt levels can preempt lower-numbered interrupt levels. See [RISC-V-SMCLIC] for details.

CHAPTER

SEVENTEEN

DEBUG & TRIGGER

CV32E40X offers support for execution-based debug according to [RISC-V-DEBUG]. The main requirements for the core are described in Chapter 4: RISC-V Debug, Chapter 5: Trigger Module, and Appendix A.2: Execution Based.

Note: As execution based debug is used, the Debug Module (with code entry points defined by `dm_halt_addr_i` and `dm_exception_addr_i`) needs to be located in a memory region that supports code execution. This therefore (at least) requires that the related memory region is marked as Main in the PMA (*Physical Memory Attribution (PMA)*), which is the default behavior if the PMA is deconfigured.

The following list shows the simplified overview of events that occur in the core when debug is requested:

1. Enters Debug Mode
2. Saves the PC to `dpc`
3. Updates the cause in `dcsr`
4. Points the PC to the location determined by the input port `dm_haltaddr_i`
5. Begins executing debug control code

Debug Mode can be entered by one of the following conditions:

- External debug event using the `debug_req_i` signal
- Trigger Module match event with `tdata1.action` set to 1
- `ebreak` instruction in machine mode when `dcsr.EBREAKM` == 1 (see *EBREAK Behavior* below)

A user wishing to perform an abstract access, whereby the user can observe or control a core's GPR or CSR register from the hart, is done by invoking debug control code to move values to and from internal registers to an externally addressable Debug Module (DM). Using this execution-based debug allows for the reduction of the overall number of debug interface signals.

Note: Debug support in CV32E40X is only one of the components needed to build a System on Chip design with run-control debug support (think "the ability to attach GDB to a core over JTAG"). Additionally, a Debug Module and a Debug Transport Module, compliant with [RISC-V-DEBUG], are needed.

A supported open source implementation of these building blocks can be found in the RISC-V Debug Support for PULP Cores IP block.

The CV32E40X also supports a Trigger Module to enable entry into Debug Mode on a trigger event with the following features:

- Number of trigger register(s): Parametrizable number of triggers using parameter `DBG_NUM_TRIGGERS`.

- Supported trigger types: Execute/load/store address match (Match Control) and exception trigger.

The compare value used to determine an execute address match is the PC of the instruction, i.e. only the lowest virtual address of the instruction is used. The compare value(s) used to determine a load/store address match depend(s) on the size of the transferred data item as well as the lowest virtual address of the access. A byte load/store for address `A` only uses `A` as compare value; a halfword load/store for address `A` uses `A` and `A+1` as compare values; a word load/store for address `A` uses `A`, `A+1`, `A+2` and `A+3` as compare values.

A trigger match will cause debug entry if `tdata1.ACTION` is 1.

Note: Hardware triggers and breakpoints are not supported for the table fetch used in table jump instructions and CLIC hardware vectored interrupts.

The CV32E40X will not support the optional debug features 10, 11, & 12 listed in Section 4.1 of [RISC-V-DEBUG]. Specifically, a control transfer instruction's destination location being in or out of the Program Buffer and instructions depending on PC value shall **not** cause an illegal instruction.

CV32E40X prioritizes debug mode entry below NMIs, but above regular interrupts and synchronous exceptions.

17.1 Interface

Signal	Direction	Description
`debug_req_i`	input	Request to enter Debug Mode
`debug_havereset_o`	output	Debug status: Core has been reset
`debug_running_o`	output	Debug status: Core is running
`debug_halted_o`	output	Debug status: Core is halted
`debug_pc_valid_o`	output	Valid signal for `debug_pc_o`
`debug_pc_o`	output	PC of last retired instruction
`dm_halt_addr_i[31:0]`	input	Address for debugger entry
`dm_exception_addr_i[31:0]`	input	Address for debugger exception entry

`debug_req_i` is the "debug interrupt", issued by the debug module when the core should enter Debug Mode. The `debug_req_i` is synchronous to `clk_i` and requires a minimum assertion of one clock period to enter Debug Mode. The instruction being decoded during the same cycle that `debug_req_i` is first asserted shall not be executed before entering Debug Mode.

`debug_havereset_o`, `debug_running_o`, and `debug_mode_o` signals provide the operational status of the core to the debug module. The assertion of these signals is mutually exclusive.

`debug_havereset_o` is used to signal that the CV32E40X has been reset. `debug_havereset_o` is set high during the assertion of `rst_ni`. It will be cleared low a few (unspecified) cycles after `rst_ni` has been deasserted **and** `fetch_enable_i` has been sampled high.

`debug_running_o` is used to signal that the CV32E40X is running normally.

`debug_halted_o` is used to signal that the CV32E40X is in debug mode.

`debug_pc_o` is the PC of the last retired instruction. This signal is only valid when `debug_pc_valid_o` = 1.

`dm_halt_addr_i` is the address where the PC jumps to for a debug entry event. When in Debug Mode, an ebreak instruction will also cause the PC to jump back to this address without affecting status registers. (see *EBREAK Behavior* below)

dm_exception_addr_i is the address where the PC jumps to when an exception occurs during Debug Mode. When in Debug Mode, the mret and ecall instructions will also cause the PC to jump back to this address without affecting status registers.

Both dm_halt_addr_i and dm_exception_addr_i must be word aligned.

17.2 Core Debug Registers

CV32E40X implements four core debug registers, namely *Debug Control and Status (dcsr)*, *Debug PC (dpc)*, and two debug scratch registers. Access to these registers in non Debug Mode results in an illegal instruction.

The trigger related CSRs (tselect, tdata1, tdata2, tdata3, tinfo, tcontrol) are only included if DBG_NUM_TRIGGERS is set to a value greater than 0. Further descriptions of these CSRs can be found in *Trigger Select Register (tselect)*, *Trigger Data 1 (tdata1)*, *Trigger Data Register 2 (tdata2)*, *Trigger Data Register 3 (tdata3)*, *Trigger Info (tinfo)*, *Trigger Control (tcontrol)* and [RISC-V-DEBUG]. The optional mcontext and mscontext CSRs are not implemented.

If DBG_NUM_TRIGGERS is 0, access to the trigger registers will result in an illegal instruction exception.

The tdata1.DMODE bitfield controls write access permission to the currently selected triggers tdata* registers. In CV32E40X this bit is tied to 1, and thus only debug mode is able to write to the trigger registers.

17.3 Debug state

As specified in RISC-V Debug Specification ([RISC-V-DEBUG]) every hart that can be selected by the Debug Module is in exactly one of four states: nonexistent, unavailable, running or halted.

The remainder of this section assumes that the CV32E40X will not be classified as nonexistent by the integrator.

The CV32E40X signals to the Debug Module whether it is running or halted via its debug_running_o and debug_halted_o pins respectively. Therefore, assuming that this core will not be integrated as a nonexistent core, the CV32E40X is classified as unavailable when neither debug_running_o or debug_halted_o is asserted. Upon rst_ni assertion the debug state will be unavailable until some cycle(s) after rst_ni has been deasserted and fetch_enable_i has been sampled high. After this point (until a next reset assertion) the core will transition between having its debug_halted_o or debug_running_o pin asserted depending whether the core is in debug mode or not. Exactly one of the debug_havereset_o, debug_running_o, debug_halted_o is asserted at all times.

Figure 17.1 and show Figure 17.2 show typical examples of transitioning into the running and halted states.

The key properties of the debug states are:

- The CV32E40X can remain in its unavailable state for an arbitrarily long time (depending on rst_ni and fetch_enable_i).
- If debug_req_i is asserted after rst_ni deassertion and before or coincident with the assertion of fetch_enable_i, then the CV32E40X is guaranteed to transition straight from its unavailable state into its halted state. If debug_req_i is asserted at a later point in time, then the CV32E40X might transition through the running state on its ways to the halted state.
- If debug_req_i is asserted during the running state, the core will eventually transition into the halted state (typically after a couple of cycles).

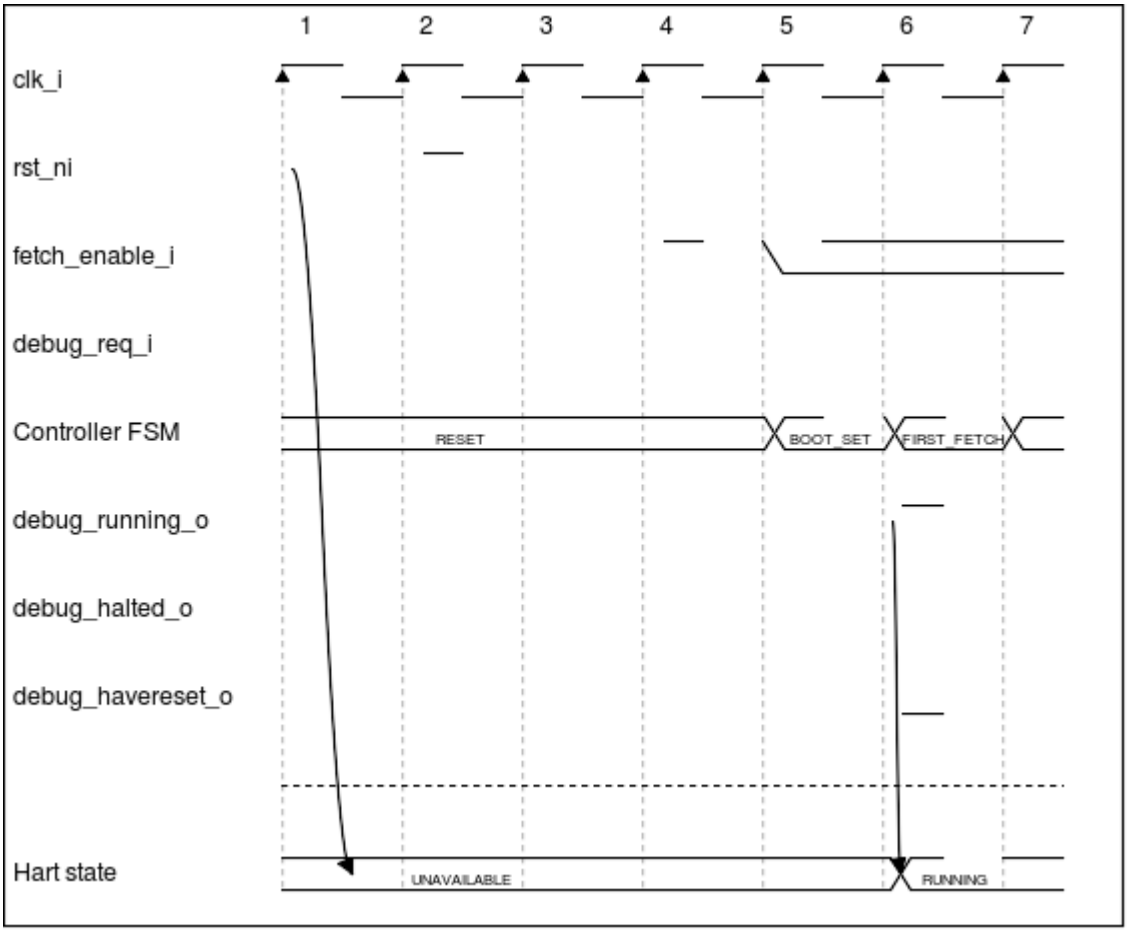


Figure 17.1: Transition into debug `running` state

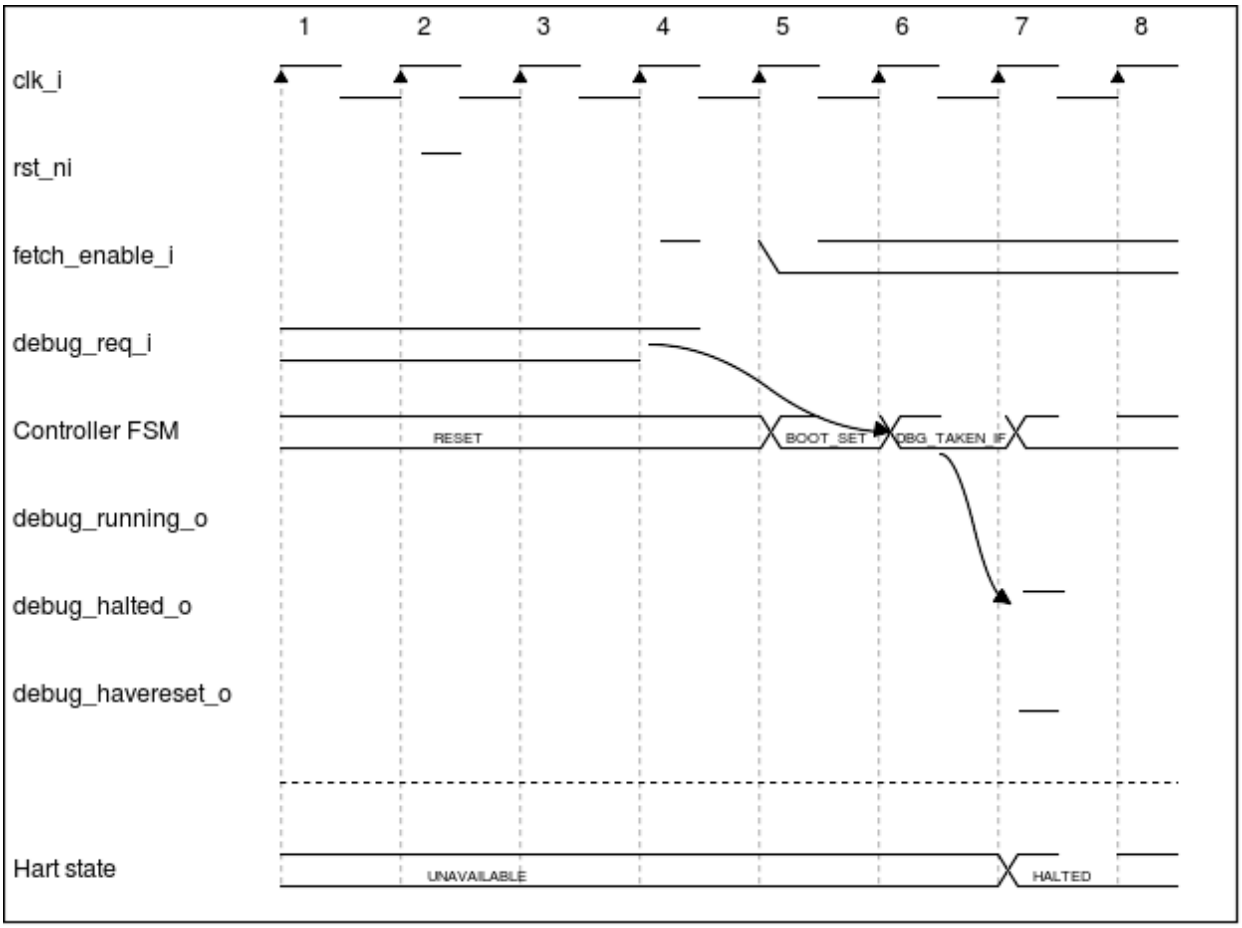


Figure 17.2: Transition into debug `halted` state

17.4 EBREAK Behavior

The `ebreak` instruction description is distributed across several RISC-V specifications: [RISC-V-DEBUG], [RISC-V-PRIV], [RISC-V-UNPRIV]. The following is a summary of the behavior for three common scenarios.

17.4.1 Scenario 1 : Enter Exception

Executing the `ebreak` instruction in machine mode when the core is **not** in Debug Mode and `dcsr.EBREAKM` == 0 shall result in the following actions:

- The core enters the exception handler routine located at `mtvec` (Debug Mode is not entered)
- `mepc` and `mcause` are updated

To properly return from the exception, the ebreak handler will need to increment the `mepc` to the next instruction. This requires querying the size of the `ebreak` instruction that was used to enter the exception (16 bit `c.ebreak` or 32 bit `ebreak`).

Note: CV32E40X does not support `mtval` CSR register which would have saved the value of the instruction for exceptions.

17.4.2 Scenario 2 : Enter Debug Mode

Executing the `ebreak` instruction in machine mode when the core is **not** in Debug Mode and `dcsr.EBREAKM` == 1 shall result in the following actions:

- The core enters Debug Mode and starts executing debug code located at `dm_halt_addr_i` (exception routine not called)
- `dpc` and `dcsr` are updated

Similar to the exception scenario above, the debugger will need to increment the `dpc` to the next instruction before returning from Debug Mode.

Note: The default value of `dcsr.EBREAKM` is 0 and the `dcsr` is only accessible in Debug Mode. To enter Debug Mode from `ebreak`, the user will first need to enter Debug Mode through some other means, such as from the external `debug_req_i`, and set `dcsr.EBREAKM`.

17.4.3 Scenario 3 : Exit Program Buffer & Restart Debug Code

Executing the `ebreak` instruction when the core is in Debug Mode shall result in the following actions:

- The core remains in Debug Mode and execution jumps back to the beginning of the debug code located at `dm_halt_addr_i`
- None of the CSRs are modified

CHAPTER

EIGHTEEN

RISC-V FORMAL INTERFACE

Note: A bindable RISC-V Formal Interface (RVFI) interface will be provided for CV32E40X. See [SYMBIOTIC-RVFI] for details on RVFI.

The module `cv32e40x_rvfi` can be used to create a log of the executed instructions. It is a behavioral, non-synthesizable, module that can be bound to the `cv32e40x_core`.

RVFI serves the following purposes:

- It can be used for formal verification.
- It can be used to produce an instruction trace during simulation.
- It can be used as a monitor to ease interfacing with an external scoreboard that itself can be interfaced to an Instruction Set Simulator (ISS) for verification reasons.

18.1 New Additions

Debug Signals

```
output [NRET * 3 - 1 : 0] rvfi_dbg
output [NRET     - 1 : 0] rvfi_dbg_mode
```

Debug entry is seen by RVFI as happening between instructions. This means that neither the last instruction before debug entry nor the first instruction of the debug handler will signal any direct side-effects. The first instruction of the handler will however show the resulting state caused by these side-effects (e.g. the CSR `rmask`/`rdata` signals will show the updated values, `pc_rdata` will be at the debug handler address, etc.).

For the first instruction after entering debug, the `rvfi_dbg` signal contains the debug cause (see table below). The signal is otherwise 0. The `rvfi_dbg_mode` signal is high if the instruction was executed in debug mode and low otherwise.

Table 18.1: Debug Causes

Cause	Value
None	0x0
Ebreak	0x1
Trigger Match	0x2
External Request	0x3
Single Step	0x4

Note: `rvfi_dbg` will not always match `dcsr.CAUSE` because an `ebreak` in debug mode will be reported via `rvfi_dbg`, whereas `dcsr.CAUSE` will remain unchanged for that case.

NMI signals

```
output [1:0] rvfi_nmip
```

Whenever CV32E40X has a pending NMI, the `rvfi_nmip` will signal this. `rvfi_nmip[0]` will be 1 whenever an NMI is pending, while `rvfi_nmip[1]` will be 0 for loads and 1 for stores.

18.2 Compatibility

This chapter specifies interpretations and compatibilities to the [SYMBIOTIC-RVFI].

Interface Qualification

All RVFI output signals are qualified with the `rvfi_valid` signal. Any RVFI operation (retired or trapped instruction) will set `rvfi_valid` high and increment the `rvfi_order` field. When `rvfi_valid` is low, all other RVFI outputs can be driven to arbitrary values.

Trap Signal

The trap signal indicates that a synchronous trap has ocurred and side-effects can be expected.

```
output rvfi_trap_t[NRET - 1 : 0] rvfi_trap
```

Where the rvfi_trap_t struct contains the following fields:

Table 18.2: RVFI trap type

Field	Type	Bits
trap	logic	[0]
exception	logic	[1]
debug	logic	[2]
exception_cause	logic [5:0]	[8:3]
debug_cause	logic [2:0]	[11:9]
cause_type	logic [1:0]	[13:12]

`rvfi_trap` consists of 14 bits. `rvfi_trap.trap` is asserted if an instruction causes an exception or debug entry. `rvfi_trap.exception` is set for synchronous traps that do not cause debug entry. `rvfi_trap.debug` is set for synchronous traps that do cause debug mode entry. `rvfi_trap.exception_cause` provide information about non-debug traps, while `rvfi_trap.debug_cause` provide information about traps causing entry to debug mode. `rvfi_trap.cause_type` differentiates between fault causes that map to the same exception code in `rvfi_trap.exception_cause` and `rvfi_trap.debug_cause`. When an exception is caused by a single stepped instruction, both `rvfi_trap.exception` and `rvfi_trap.debug` will be set. When `rvfi_trap` signals a trap, CSR side effects and a jump to a trap/debug handler in the next cycle can be expected. The different trap scenarios, their expected side-effects and trap signalling are listed in the table below:

Table 18.3: Table of synchronous trap types

Scenario	Trap Type	rvfi_trap						CSRs updated	Description
		trap	exception	debug	exception_cause	debug_cause	cause		
Instruction Access Fault	Exception	1	1	X	0x01	X	0x0	mcause, mepc	PMA detects instruction execution from non-executable memory.
Illegal Instruction	Exception	1	1	X	0x02	X	0x0	mcause, mepc	Illegal instruction decode.
Breakpoint	Exception	1	1	X	0x03	X	0x0	mcause, mepc	EBREAK executed with dcsr.ebreakm = 0.
Load Access Fault	Exception	1	1	X	0x05	X	0x0	mcause, mepc	Non-naturally aligned load access attempt to an I/O region.
							0x1	mcause, mepc	Load-Reserved attempt to region without atomic support.
Store/AMO Access Fault	Exception	1	1	X	0x07	X	0x0	mcause, mepc	Non-naturally aligned store access attempt to an I/O region.
							0x1	mcause, mepc	SC or AMO attempt to region without atomic support.
Environment Call	Exception	1	1	X	0x0B	X	0x0	mcause, mepc	ECALL executed from Machine mode.
Instruction Bus Fault	Exception	1	1	X	0x24	X	0x0	mcause, mepc	OBI bus error on instruction fetch.
Breakpoint to debug	Debug	1	0	1	X	0x1	0x0	dpc, dcsr	EBREAK from non-debug mode executed with dcsr.ebreakm == 1.
Breakpoint in debug	Debug	1	0	1	X	0x1	0x0	No CSRs updated	EBREAK in debug mode jumps to debug handler.
Debug Trigger Match	Debug	1	0	1	X	0x2	0x0	dpc, dcsr	Debug trigger address match with mcontrol.timing = 0.
Single step	Debug	1	X	1	X	0x4	X	dpc, dcsr	Single step.

Interrupts

Interrupts are seen by RVFI as happening between instructions. This means that neither the last instruction before the interrupt nor the first instruction of the interrupt handler will signal any direct side-effects. The first instruction of the handler will however show the resulting state caused by these side-effects (e.g. the CSR rmask/rdata signals will show the updated values, pc_rdata will be at the interrupt handler address etc.).

```
output rvfi_intr_t[NRET - 1 : 0] rvfi_intr
```

Where the rvfi_intr_t struct contains the following fields:

Table 18.4: RVFI intr type

Field	Type	Bits
intr	logic	[0]
exception	logic	[1]
interrupt	logic	[2]
cause	logic [10:0]	[13:3]

`rvfi_intr` consists of 14 bits. `rvfi_intr.intr` is set for the first instruction of the trap handler when encountering an exception or interrupt. `rvfi_intr.exception` indicates it was caused by synchronous trap and `rvfi_intr.interrupt` indicates it was caused by an interrupt. `rvfi_intr.cause` signals the cause for entering the trap handler.

Table 18.5: Table of scenarios for first instruction of exception/interrupt/debug handler

Scenario	rvfi_intr				rvfi_dbg	rvfi_dbg_cause[8:6] (cause)	
	intr	exception	interrupt	cause			
Synchronous trap	1	1	0	Sync trap cause	0x0	0	
Interrupt (includes NMIs from bus errors)	1	0	1	Interrupt cause	0x0	1	
Debug entry due to EBREAK (from non-debug mode)	0	0	0	0x0	0x1		0x1
Debug entry due to EBREAK (from debug mode)	0	0	0	0x0	0x1		
Debug entry due to trigger match	0	0	0	0x0	0x2		0x2
Debug entry due to external debug request	X	X	X	X	0x3 or 0x5	X	0x3 or 0x5
Debug handler entry due to single step	X	X	X	X	0x4	X	0x4

Note: In above table the - symbol indicates an unchanged value. The `X` symbol indicates that multiple values are possible.

Note: `rvfi_intr` is not set for debug traps unless a debug entry happens during the first instruction of a trap handler (see `rvfi_intr == X` in the table above). In this case CSR side-effects (to `mepc` and `mcause`) can be expected as well.

Program Counter

The `pc_wdata` signal shows the predicted next program counter. This prediction ignores asynchronous traps (asynchronous debug requests and interrupts) and single step debug requests that may have happened at the same time as the instruction.

Memory Access

For CV32E40X, the `rvfi_mem` interface has been expanded to support multiple memory operations per instruction. The new format of the rvfi_mem signals can be seen in the code block below.

```
output [NRET * NMEM * XLEN - 1 : 0]   rvfi_mem_addr
output [NRET * NMEM * XLEN/8 - 1 : 0] rvfi_mem_rmask
```

(continues on next page)

(continued from previous page)

```
output [NRET * NMEM * XLEN/8 - 1 : 0] rvfi_mem_wmask
output [NRET * NMEM * XLEN - 1 : 0]   rvfi_mem_rdata
output [NRET * NMEM * XLEN - 1 : 0]   rvfi_mem_wdata
```

Instructions will populate the `rvfi_mem` outputs with incrementing `NMEM`, starting at `NMEM=1`.

Instructions with a single memory operation (e.g. all RV32I instructions), including split misaligned transfers, will only use NMEM = 1. Instructions with multiple memory operations (e.g. the push and pop instructions from Zcmp) use NMEM > 1 in case multiple memory operations actually occur.

For cores as CV32E40X that support misaligned access `rvfi_mem_addr` will not always be 4 byte aligned. For misaligned accesses the start address of the transfer is reported (i.e. the start address of the first sub-transfer).

CSR Signals

To reduce the number of signals in the RVFI interface, a vectorized CSR interface has been introduced for register ranges.

```
output [<NUM_CSRNAME>-1:0] [NRET * XLEN - 1 : 0] rvfi_csr_<csrname>_rmask
output [<NUM_CSRNAME>-1:0] [NRET * XLEN - 1 : 0] rvfi_csr_<csrname>_wmask
output [<NUM_CSRNAME>-1:0] [NRET * XLEN - 1 : 0] rvfi_csr_<csrname>_rdata
output [<NUM_CSRNAME>-1:0] [NRET * XLEN - 1 : 0] rvfi_csr_<csrname>_wdata
```

Example:

```
output [31:0] [31:0] rvfi_csr_name_rmask
output [31:0] [31:0] rvfi_csr_name_wmask
output [31:0] [31:0] rvfi_csr_name_rdata
output [31:0] [31:0] rvfi_csr_name_wdata
```

Instead of:

```
output [31:0] rvfi_csr_name0_rmask
output [31:0] rvfi_csr_name0_wmask
output [31:0] rvfi_csr_name0_rdata
output [31:0] rvfi_csr_name0_wdata
. . .
output [31:0] rvfi_csr_name31_rmask
output [31:0] rvfi_csr_name31_wmask
output [31:0] rvfi_csr_name31_rdata
output [31:0] rvfi_csr_name31_wdata
```

CSR mnxti

CSR accesses to the `mnxti` CSR do a read-modify-write on the `mstatus` CSR, and return a pointer address if there is a pending non-SHV CLIC interrupt. If there is a pending non-SHV CLIC interrupt, it also updates `mintstatus` and `mcause`. To reflect this behavior, the `rvfi_csr_mnxti*` outputs for `mnxti` have a different semantic than other CSRs.

The `rvfi_csr_mnxti*` is reported as follows on RVFI:

- The `rmask` will always be all ones as for other CSRs.
- The `wmask` will be all ones whenever the CSR instruction actually writes to `mstatus`.
- The `wdata` will be the data written to `mstatus`.
- The `rdata` will report a pointer address if an interrupt is pending, or 0 if no interrupt is pending.

Note that the rvfi_csr_mstatus* will also reflect the access to mstatus due to an mnxti access. In case the access to mnxti returns a valid pointer address, the rvfi_csr_mintstatus* and rvfi_csr_mcause* will also have values showing the side effects of accessing mnxti.

GPR signals

For CV32E40X, RVFI has been expanded to allow reporting multiple register file operations per instruction (more than two reads and one write). The interface is defined as follows:

```
output [NRET * 32 * XLEN - 1 : 0] rvfi_gpr_rdata
output [NRET * 32 -1 : 0]         rvfi_gpr_rmask
output [NRET * 32 * XLEN - 1 : 0] rvfi_gpr_wdata
output [NRET * 32 -1 : 0]         rvfi_gpr_wmask
```

The outputs rvfi_gpr_rdata and rvfi_gpr_wdata reflect the entire register file, with each XLEN field of the vector representing one GPR, with [x0] starting at index [XLEN - 1 : 0], [x1] at index [2*XLEN-1 -: XLEN] and so on. Each bit in the outputs rvfi_gpr_rmask and rvfi_gpr_wmask indicates if a GPR has been read or written during an instruction. The index of the bit indicates the address of the GPR accessed. Entries in rvfi_gpr_rdata and rvfi_gpr_wdata are only considered valid if the corresponding bit in the rvfi_gpr_rmask or rvfi_gpr_wmask is set.

Machine Counter/Timers

In contrast to [SYMBIOTIC-RVFI], the **mcycle[h]** and **minstret[h]** registers are not modelled as happening "between instructions" but rather as a side-effect of the instruction. This means that an instruction that causes an increment (or decrement) of these counters will set the rvfi_csr_mcycle_wmask, and that rvfi_csr_mcycle_rdata is not necessarily equal to rvfi_csr_mcycle_wdata.

Halt Signal

The rvfi_halt signal is meant for liveness properties of cores that can halt execution. It is only needed for cores that can lock up. Tied to 0 for RISC-V compliant cores.

Mode Signal

The rvfi_mode signal shows the *current* privilege mode as opposed to the *effective* privilege mode of the instruction. I.e. for load and store instructions the reported privilege level will therefore not depend on mstatus.mpp and mstatus.mprv.

18.3 Trace output file

Tracing can be enabled during simulation by defining **CV32E40X_TRACE_EXECUTION**. All traced instructions are written to a log file. The log file is named trace_rvfi.log.

18.4 Trace output format

The trace output is in tab-separated columns.

1. **PC**: The program counter
2. **Instr**: The executed instruction (base 16). 32 bit wide instructions (8 hex digits) are uncompressed instructions, 16 bit wide instructions (4 hex digits) are compressed instructions.
3. **rs1_addr** Register read port 1 source address, 0x0 if not used by instruction
4. **rs1_data** Register read port 1 read data, 0x0 if not used by instruction

5. **rs2_addr** Register read port 2 source address, 0x0 if not used by instruction
6. **rs2_data** Register read port 2 read data, 0x0 if not used by instruction
7. **rd_addr** Register write port 1 destination address, 0x0 if not used by instruction
8. **rd_data** Register write port 1 write data, 0x0 if not used by instruction
9. **mem_addr** Memory address for instructions accessing memory
10. **rvfi_mem_rmask** Bitmask specifying which bytes in `rvfi_mem_rdata` contain valid read data
11. **rvfi_mem_wmask** Bitmask specifying which bytes in `rvfi_mem_wdata` contain valid write data
12. **rvfi_mem_rdata** The data read from memory address specified in `mem_addr`
13. **rvfi_mem_wdata** The data written to memory address specified in `mem_addr`

```
PC        Instr     rs1_addr  rs1_rdata  rs2_addr  rs2_rdata  rd_addr  rd_wdata    mem_
↪addr mem_rmask mem_wmask mem_rdata mem_wdata
00001f9c  14c70793        0e   000096c8        0c   00000000       0f  00009814   ␣
↪00009814         0         0  00000000  00000000
00001fa0  14f72423        0e   000096c8        0f   00009814       00  00000000   ␣
↪00009810         0         f  00000000  00009814
00001fa4  0000bf6d        1f   00000000        1b   00000000       00  00000000   ␣
↪00001fa6         0         0  00000000  00000000
00001f5e  000043d8        0f   00009814        04   00000000       0e  00000000   ␣
↪00009818         f         0  00000000  00000000
00001f60  0000487d        00   00000000        1f   00000000       10  0000001f   ␣
↪0000001f         0         0  00000000  00000000
```

CHAPTER

NINETEEN

CORE-V INSTRUCTION SET EXTENSIONS

19.1 Custom instructions

CV32E40X supports the custom instruction(s) listed in Table 19.1.

Table 19.1: Custom instructions

Custom instruction	Encoding	Description
`wfe`	0x8C00_0073	Wait For Event, see *WFE*.

Further custom instructions can be added external to the core via the eXtension interface described in *eXtension Interface*.

19.2 Custom CSRs

CV32E40X does not contain custom CSRs.

CHAPTER

TWENTY

CORE VERSIONS AND RTL FREEZE RULES

The CV32E40X is defined by the `marchid` and `mimpid` tuple. The tuple identify which sets of parameters have been verified by OpenHW Group, and once RTL Freeze is achieved, no further non-logically equivalent changes are allowed on that set of parameters.

The RTL Freeze version of the core is indentified by a GitHub tag with the format cv32e40x_vMAJOR.MINOR.PATCH (e.g. cv32e40x_v1.0.0). In addition, the release date is reported in the documentation.

20.1 What happens after RTL Freeze?

20.1.1 A bug is found

If a bug is found that affect the already frozen parameter set, the RTL changes required to fix such bug are non-logically equivalent by definition. Therefore, the RTL changes are applied only on a different `mimpid` value and the bug and the fix must be documented. These changes are visible by software as the `mimpid` has a different value. Every bug or set of bugs found must be followed by another RTL Freeze release and a new GitHub tag.

20.1.2 RTL changes on non-verified yet parameters

If changes affecting the core on a non-frozen parameter set are required, then such changes must remain logically equivalent for the already frozen set of parameters (except for the required mimpid update), and they must be applied on a different `mimpid` value. They can be non-logically equivalent to a non-frozen set of parameters. These changes are visible by software as the `mimpid` has a different value. Once the new set of parameters is verified and achieved the sign-off for RTL freeze, a new GitHub tag and version of the core is released.

20.1.3 PPA optimizations and new features

Non-logically equivalent PPA optimizations and new features are not allowed on a given set of RTL frozen parameters (e.g., a faster divider). If PPA optimizations are logically-equivalent instead, they can be applied without changing the `mimpid` value (as such changes are not visible in software). However, a new GitHub tag should be released and changes documented.

20.2 Released core versions

The verified parameter sets of the core, their implementation version, GitHub tags, and dates are reported here.

CHAPTER

TWENTYONE

GLOSSARY

- **ALU**: Arithmetic/Logic Unit
- **ASIC**: Application-Specific Integrated Circuit
- **Byte**: 8-bit data item
- **CPU**: Central Processing Unit, processor
- **CSR**: Control and Status Register
- **Custom extension**: Non-Standard extension to the RISC-V base instruction set (RISC-V Instruction Set Manual, Volume I: User-Level ISA)
- **EXE**: Instruction Execute
- **FPGA**: Field Programmable Gate Array
- **FPU**: Floating Point Unit
- **Halfword**: 16-bit data item
- **Halfword aligned address**: An address is halfword aligned if it is divisible by 2
- **ID**: Instruction Decode
- **IF**: Instruction Fetch (*Instruction Fetch*)
- **ISA**: Instruction Set Architecture
- **KGE**: kilo gate equivalents (NAND2)
- **LSU**: Load Store Unit (*Load-Store-Unit (LSU)*)
- **M-Mode**: Machine Mode (RISC-V Instruction Set Manual, Volume II: Privileged Architecture)
- **NMI**: Non-Maskable Interrupt
- **OBI**: Open Bus Interface
- **PC**: Program Counter
- **PMA**: Physical Memory Attribution
- **RV32C**: RISC-V Compressed (C extension)
- **RV32F**: RISC-V Floating Point (F extension)
- **SIMD**: Single Instruction/Multiple Data
- **Standard extension**: Standard extension to the RISC-V base instruction set (RISC-V Instruction Set Manual, Volume I: User-Level ISA)
- **WARL**: Write Any Values, Reads Legal Values

- **WB**: Write Back of instruction results
- **WLRL**: Write/Read Only Legal Values
- **Word**: 32-bit data item
- **Word aligned address**: An address is word aligned if it is divisible by 4
- **WPRI**: Reserved Writes Preserve Values, Reads Ignore Values

BIBLIOGRAPHY

[RISC-V-UNPRIV] RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213 (December 13, 2019), https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

[RISC-V-PRIV] RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211105-signoff (November 5, 2021), https://github.com/riscv/riscv-isa-manual/releases/download/draft-20211105-c30284b/riscv-privileged.pdf

[RISC-V-DEBUG] RISC-V Debug Support, version 1.0.0-STABLE, 9dc1ee4e4653730f1b318731f9ea8e97f116670a, https://github.com/riscv/riscv-debug-spec/blob/14a8d628e1fb736043eb54e0596adddb9717f0de/riscv-debug-stable.pdf

[RISC-V-SMCLIC] "Smclic" Core-Local Interrupt Controller (CLIC) RISC-V Privileged Architecture Extension, version 0.9-draft, 9/27/2022, https://github.com/riscv/riscv-fast-interrupt/blob/2749a7b7adb01fc308173441b8a6131a5d128f20/clic.pdf

[RISC-V-ZBA_ZBB_ZBC_ZBS] RISC-V Bit Manipulation ISA-extensions, Version 1.0.0-38-g865e7a7, 2021-06-28, https://github.com/riscv/riscv-bitmanip/releases/download/1.0.0/bitmanip-1.0.0-38-g865e7a7.pdf

[RISC-V-ZCA_ZCB_ZCMP_ZCMT] RISC-V Standard Extension for the **Zca**, **Zcb**, **Zcmp**, **Zcmt** subsets of **Zc**, v1.0.0-RC5.6 (not ratified yet), https://github.com/riscv/riscv-code-size-reduction/blob/cd13c6b17ccb7e1b8fc8b69e76179b339bcc2b32/Zc-specification/Zc.adoc

https://github.com/riscv/riscv-code-size-reduction/releases/download/v1.0.0-RC5.6/Zc-v1.0.0-RC5.6.pdf

[RISC-V-CRYPTO] RISC-V Cryptography Extensions Volume I, Scalar & Entropy Source Instructions, Version v1.0.0, 2'nd December, 2021: Ratified, https://github.com/riscv/riscv-crypto/releases/download/v1.0.0-scalar/riscv-crypto-spec-scalar-v1.0.0.pdf

[OPENHW-OBI] OpenHW Open Bus Interface (OBI) protocol, version 1.5.0, https://github.com/openhwgroup/core-v-docs/blob/master/cores/obi/OBI-v1.5.0.pdf

[OPENHW-XIF] OpenHW eXtension Interface, revision 458c8a73, https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/

[SYMBIOTIC-RVFI] Symbiotic EDA RISC-V Formal Interface https://github.com/SymbioticEDA/riscv-formal/blob/master/docs/rvfi.md