CC-RV32HR-C HIGH-RELIABILITY RISC-V PROCESSOR CORE

USER MANUAL

VERSION 1.1

ChipCraft Sp. z o.o.
Bukowinska 22 lok. 9B
02-703 Warsaw, POLAND

1. FEATURES

- **RV32HR Processor Core**
  - RV32IMAFD (RV32G) instruction set
  - high-performance, in-order, single-issue pipeline
  - PHT, GHR, BTB dynamic branch prediction
  - 2 processing cores working in split/lock DCLS mode
- **Memory Architecture**
  - Hardware Stack Protection
  - Physical Memory Protection
  - Instruction Cache with optional scrambling and parity protection
  - Data Cache with optional scrambling and ECC protection (1-bit correction, 2-bit detection per 32-bit word)
  - Tightly coupled SRAM, single-cycle access at system speed per Core, with optional ECC protection (1-bit correction, 2-bit detection per 32-bit word)
- **Tightly Coupled Peripherals**
  - PLIC, CLINT, CSR, Multicore, Power Management, running at core speed
  - Machine Time Counter running at core speed
- **Debugging**
  - On-Chip Debugger with UART and JTAG Debug Link
  - JTAG Boundary Scan on All Digital Pins

2. INSTRUCTION SET

This chapter contains the brief introduction to the CC-RV32HR-C instruction set architecture.

The CC-RV32HR-C processor adheres to the **"The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Document Version 20190608-Base-Ratified"** and **"The RISC-V Instruction Set Manual Volume II: Privileged Architecture Document Version 20211203"**.

The CC-RV32HR-C processing unit comprises of single-issue, 6-stage fully interlocked pipeline executing **RV32IMAFD (RV32G)** instruction set including **Zicsr**, **Zifence** and **Zicntr** extensions. The CC-RV32HR-C implements **M**, and **U** privileged levels.

For detailed informations about integer unit pipeline microarchitecture please refer to the **CC-RV32ST-IU** document.

3. PERFORMANCE AND UTILIZATION

Table 3.1 shows the available single core integer unit performance. The achieved results will depend on the final system specification.

Table 3.1: Integer unit performance.

Benchmark	Score
Dhrystone	up to 1.32 DMIPS/MHz/Core
Coremark	up to 2.26 CoreMark/MHz/Core

Table 3.2 shows the example DCLS processor cluster utilization with Xiling Ultrascale FPGA. The processor system was configured with typical parameters without floating point unit and Physical Memory Protection. Register file and cache memories are not included.

Table 3.2: Xilinx Ultrascale resource utilization.

Block	CLB LUTs	CLB REGs	DSPs
Processor cluster:	16484	13859	8
Core 0	2776	1637	4
Core 1	2776	1637	4
I-Cache Controller (shared)	881	854	0
D-Cache Controller 0	1026	1268	0
D-Cache Controller 1	1026	1268	0
CSRs (shared)	523	1133	0
PLIC	637	412	0
CLINT	300	465	0
Multicore Controller (DCLS)	398	818	0
Power Management	38	100	0
Internal Interconnect	1037	611	0
Debug System (shared)	4660	2983	0

4. BLOCK DIAGRAM

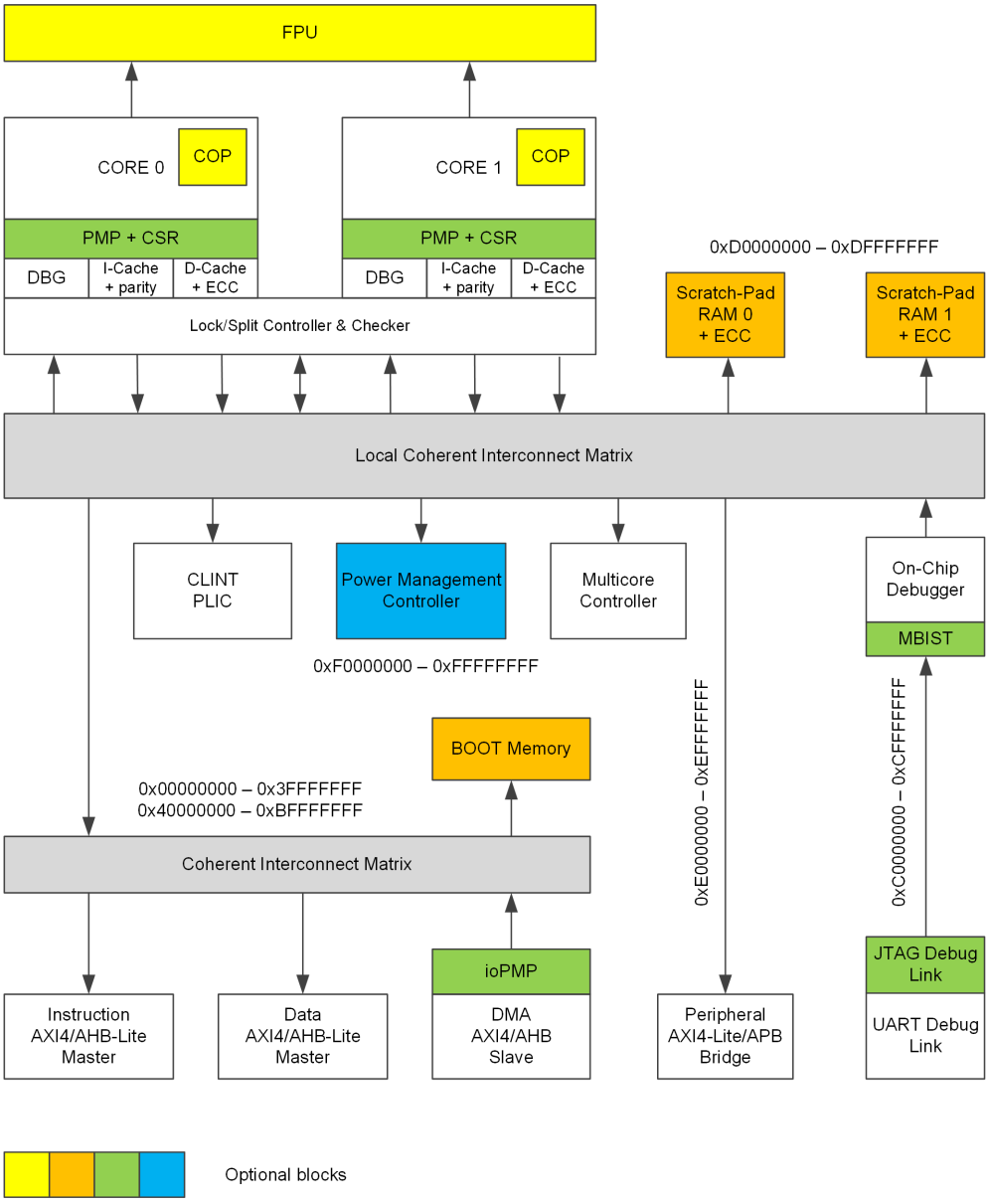
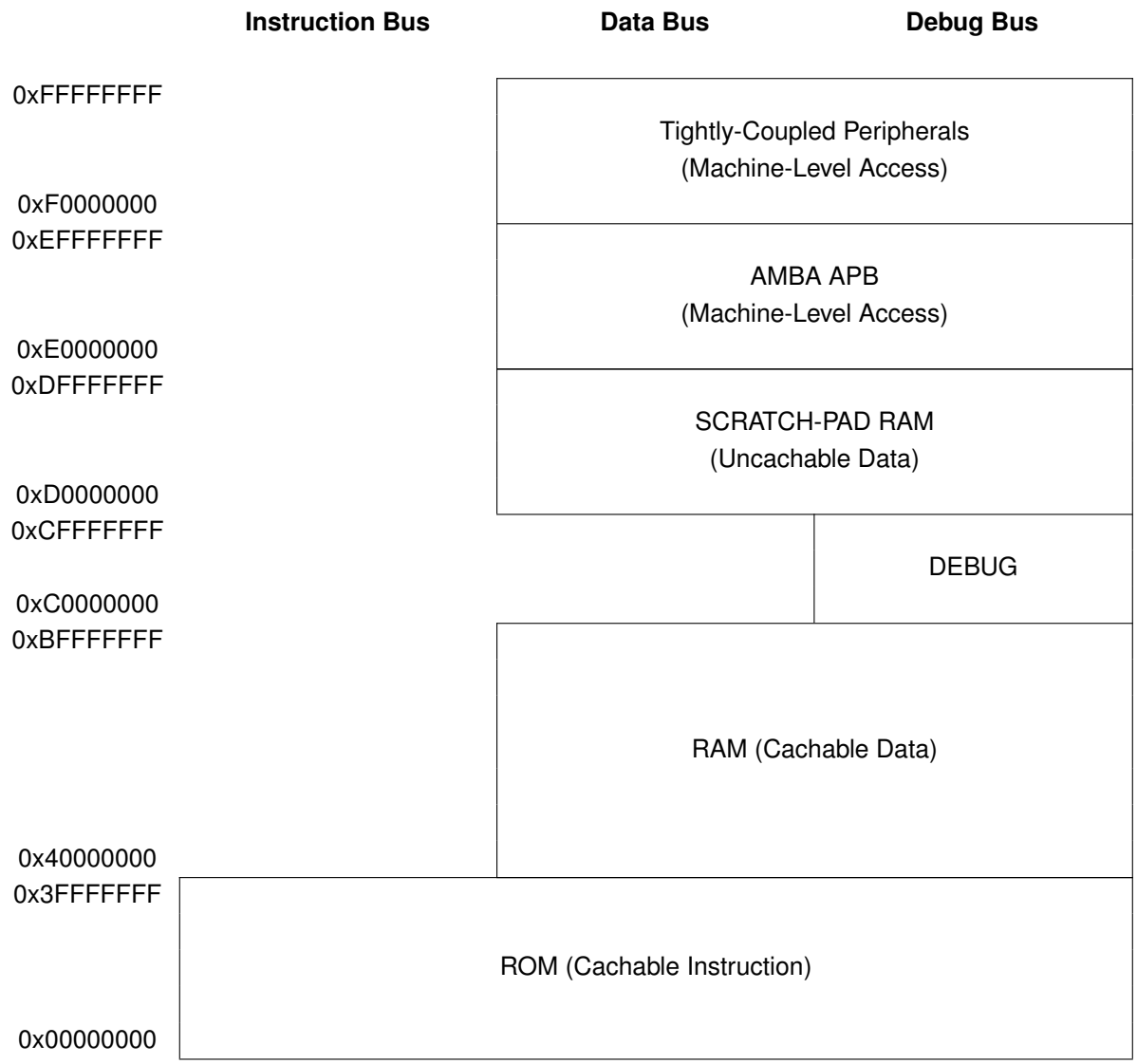


Figure 4.1: Processor block diagram.

5. MEMORY MAP


Instruction Bus
Data Bus
Debug Bus
0xFFFFFFFF
0xF0000000
0xEFFFFFFF
0xE0000000
0xDFFFFFFF
0xD0000000
0xCFFFFFFF
0xC0000000
0xBFFFFFFF
0x40000000
0x3FFFFFFF
0x00000000
Tightly-Coupled Peripherals
(Machine-Level Access)
AMBA APB
(Machine-Level Access)
SCRATCH-PAD RAM
(Uncachable Data)
DEBUG
RAM (Cachable Data)
ROM (Cachable Instruction)

5.1. REGION DESCRIPTIONS

CC-RV32HR-C processor implements little-endian memory model only. This means that the most significant byte, which is the byte containing the most significant bit, is stored at the lower address.

5.1.1. ROM REGION

Cachable read-execute memory region containing user program binary image. The indirect writing to the ROM region can be possible using dedicated peripherals if processor configuration supports any. Additionally, region should be unlocked for writing using mromunlock CSR. If region is locked for writing the store attempt will silently fail without raising exception. To increase system bandwidth, store operation that results in ROM bus error will also silently fail without raising exception.

5.1.2. SCRATCH-PAD RAM REGION

Uncachable, high-speed internal memory region used for temporary data storage. The region is dedicated for storage of local and private data structures (e.g. program stack). Under this address range each processor core has its own private memory region inaccessible by other cores.

5.1.3. RAM REGION

Cachable and coherent memory region used for temporary data storage. The region is shared across all processor cores and DMA channels. RAM region is the only memory region that supports load-reserved (LR) and store conditional (SC) atomic instructions. To increase system bandwidth, store operation that results in RAM bus error will silently fail without raising exception.

5.1.4. DEBUG REGION

The region is accessible only in Debug Mode using On-Chip Debugger and is seen as reserved for user program. The available subregions are listed in Table 5.1.

Table 5.1: Debug Mode regions.

Address	Description
0xC0000000 - 0xC000000C	4 x Breakpoint
0xC0000010 - 0xC000001C	4 x Watchpoint
0xC0000020 - 0xC0000020	Burst Counter Register
0xC0000024 - 0xC0000024	Debug Version Register
0xC1000000 - 0xC100007C	Integer Register-File
0xC1000080 - 0xC10000FC	Floating-Point Register-File
0xC2000000 - 0xC20xxxxx	Instruction Cache Data
0xC2100000 - 0xC21xxxxx	Instruction Cache Tag
0xC2200000 - 0xC22xxxxx	Data Cache Data
0xC2300000 - 0xC23xxxxx	Data Cache Tag

6. CONTROL AND STATUS REGISTERS

Control and Status Registers (CSRs) are implemented according to the RV32 ISA specification. Tthe SYSTEM major opcode is used to encode all privileged instructions including atomic access to CSRs. In addition to the user-level state, an implementation may contain additional CSRs, accessible by some subset of the privilege levels using the CSR instructions described in the user-level manual. The following chapters summarizes the implemented CSRs and their privilege level. Note that although CSRs and instructions are associated with one privilege level, they are also accessible at all higher privilege levels.

6.1. MACHINE-LEVEL CSRS

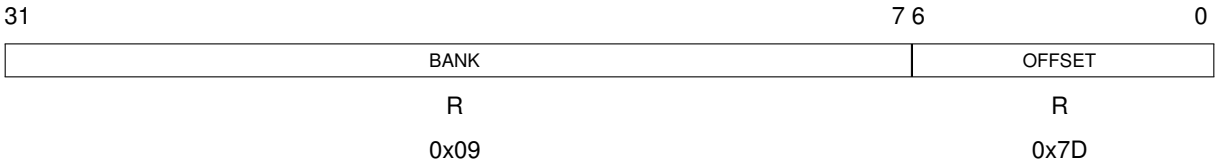
Table 6.1 summarizes the implemented machine-level CSRs.

Table 6.1: Machine-level CSR registers.

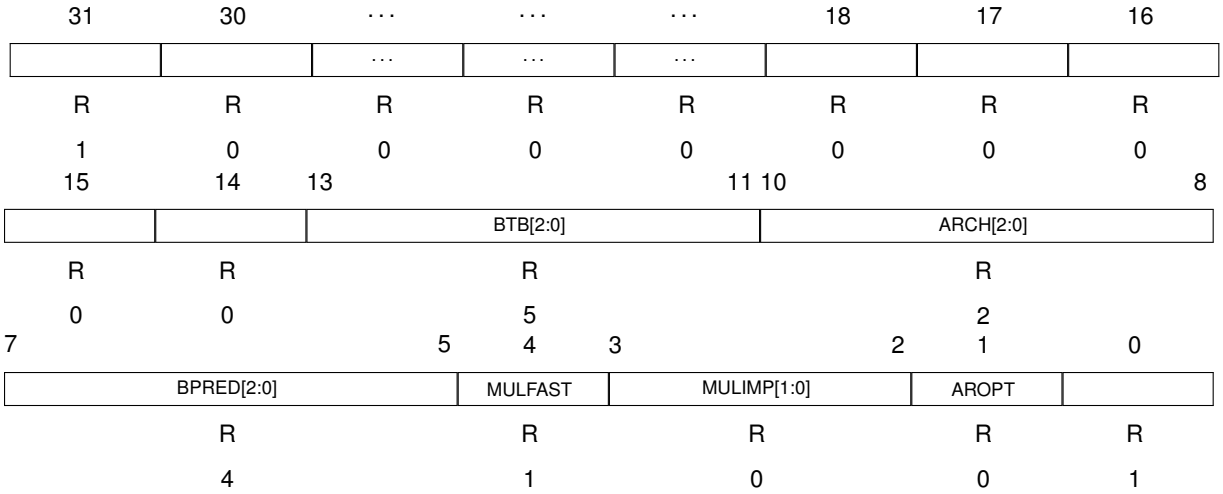
Number	Privilege	Name	Description
Machine Information Registers			
0xF11	MRO	mvendorid	Vendor ID
0xF12	MRO	marchid	Architecture ID
0xF13	MRO	mimpid	Implementation ID
0xF14	MRO	mhartid	Hardware thread ID
0xF15	MRO	mconfigptr	Pointer to configuration data structure
Machine Trap Setup			
0x300	MRW	mstatus	Machine status register
0x301	MRW	misa	ISA and extensions
0x304	MRW	mie	Machine interrupt-enable register
0x305	MRW	mtvec	Machine trap-handler base address
0x306	MRW	mcounteren	Machine counter enable
0x310	MRW	mstatush	Additional machine status register
Machine Trap Handling			
0x340	MRW	mscratch	Scratch register for machine trap handlers
0x341	MRW	mepc	Machine exception program counter
0x342	MRW	mcause	Machine trap cause
0x343	MRW	mtval	Machine bad address or instruction
0x344	MRW	mip	Machine interrupt pending
Machine Memory Protection			
0x3A0	MRW	mpmcfg0	Physical memory protection configuration
0x3A1	MRW	mpmcfg1	Physical memory protection configuration, RV32 only
0x3A2	MRW	mpmcfg2	Physical memory protection configuration
0x3A3	MRW	mpmcfg3	Physical memory protection configuration, RV32 only
0x3B0	MRW	mpmaddr0	Physical memory protection address register
0x3B1	MRW	mpmaddr1	Physical memory protection address register
		...	
0x3BF	MRW	mpmaddr15	Physical memory protection address register

Machine Counter/Timers			
0xB00	MRW	mcycle	Machine cycle counter
0xB02	MRW	minstret	Machine instructions-retired counter
0xB80	MRW	mcycleh	Upper 32-bits of mcycle, RV32I only
0xB82	MRW	minstreth	Upper 32-bits of minstret, RV32I only
Machine Counter Setup			
0x320	MRW	mcounterinhibit	Machine counter-inhibit register

6.1.1. MACHINE VENDOR ID REGISTER



6.1.2. MACHINE ARCHITECTURE ID REGISTER



AROPT *Area Optimization*

0 Not implemented.

1 Implemented.

MULIMP[1:0] *Multiplier Implementation*

0 Iterative 16x16.

1 Single-cycle 32x32.

2 Iterative 1-bit.

MULFAST *Fast 16x16 Multiplications*

0 Not implemented.

1 Implemented.

BPRED[2:0] *Branch Prediction Scheme*

0 Branch-always.

1 Branch-never.

2 Opcode.

3 PHT.

4 Gshare.

ARCH[2:0] *Processor Microarchitecture*

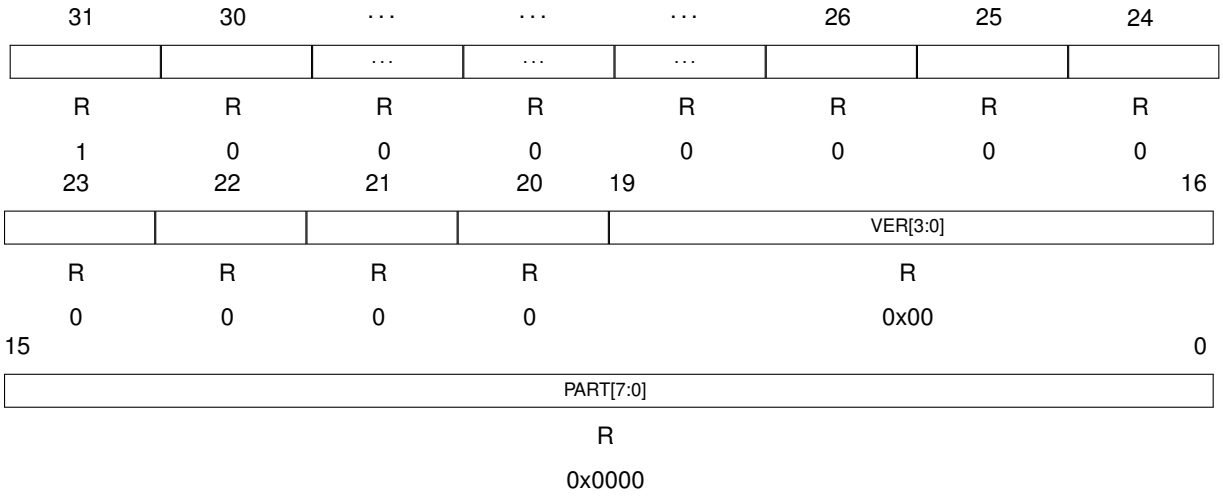
1 RV32 Low Power.

2 RV32 High Performance.

BTB[2:0] *BTB Implementation Size*

If greater than zero, the number of entries equals 2^{BTB}.

6.1.3. MACHINE IMPLEMENTATION ID REGISTER



VER[3:0] *Version Number*

Field contains JTAG IDCODE version number.

PART[7:0] *Part Number*

Field contains JTAG IDCODE part number.

6.1.4. MACHINE CAUSE REGISTER

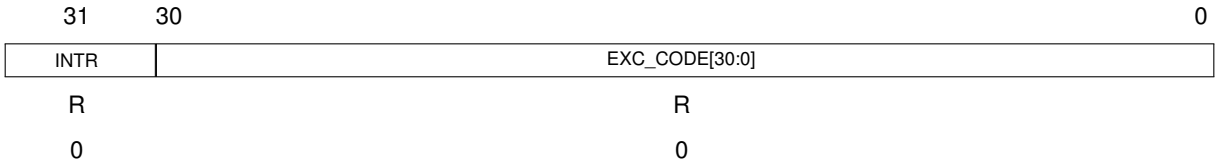


Table 6.2: Exception codes.

Interrupt	Exception Code	Description
1	0	User software interrupt
1	1	Supervisor software interrupt
1	2	Reserved
1	3	Machine software interrupt
1	4	User timer interrupt
1	5	Supervisor timer interrupt
1	6	Reserved
1	7	Machine timer interrupt
1	8	User external interrupt
1	9	Supervisor external interrupt
1	10	Reserved
1	11	Machine external interrupt
1	$\geq$ 12	Reserved
1	0xFFFF	Non-maskable interrupt

0	0	Instruction address misalign
0	1	Instruction access fault
0	2	Illegal instruction
0	3	Breakpoint
0	4	Load address misalign
0	5	Load access fault
0	6	Store/AMO adddress misalign
0	7	Store/AMO access fault
0	8	Environment call from U-mode
0	9	Environment call from S-mode
0	10	Reserved
0	11	Environment call from M-mode
0	12	Instruction page fault
0	13	Load page fault
0	14	Reserved
0	15	Store/AMO page fault
0	≥ 16	Reserved

6.2. USER-LEVEL CSRS

Table 6.3 below summarizes the implemented user-level CSRs.

Table 6.3: Machine-level CSR registers.

Number	Privilege	Name	Description
User Floating-Point Registers			
0x001	URW	fflags	Floating-Point Accrued Exceptions
0x002	URW	frm	Floating-Point Dynamic Rounding Mode
0x003	URW	fcsr	Floating-Point Control and Status Register
User Counter/Timers			
0xC00	URO	cycle	Cycle counter for RDCYCLE instruction
0xC01	URO	time	Timer for RDTIME instruction
0xC02	URO	instret	Instructions-retired counter for RDINSTRET instruction
0xC80	URO	cycleh	Upper 32-bits of cycle, RV32I only
0xC81	URO	timeh	Upper 32-bits of time, RV32I only
0xC82	URO	instreth	Upper 32-bits of instret, RV32I only

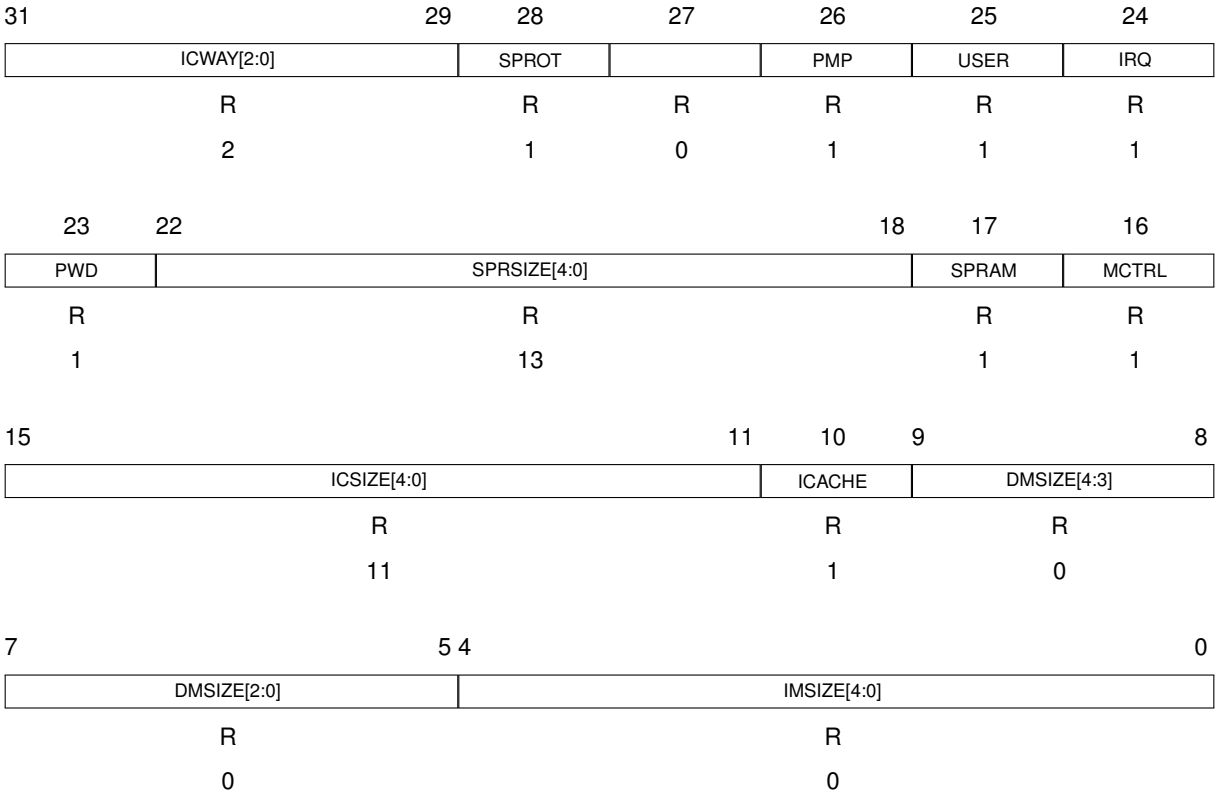
6.3. CUSTOM MACHINE-LEVEL CSRS

Table 6.4 below summarizes the implemented custom machine-level CSRs.

Table 6.4: Machine-level custom CSR registers.

Number	Privilege	Name	Description
User Counter/Timers			
0x7C0	MRW	mconfig0	Processor Configuration Register 0
0x7C1	MRW	mconfig1	Processor Configuration Register 1
0x7C2	MRW	mconfig2	Processor Configuration Register 2
0x7C3	MRW	mtileid	Processor Tile Index
0x7C8	MRW	mcontrol	Processor Control Register
0x7C9	MRW	mstackmin	Minimum Stack Pointer Value Register
0x7CA	MRW	mstackmax	Maximum Stack Pointer Value Register
0x7CB	MRW	mdbgbaud	On-Chip Debugger Baud Register
0x7CC	MRW	mremap	Memory Remap Register
0x7CD	MRW	mromunlock	ROM Unlock Register

6.3.1. PROCESSOR CONFIGURATION REGISTER 0



IMSIZE[4:0] *On-Chip Instruction Memory Size*

Instruction memory bus address size.

DMSIZE[4:0] *On-Chip Data Memory Size*

Data memory bus address size.

ICACHE *Instruction Cache*

Bit is set if processor has instruction cache.

ICSIZE[4:0] *Instruction Cache Size*

Size of instruction cache way - $2^{ICSIZE}b$.

MCTRL *Multicore Controller*

Bit is set if processor has Multicore Controller.

SPRAM *Scratch-pad RAM*

Bit is set if processor has scratch-pad RAM memory region.

SPRSIZE[4:0] *Scratch-pad RAM Size*

Size of scratch-pad RAM memory - $2^{SPRSIZE}b$.

PWD *Power Management Controller*

Bit is set if processor has Power Management Controller.

CSR *System Controller*

Bit is set if processor has System Controller.

USER *User Mode*

Bit is set if processor supports user mode.

PMP *Physical Memory Protection*

Bit is set if processor has Physical Memory Protection Unit.

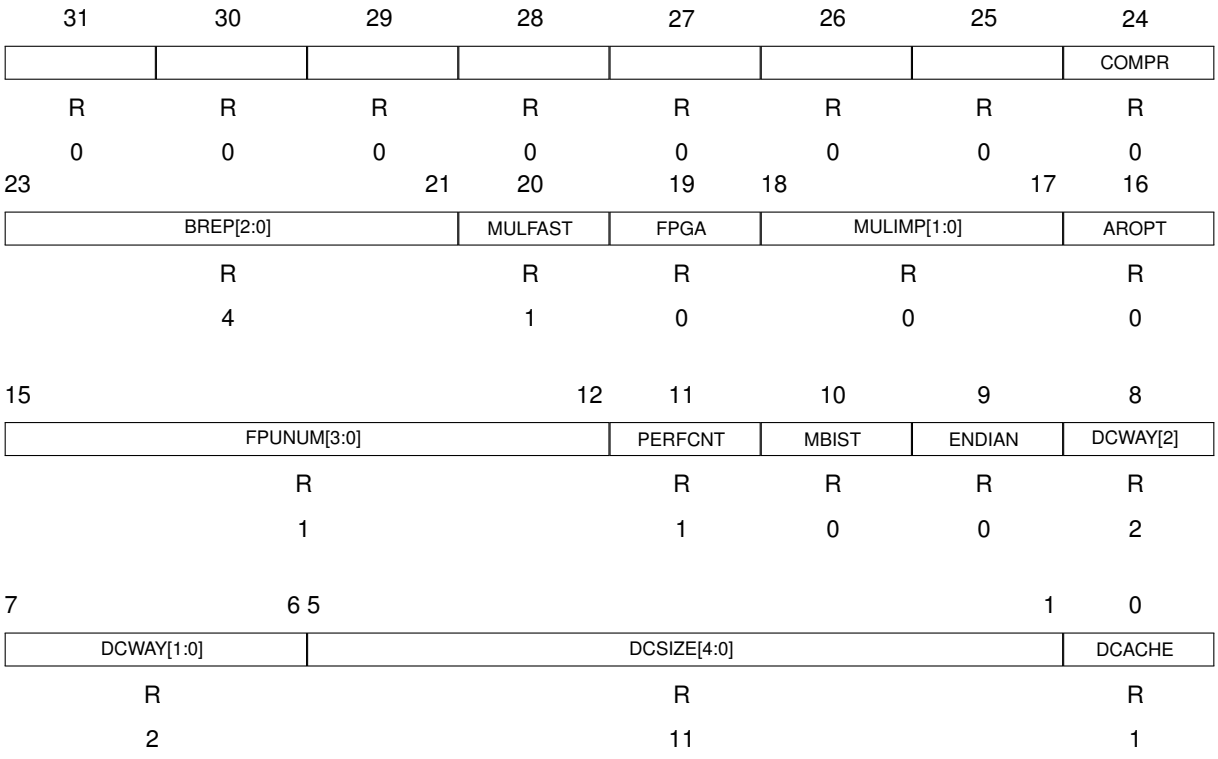
SPROT *Stack Protection*

Bit is set if processor has stack protection hardware.

ICWAY[2:0] *Instruction Cache Ways*

Number of instruction cache ways.

6.3.2. PROCESSOR CONFIGURATION REGISTER 1



DCACHE *Data Cache*

Bit is set if processor has data cache.

DCSIZE[4:0] *Data Cache Size*

Size of data cache way - $2^{DCSIZE}{}_b$.

DCWAY[2:0] *Data Cache Ways*

Number of data cache ways.

ENDIAN *System Endianness*

0 Little-endian.

1 Big-endian.

MBIST *MBIST Controller*

Bit is set if memory BIST controller is implemented.

PERFCNT *Performance Counter*

Bit is set if high-resolution performance counter is implemented.

FPUNUM[3:0] *Number of FPUs*

Number of Floating Point Units.

AROPT *Area Optimization*

0 Not implemented.

1 Implemented.

MULIMP[1:0] *Multiplier Implementation*

0 Iterative 16x16.

1 Single-cycle 32x32.

2 Iterative 1-bit.

FPGA *Technology*

0 ASIC.

1 FPGA.

MULFAST *Fast 16x16 Multiplications*

0 Not implemented.

1 Implemented.

BPRED[2:0] *Branch Prediction Scheme*

0 Branch-always.

1 Branch-never.

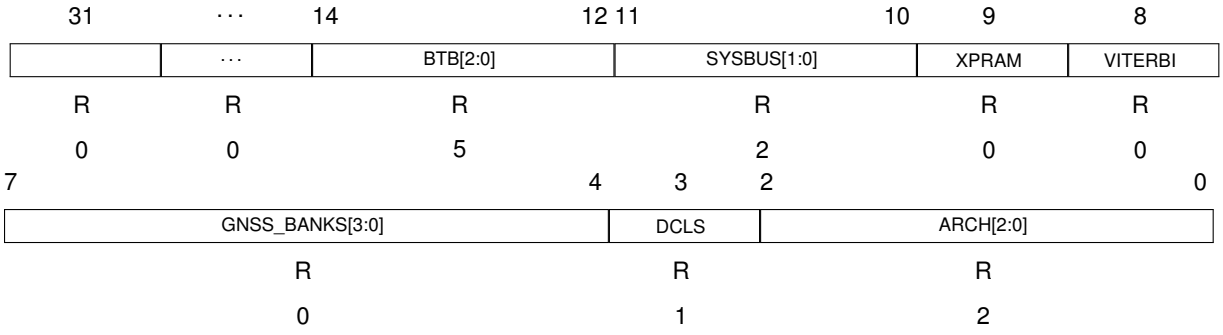
2 Opcode.

3 Dynamic.

COMPR *Compressed ISE*

Compressed 16-bit instruction set extension.

6.3.3. PROCESSOR CONFIGURATION REGISTER 2



ARCH[2:0] *Processor Microarchitecture*

0 Non-RV32.

1 RV32 Low Power.

2 RV32 High Performance.

DCLS *Dual-core Lockstep Implementation*

0 Not implemented.

1 Implemented.

GNSS_BANKS[3:0] *GNSS Banks*

Number of GNSS banks per processor core.

VITERBI *Viterbi Decoder Implementation*

0 Not implemented.

1 Implemented.

XPRAM *Executable Scratch-pad RAM*

0 Not implemented.

1 Implemented.

SYSBUS[1:0] *System Bus Width*

0 32-bit bus.

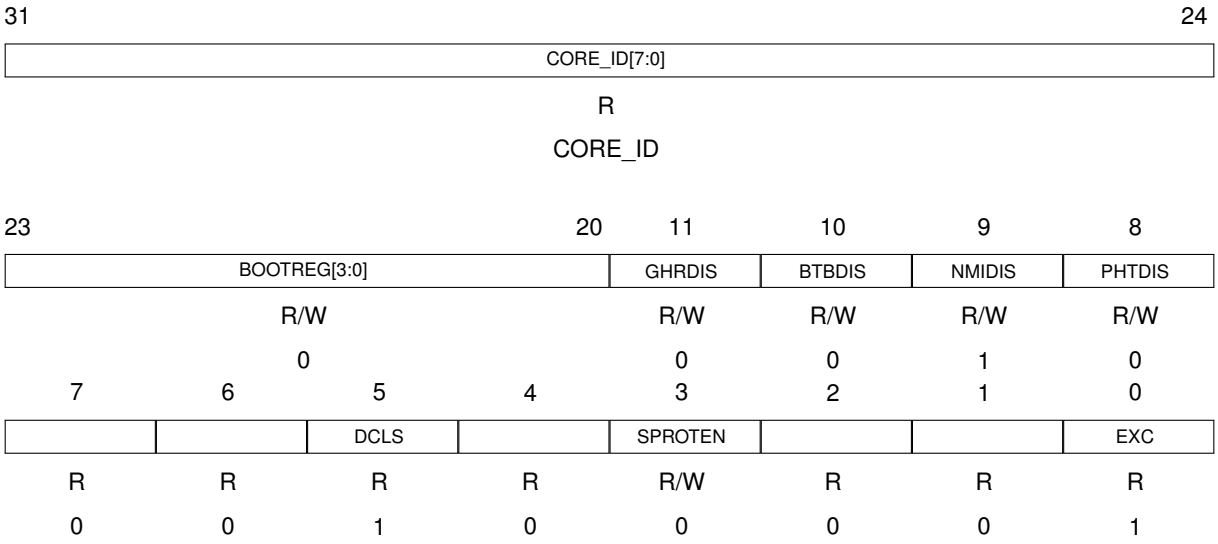
1 64-bit bus.

2 128-bit bus.

BTB[2:0] *BTB Implementation Size*

If greater than zero, the number of entries equals 2^{BTB}.

6.3.4. PROCESSOR CONTROL REGISTER



EXC *Exception Support*

Bit is set if processor supports exceptions.

SPROTEN *Stack Protection Enable*

Bit is used to enable or disable hardware stack protection.

DCLS *Dual-core Lockstep Mode*

Bit is set if processor is running in dual-core lockstep mode. If this bit is 0 after power-on reset processor does not support dual-core locktep mode.

PHTDIS *Disable PHT Branch Prediction*

Set this bit to disable PHT branch prediction.

NMIDIS *Disable NMI*

Clean this bit to enable non-maskable interrupts. Can be written only once and impacts all processor cores.

BTBDIS *Disable BTB Branch Prediction*

Set this bit to disable BTB branch prediction.

GHRDIS *Disable GHR Branch Prediction*

Set this bit to disable GHR branch prediction.

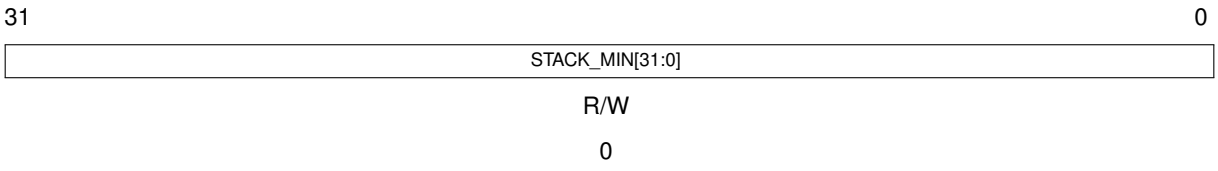
BOOTREG[3:0] *Boot Region*

Processor boot region. Can be used with conjunction with Executable Scratch-pad RAM. Can be changed only by core 0.

CORE_ID[7:0] *Core ID*

Stores index of processor core.

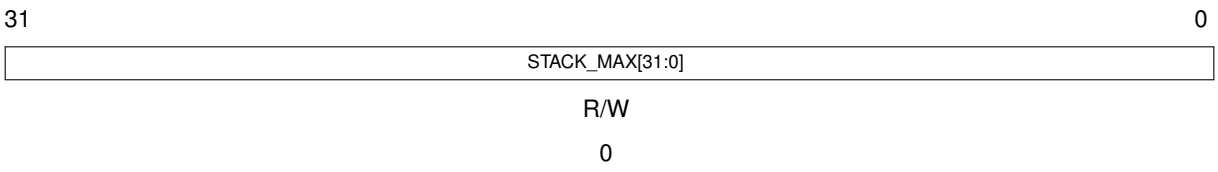
6.3.5. MINIMUM STACK POINTER VALUE REGISTER



STACK_MIN[31:0] *Minimum Stack Pointer Value*

Stores the minimal value of Stack Pointer Register. If Stack Protection is enabled, going below this value will cause an exception.

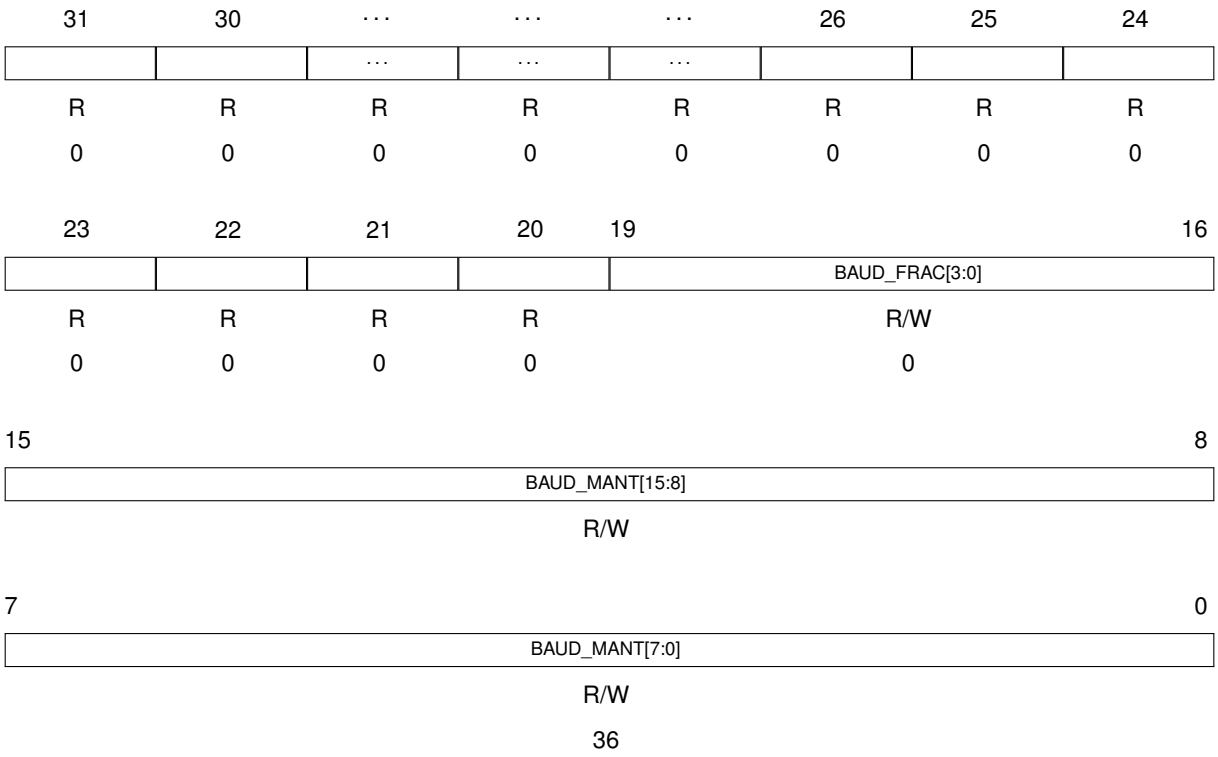
6.3.6. MAXIMUM STACK POINTER VALUE REGISTER



STACK_MAX[31:0] *Maximum Stack Pointer Value*

Stores the maximum value of Stack Pointer Register. If Stack Protection is enabled, going above this value will cause an exception.

6.3.7. ON-CHIP DEBUGGER BAUD REGISTER



BAUD_MANT[15:0] *Debug Baud Rate Mantissa*

Mantissa part of On-Chip Debugger baud rate generator.

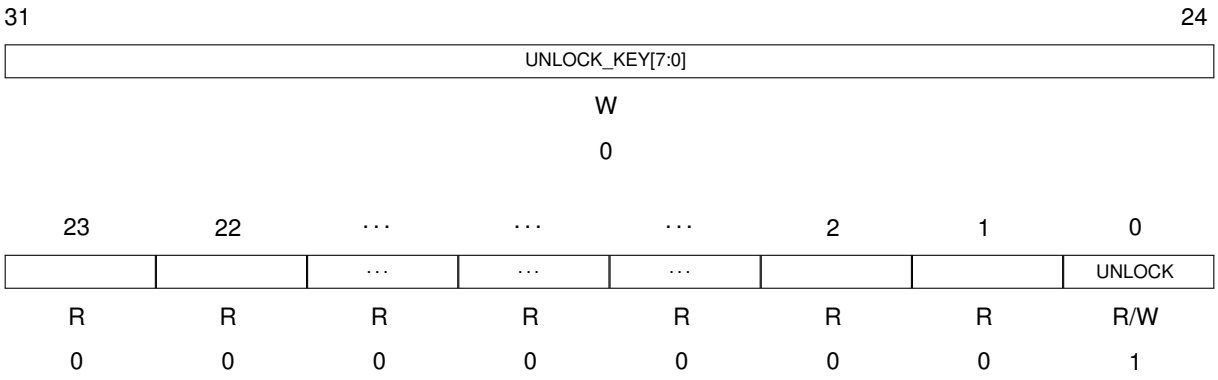
$$BR = \frac{F_{CLK}}{16 * (DIV\_MANT[15:0] + \frac{DIV\_FRAC[3:0]}{16})}.$$

BAUD_FRAC[3:0] *Debug Baud Rate Fraction*

Fractional part of On-Chip Debugger baud rate generator.

$\frac{BAUD\_FRAC[3:0]}{16}$.

6.3.8. ROM UNLOCK REGISTER



UNLOCK *ROM Unlock*

Setting this bit will unlock write operation to the ROM region. ROM region supports only 32-bit access.

UNLOCK_KEY[7:0] *Unlock Key*

This field has to be set to 0xA5 value in order to enable changing ROM unlock state.

7. SCRATCH-PAD RAM CONTROLLER

Scratchpad memory (SPRAM) is a high-speed internal memory directly connected to the CPU core and used for temporary data storage. The SPRAM memory is equiped with EDAC (Error Detection And Correction) module that contains coder/decoder circuitry for parity bits calculation. Coder module appends parity bits to the data word, while the decoder checks stored data agains erros. The EDAC controller supports SEC-DED, what means capability of correcting single error and detection of double error in a data block. Single errors are corrected transparently to the software, while detection of double errror causes issuing of the NMI. The memory content along with parity bits are automatically initialized after power-on-reset or deep reset request.

8. TIGHTLY-COUPLED PERIPHERALS

8.1. REGISTER DESCRIPTION CONVENTION

This section presents the register description convention. The exemplary register description is presented below. The description contains register address and states if register is shared among all processor cores or each core have its exclusive copy. The next lines show particular bits and bit-fields with their names and bit positions. The third line describes the access type which can be read (R), write (W), read/write (R/W) or if bit is not used (N/A). The last line shows the default startup values.

Address: 0xF00A0014

Type: shared/exclusive

31	30	···	···	8	1	0
BIT31	BIT30	···	···	FIELD[4:0]		BIT0
R	W	R	R	R/W		N/A
1	0	0	0	5		0

8.2. MEMORY MAP

Table 8.1 shows the memory map of Tightly-Coupled Peripherals region. Only word-size access is allowed. Otherwise peripheral exception will be raised. Unimplemented memory regions are read-only as 0x00000000. Attempt to perform a store operation on read-only address will silently fail.

Table 8.1: Tightly-Coupled Peripherals memory map.

Address	Peripheral
0xF0010000 - 0xF0010FFF	Multicore Controller
0xF0020000 - 0xF0020FFF	Power Management Controller
0xF0030000 - 0xF0033FFF	CSR Controller
0xF0038000 - 0xF00381FF	Platform-Level Interrupt Controller
0xF0038200 - 0xF00382FF	Inter-Core Interrupt Controller
0xF003C000 - 0xF003C1FF	Core Local Interruptor
0xF0070000 - 0xF0070FFF	Instruction Cache Controller
0xF0072000 - 0xF0072FFF	Data Cache Controller
0xF00B0000 - 0xF00B0FFF	Debug Interface Controller

8.3. MULTICORE CONTROLLER

Multicore Controller is responsible for starting and stopping cores other than Core 0. When not used, cores are in reset state with their clocks turned off. The running core is turned off automatically after finishing its job or by using *Shutdown Register*. Multicore Controller registers are described in detail below.

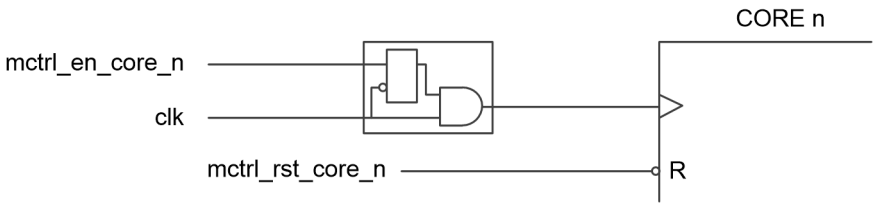


Figure 8.1: Simplified processor cores clock gating scheme.

8.3.1. LOCKSTEP CONTROLLER

Both processor cores can work in either one of the two modes: dual-core lockstep mode or split mode. In split mode, both cores work independently. In dual-core lockstep mode, cores are coupled together and acts as one, transparently to the software. The cycle-by-cycle hardware checker generates and compares cores output signatures. On miscompare hardware error is issued and processor is reset. Reset cause can be checked in Reset Cause Register 8.4.2. After reset, the processor starts its operation in dual-core lockstep mode. To transit to the split mode the Shutdown Register 8.3.5 should be used on the second core.

The Sphere of Replication (SoR) includes:

- Processor Core,
- Instruction Cache,
- Data Cache,
- Scratch-Pad RAM,
- Interrupt Controller,
- Physical Memory Protection,
- Exclusive CRS registers.

8.3.2. REGISTERS LIST

Table 8.2: Multicore Controller registers list.

Address Offset	Register	Name
0xF0010000	STATUS	Status Register
0xF0010004	CORE_NUM	Core Number Register
0xF0010008	CORE_SHDN	Shutdown Register
0xF0010010	CORE_0_ADDR	Core 0 Start Address Register
0xF0010014	CORE_1_ADDR	Core 1 Start Address Register
0xF0010018	CORE_2_ADDR	Core 2 Start Address Register
...	...	...
0xF00100D8	CORE_0_RUN	Core 0 Run Register
0xF00100DC	CORE_1_RUN	Core 1 Run Register
0xF00100E0	CORE_2_RUN	Core 2 Run Register
...	...	...
0xF00101D8	INJECT_MASK_0	Core 0 Injection Mask Register
0xF00101DC	INJECT_MASK_1	Core 1 Injection Mask Register
0xF00101E0	DEADLOCK_MAX	Core Deadlock Counter Max Register
0xF00101E4	ERROR_PC	Error Program Counter Register
0xF00101E8	ERROR_STAT	Error Statistics Register
0xF00101EC	DCLS_CTRL	Lockstep Control Register

8.3.3. STATUS REGISTER

Address: 0x00

Type: shared

31	30	···	···	···	18	17	16
		···	···	···			
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

15	14	···	···	···	2	1	0
STAT15	STAT14	···	···	···	STAT2	STAT1	STAT0
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	1

STAT*n* *Core n Status*

Indicates if Core *n* is currently running. Only necessary bits are implemented.

8.3.4. CORE NUMBER REGISTER

Address: 0x04

Type: shared

31	0
CORE_NUM[31:0]	
R	
2	

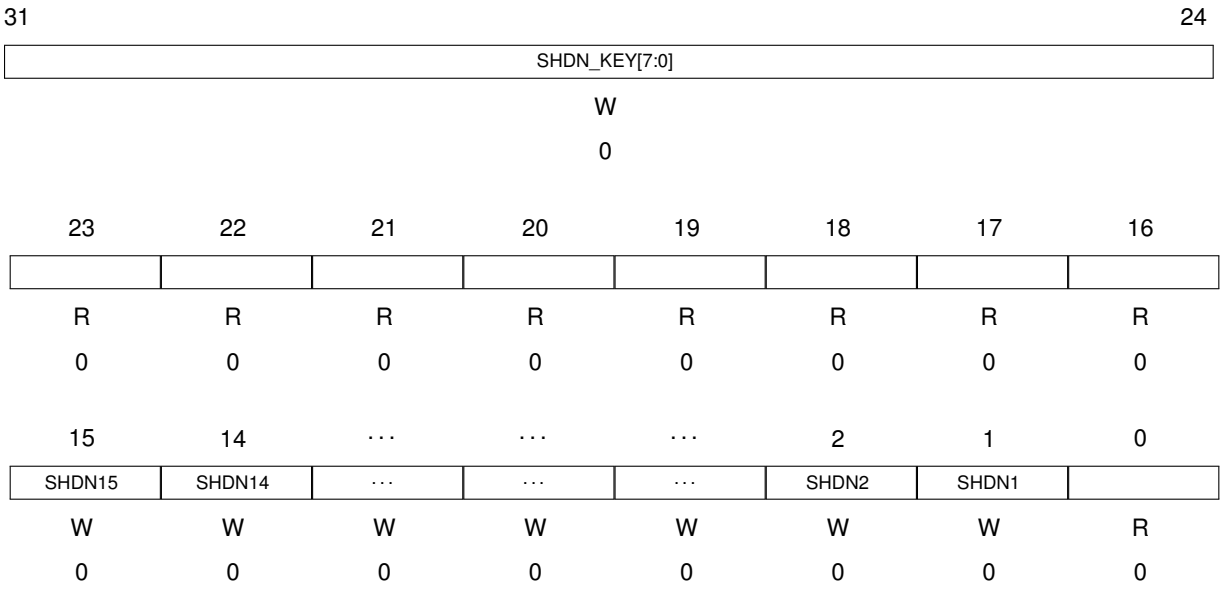
CORE_NUM[31:0] *Core Number*

Stores the number of processor cores.

8.3.5. SHUTDOWN REGISTER

Address: 0x08

Type: shared



SHDN*n* *Shut down Core n*

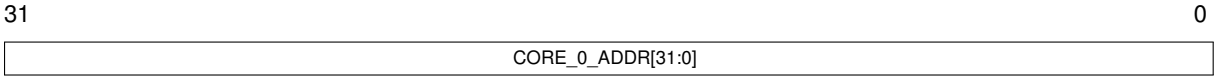
Writing immediately causes Core *n* to shut down. The operation will succeed only when attempting to shut down cores that are currently active and *SHDN_KEY* field is set properly. The operation will fail when bit corresponding to the Core 0 will be set. Use only when necessary. Only necessary bits are implemented.

SHDN_KEY[7:0] *Shutdown Key*

This field has to be set to 0xA5 value in order to unlock shutdown function.

8.3.6. START ADDRESS REGISTERS

Address: 0x10



N/A

0

Address: 0x14

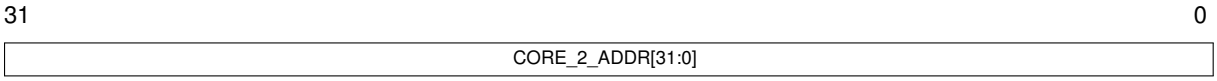
Type: shared

R/W

0

Address: 0x18

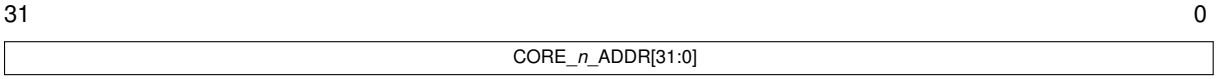
Type: shared



R/W

0

Address: . . .



R/W

0

CORE_*n*_ADDR[31:0] *Core n Start Address*

Stores the Core *n* start address.

8.3.7. CORE RUN REGISTERS

Address: 0xD8

31	30	···	···	8	7	0
		···	···		CORE_0_RUN[7:0]	
R	R	R	R	R	N/A	
0	0	0	0	0	0	

Address: 0xDC

Type: shared

31	30	···	···	8	7	0
		···	···		CORE_1_RUN[7:0]	
R	R	R	R	R	W	
0	0	0	0	0	0	

Address: 0xE0

Type: shared

31	30	···	···	8	7	0
		···	···		CORE_2_RUN[7:0]	
R	R	R	R	R	W	
0	0	0	0	0	0	

Address: . . .

31	30	···	···	8	7	0
		···	···		CORE_*n*_RUN[7:0]	
R	R	R	R	R	W	
0	0	0	0	0	0	

CORE_*n*_RUN[7:0] *Core n Run*

Core *n* begins to operate after writing 0xA5 value to this register.

8.3.8. CORE 0 INJECTION MASK REGISTER

Address: 0x1D8

Type: shared

31	0
INJDATA[31:0]	
R/W	
0	

INJDATA[31:0] *Core 0 Error Injection Mask*

Error injection mask for Core 0. Should be used to self-test of signeture checker. The entered value will be xored with generated Core 0 signature. In dual-core lockstep signatures miscompare will trigger hard error reset.

8.3.9. CORE 1 INJECTION MASK REGISTER

Address: 0x1DC

Type: shared

31	0
INJDATA[31:0]	
R/W	
0	

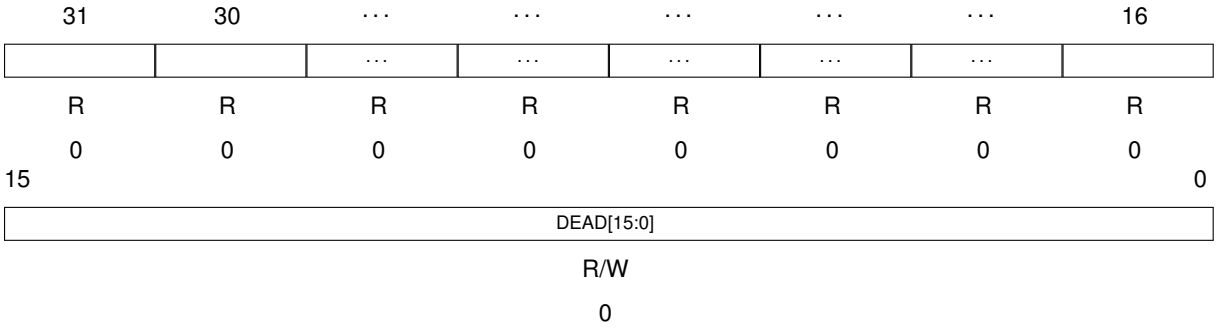
INJDATA[31:0] *Core 1 Error Injection Mask*

Error injection mask for Core 1. Should be used to self-test of signeture checker. The entered value will be xored with generated Core 1 signature. In dual-core lockstep signatures miscompare will trigger hard error reset.

8.3.10. CORE DEADLOCK COUNTER MAX REGISTER

Address: 0x1E0

Type: shared



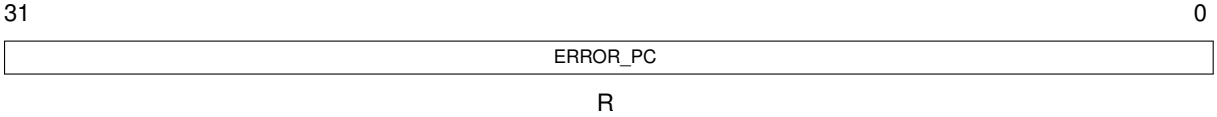
DEAD[15:0] *Core Deadlock Counter Max*

Lockstep deadlock counter maximum value. The register can be written only once after reset. The value greater than zero enables the deadlock counter. The harware counter counts the cores stall cycles. When the maximum valu is reached, the hardware error is issued. Reset cause can be checkek in Reset Cause Register 8.4.2.

8.3.11. ERROR PROGRAM COUNTER REGISTER

Address: 0x1E4

Type: shared



ERROR_PC[31:0] *Error Program Counter*

During lockstep hardware error, this register holds the PC value where the error occured. Value is not cleared after reset and is valid only after lockstep hardware error event.

8.3.12. ERROR STATISTICS REGISTER

Address: 0x1E8

Type: shared

31	0
ERROR_STAT	
R	

ERROR_STAT[31:0] *Error Program Counter*

During lockstep hardware error, this register holds the additional informations regarding encountered error fr further analysis.

8.3.13. LOCKSTEP CONTROL REGISTER

Address: 0x1EC

Type: shared

31	30	...	...	...	...	...	8
		...	...	...	...	...	
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

7	6	5	0
		RNG_THRES[5:0]	
R	R	R/W	
0	0	0	

RNG_THRES[5:0] *Dummy Cycles RNG Threshold*

Propability of dummy cycles injection.

8.4. POWER MANAGEMENT CONTROLLER

Power Management Controller stores the last cause of processor reset and allows to software reset the processor. Detailed registers description is shown below.

8.4.1. REGISTERS LIST

Table 8.3: Power Management Controller registers list.

Address Offset	Register	Name
0xF0020004	RSTRSN	Reset Cause Register
0xF0020008	PWDRST	Reset Register
0xF002000C	DPRST	Deep Reset Register
0xF0020014	INFO	Info Register

8.4.2. RESET CAUSE REGISTER

Address: 0x04

Type: shared

31	30	···	···	···	10	9	7
		···	···	···			DLHARDERR
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
ICHARDERR	DCHARDERR	LSHARDERR		WDTRST	PWDRST	DBGRST	PWRON
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	1

PWRON *Power On Reset*

Set if last reset cause was external reset.

BDGRST *Debug Reset*

Set if last reset was caused by On-Chip Debugger.

PWDRST *Power Management Reset*

Set if last reset was caused by using the *Reset Register* in Power Management Controller.

WDTRST *Watchdog Reset*

Set if last reset was caused by Watchdog.

LSHARDERR *Lockstep Hard Error Reset*

Set if last reset was caused by lockstep signature comparator.

DCHARDERR *Data Cache Hard Error Reset*

Set if last reset was caused by data cache hard error.

ICHARDERR *Instruction Cache Hard Error Reset*

Set if last reset was caused by instruction cache hard error.

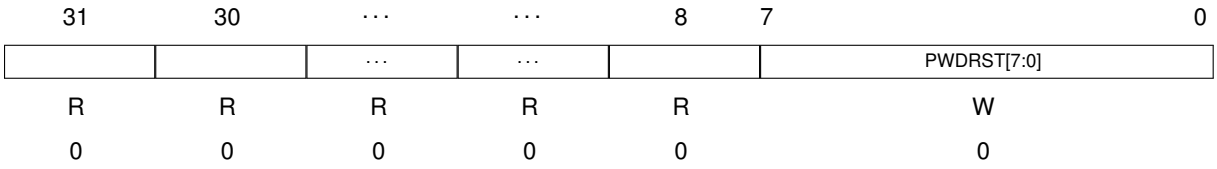
DLHARDERR *Deadlock Hard Error Reset*

Set if last reset was caused by lockstep controller deadlock hard error.

8.4.3. RESET REGISTER

Address: 0x08

Type: shared



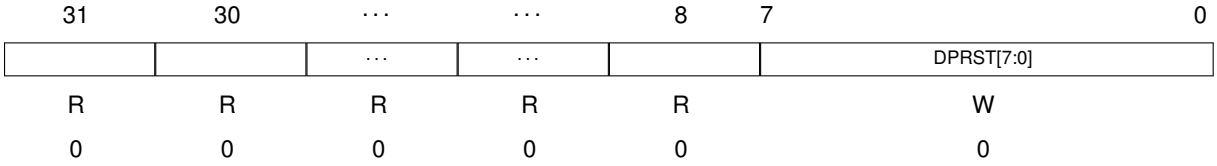
PWDRST[7:0] *Power Management Reset*

Writing 0xA5 value to this field causes processor to reset. The On-Chip Debugger state remains unchanged.

8.4.4. DEEP RESET REGISTER

Address: 0x0C

Type: shared



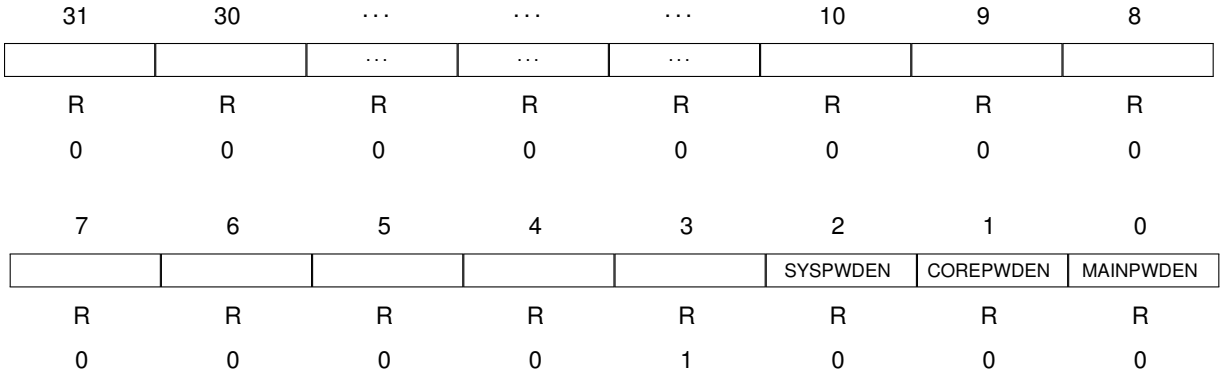
DPRST[7:0] *Power Management Deep Reset*

Writing 0xA5 value to this field causes processor to deep reset. This register resets also the On-Chip Debugger.

8.4.5. INFO REGISTER

Address: 0x14

Type: shared



MAINPWDEN *Main Core Power Down Enable*

Bit is set in multicore system that allows to power down main core.

COREPWDEN *Core Power Down Enable*

Bit is set if system allows to shut down processor core.

SYSPWDEN *System Power Down Enable*

Bit is set if System Power Down feature is enabled.

COREPRES *Core Clock Prescaler Enable*

Bit is set if core clock prescaler is present.

PER0PRES *Peripheral Clock Prescaler Enable*

Bit is set if peripheral clock prescaler is present.

8.5. CSR CONTROLLER

CSR Controller provides Machine-level access to all implemented CRSs in a memory-mapped manner. The memory-mapped address of particular CSR register can be calculated using the following formula: CSR_ADDRESS = 0xF0030000 + (CSR_NUMBER * 4).

8.5.1. STACK PROTECTION UNIT

Stack Protection Unit continuously monitors load and store operations with GPR[2] (Stack Pointer Register) used as a base register. In such a case, when generated address is out of given bounds the stack exception will be raised. Figure 8.2 presents the simplified stack exception generation datapath.

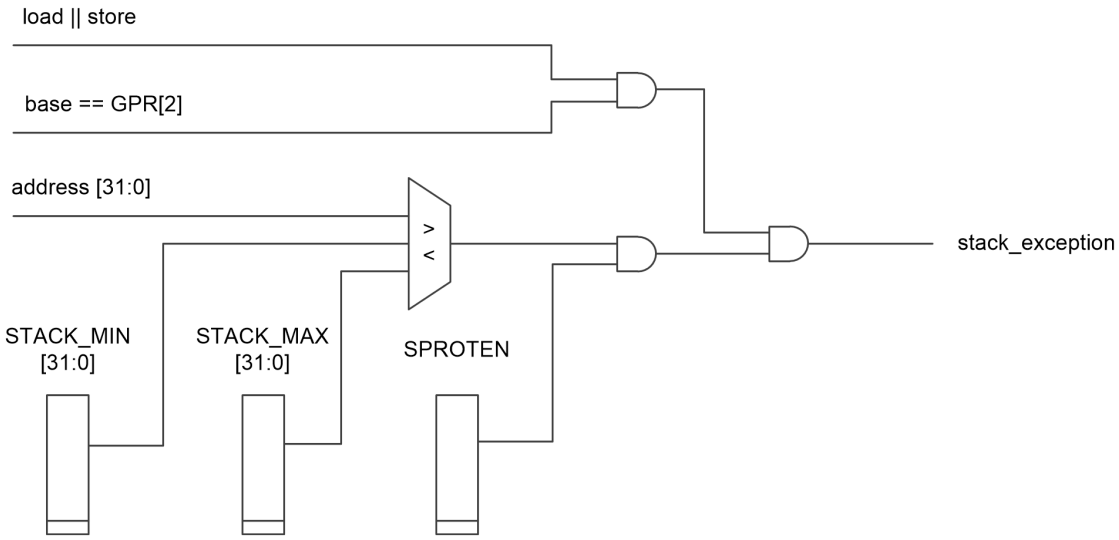


Figure 8.2: Simplified stack exception generation datapath.

8.5.2. NON-MASKABLE INTERRUPT

Non-maskable interrupt (NMI) is used for hardware error conditions, and cause an immediate jump to interrupt handler (mtvec register) running in M-mode regardless of the state of core's interrupt enable bit. The mepc register is written with the virtual address of the instruction that was interrupted, and mcause is set to a value 0xFFFF indicating the NMI. The NMI can thus overwrite state in an active machine-mode interrupt handler.

To be able to recover from NMI, mepc, mcause and mscratch registers as well as mpp and mpie fileds from mstatus register are backed up during entering the NMI. The CSR controller will not allow to issue another NMI during the execution of NMI handler. Execution of MRET instruction will restore backed up values and re-enable NMI.

8.6. PLATFORM-LEVEL INTERRUPT CONTROLLER

Platform-Level Interrupt Controller prioritizes and distributes global interrupts in a processor system. Figure 8.3 provides a quick overview of PLIC operation. The PLIC connects global interrupt sources to interrupt targets, which are usually hart contexts. The PLIC contains multiple interrupt gateways, one per interrupt source, together with a PLIC core that performs interrupt prioritization and routing. Each interrupt source is assigned a separate priority. The PLIC core contains a matrix of interrupt enable (IE) bits to select the interrupts that are enabled for each target. The PLIC core forwards an interrupt notification to one or more targets if the targets have any pending interrupts enabled, and the priority of the pending interrupts exceeds a per-target threshold. When the target takes the external interrupt, it sends an interrupt claim request to retrieve the identifier of the highest-priority global interrupt source pending for that target from the PLIC core, which then clears the corresponding interrupt source pending bit. After the target has serviced the interrupt, it sends the associated interrupt gateway an interrupt completion message and the interrupt gateway can now forward another interrupt request for the same source to the PLIC.

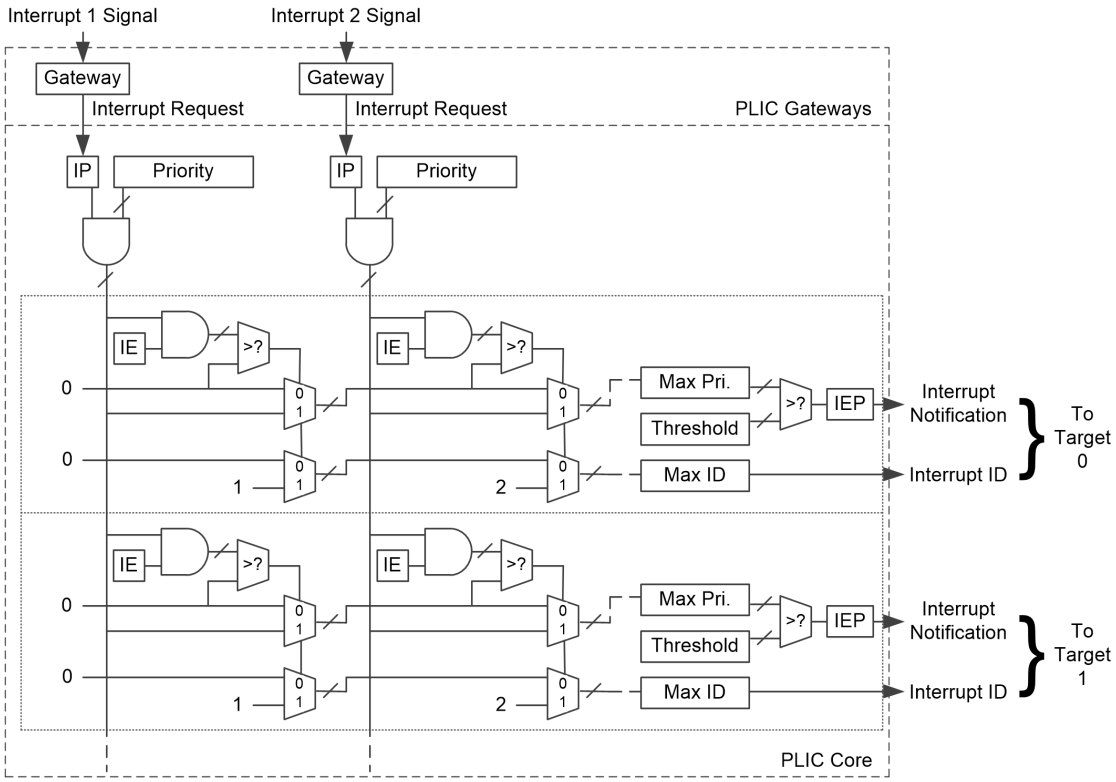


Figure 8.3: Platform-Level Interrupt Controller (PLIC) conceptual block diagram.

8.6.1. REGISTERS LIST

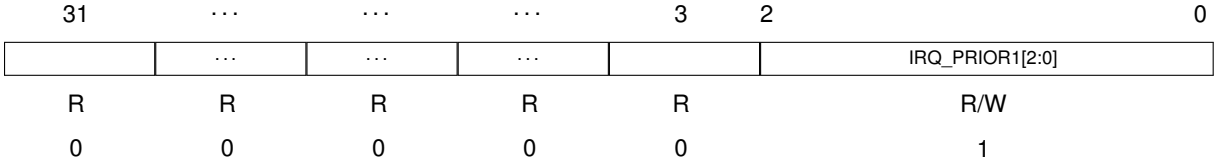
Table 8.4: Platform-Level Interrupt Controller registers list.

Address Offset	Register	Name
0xF0038004	PRIOR_1	Interrupt Priority Register 1
0xF0038008	PRIOR_2	Interrupt Priority Register 2
...	...	...
0xF003803C	PRIOR_15	Interrupt Priority Register 15
0xF0038080	PENDING	Interrupt Pending Register
0xF0038084	ENABLE	Interrupt Enable Register
0xF0038088	THRESHOLD	Interrupt Threshold Register
0xF003808C	CLAIM	Interrupt Claim Register

8.6.2. INTERRUPT PRIORITY REGISTERS

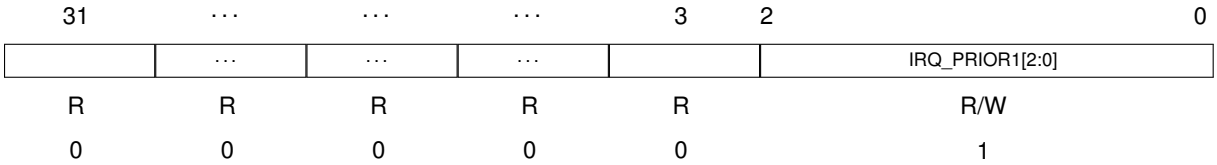
Address: 0x04

Type: shared

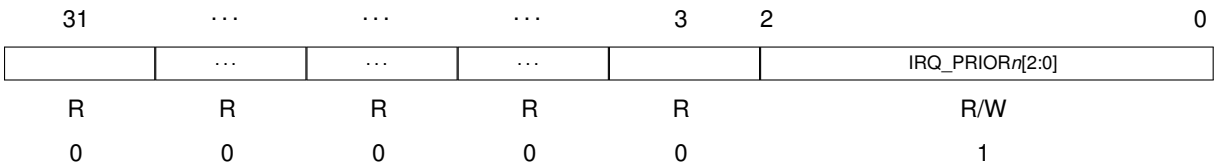


Address: 0x08

Type: shared



Address: ...



IRQ_PRIOR*n*[2:0] *Interrupt Priority n Register*

Stores the priority of consecutive interrupt sources from 1 to 15. Allowed values start from 1 (lowest priority) to 7 (highest priority). Value 0 is reserved.

8.6.3. INTERRUPT PENDING REGISTER

Address: 0x80

Type: exclusive

31	30	···	···	···	18	17	16
		···	···	···			
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

15	···	···	···	3	2	1	0
IP15	···	···	···	IP3	IP1	IP0	
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

IP*n* *Interrupt Pending Source n*

Set if interrupt source *n* is pending.

8.6.4. INTERRUPT ENABLE REGISTER

Address: 0x84

Type: exclusive

31	30	···	···	···	18	17	16
		···	···	···			
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

Address: ...

15	···	···	···	3	2	1	0
IE15	···	···	···	IE3	IE1	IE0	
R/W	R	R	R	R/W	R/W	R/W	R
0	0	0	0	0	0	0	0

IE*n* *Interrupt Enable Source n*

Setting bit IE*n* enables corresponding interrupt source.

8.6.5. INTERRUPT THRESHOLD REGISTERS

Address: 0x88

Type: exclusive

31	...	...	...	3	2	0
	...	...	...		THRESHOLD[2:0]	
R	R	R	R	R	R/W	
0	0	0	0	0	0	

THRESHOLD[2:0] *Interrupt Threshold Register*

Pending interrupt sources which priorities exceed threshold are forwarded to the target logic. Threshold value 0 means that every interrupt will be forwarded and value 7 means that no interrupt will be forwarded.

8.6.6. INTERRUPT CLAIM REGISTERS

Address: 0x8C

Type: exclusive

31	...	...	...	3	2	0
	...	...	...		CLAIM[2:0]	
R	R	R	R	R	R/W	
0	0	0	0	0	0	

CLAIM[2:0] *Interrupt Claim Register*

Reading this register will retrieve the identifier of the highest-priority global interrupt source pending the target from the PLIC core, which then clears the corresponding interrupt source pending bit. Writting the same value as previously read sends the associated interrupt gateway an interrupt completion message and the interrupt gateway can now forward another interrupt request for the same source to the PLIC.

8.7. INTER-CORE INTERRUPT CONTROLLER

Inter-Core Interrupt Controller is a device that can be used to send interrupt request from one core to another. Interrupt trigger register is used to write cores mask that will receive interrupt. Triggered core will see flag register containing mask of cores that triggered its interrupt. Interrupt flags has to be cleared by software.

8.7.1. REGISTERS LIST

Table 8.5: Inter-Core Interrupt Controller registers list.

Address Offset	Register	Name
0xF0038200	ICORE_IRQ_MAP	Inter-Core Interrupt Mapping Register
0xF0038204	ICORE_IRQ_TRIG	Inter-Core Interrupt Trigger Register
0xF0038208	ICORE_IRQ_FLAG	Inter-Core Interrupt Flags Register

8.7.2. INTER-CORE INTERRUPT MAPPING REGISTER

Address: 0x00

Type: shared

31	30	···	···	···	···	17	16
		···	···	···	···		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	IRQ11	IRQ10	IRQ9	IRQ8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
1	0	0	0	0	0	0	0

7	6	5	4	3	2	1	0
IRQ7	IRQ6	IRQ5	IRQ4	IRQ3	IRQ2	IRQ1	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
0	0	0	0	0	0	0	0

IRQ*n* *Inter-Core Interrupt Mapping n*

Stores the number of interrupt that will be recorded on inter-core interrupt event.

8.7.3. INTER-CORE INTERRUPT TRIGGER REGISTER

Address: 0x04

Type: exclusive

31	30	···	···	···	2	1	0
CT31	CT30	···	···	···	CT2	CT1	CT0
W	W	W	W	W	W	W	W
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

CT*n* *Core n Inter-Core Interrupt Trigger*

Setting bit CT*n* causes Core *n* to receive an inter-core interrupt. Only necessary bits are implemented.

8.7.4. INTER-CORE INTERRUPT FLAGS REGISTER

Address: 0x08

Type: exclusive

31	30	···	···	···	2	1	0
CTF31	CTF30	···	···	···	CTF2	CTF1	CTF0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

CTF*n* *Core n Trigger Flag*

Bit CTF*n* is set if Core *n* triggered an inter-core interrupt on this core. Only necessary bits are implemented. Flags has to be cleared manually by writing one to corresponding bits.

8.8. CORE LOCAL INTERRUPTOR CONTROLLER

The CLINT block holds memory-mapped control and status registers associated with software and timer interrupts.

8.8.1. REGISTERS LIST

Table 8.6: Core Local Interruptor Controller registers list.

Address Offset	Register	Name
0xF003C000	MSIP0	Machine Software Interrupt Pending 0
0xF003C004	MSIP1	Machine Software Interrupt Pending 1
...	...	...
0xF003C080	MTIMECMP_LO_0	Machine Time Compare Low Register 0
0xF003C084	MTIMECMP_HI_0	Machine Time Compare High Register 1
0xF003C088	MTIMECMP_LO_1	Machine Time Compare Low Register 0
0xF003C08C	MTIMECMP_HI_1	Machine Time Compare High Register 1
...	...	...
0xF003C180	MTIME_LO	Machine Time Low Register
0xF003C184	MTIME_HI	Machine Time High Register
0xF003C188	MTIMECFG	Machine Time Config Register

8.8.2. MACHINE SOFTWARE INTERRUPT PENDING

Address: 0x000

Type: shared

31	30	...	...	...	2	1	0
		...	...	...			MSIP
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0

MSIP *Machine Software Interrupt Pending*

Value is reflected in the MSIP bit of the MIP CSR. Harts may write each other's MSIP bits to effect interprocessor interrupts.

8.8.3. MACHINE TIME COMPARE

Address: 0x080

Type: shared

31	0
MTIMECMP_LO[31:0]	
R/W	
0	

Address: 0x084

Type: shared

31	0
MTIMECMP_HI[31:0]	
R/W	
0	

MTIMECMP[63:0] *Machine Time Compare*

Memory-mapped machine-mode timer compare register, which causes a timer interrupt to be posted when the MTIME register contains a value greater than or equal to the value in the MTIMECMP register. The interrupt remains posted until it is cleared by writing the MTIMECMP register. The interrupt will only be taken if interrupts are enabled and the MTIE bit is set in the MIE register. The implemented number of bits can be checked in the MTIMECFG register.

8.8.4. MACHINE TIME REGISTER

Address: 0x0180

Type: exclusive

31	0
MTIME_LO[31:0]	
R/W	
0	

Address: 0x0184

Type: exclusive

31	0
MTIME_HI[31:0]	
R/W	
0	

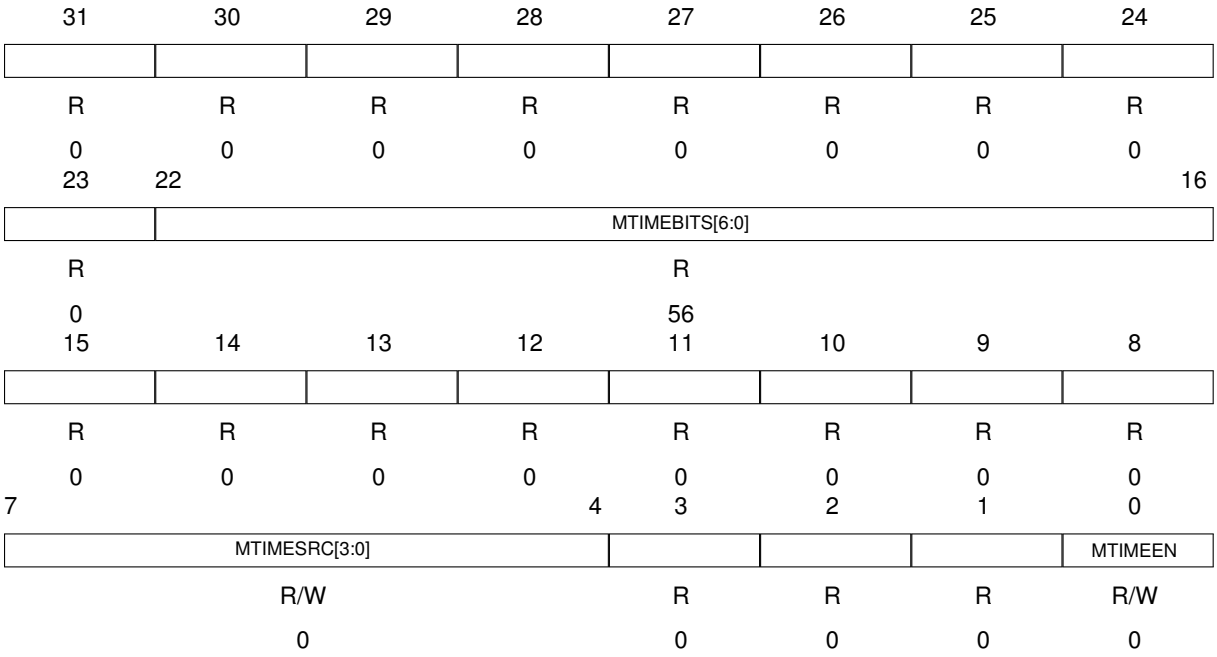
MTIME[63:0] *Machine Time Register*

Timer counter value. The implemented number of bits can be checked in the MTIMECFG register.

8.8.5. MACHINE TIME CONFIG REGISTER

Address: 0x188

Type: exclusive



MTIMEEN *Machine Time Enable*

Enable Machine Time Counter.

MTIMESRC[3:0] *Machine Time Clock Source*

Machine Time clock source:

0 Processor core clock.

1 GNSS internal AFE clock.

10 GNSS AUX AFE clock.

15 GNSS Virtual AFE clock.

MTIMEBITS[6:0] *Machine Timer Counter Bits*

Number of implemented Machine Timer Counter bits.

8.9. INSTRUCTION CACHE CONTROLLER

The instruction cache is a small, fast memory which stores copies of program instructions from frequently used main memory locations. The addresses seen by the processor core are divided into tag, index and offset bits. The index is used to select the set in the cache, therefore only a limited number of cache lines with the same index part can be stored at one time in the cache. The tag is stored in the cache and compared upon read. Figures 8.4 and 8.5 presents the cache address mapping examples.

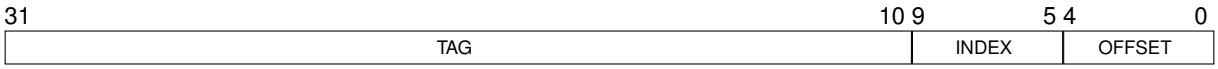


Figure 8.4: 1 KiB/way, 32 bytes/line address mapping example.

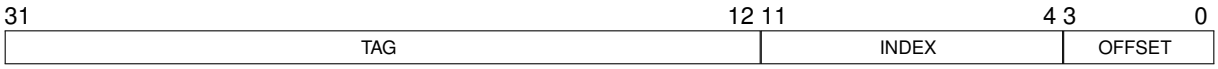


Figure 8.5: 4 KiB/way, 16 bytes/line address mapping example.

Figures 8.6 and 8.7 presents the single cache tag entry examples. The first bit is used to determine if set is valid. The LRR bit is used in Least Recently Replaced algorithm. The remaining bits store the tag. When a read from cache is performed, the tags and data for all cache ways of the corresponding set are read out in parallel, the tags and valid bits are compared to the desired address and the matching way is selected. In the hit case, the instruction cache can deliver single instruction every clock cycle. In the miss case, the processor core will be stalled till the entire instruction cache line will be replaced.

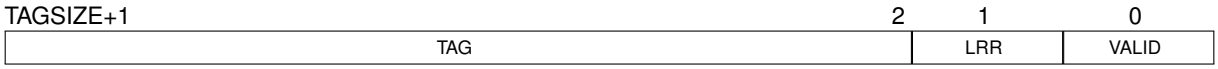


Figure 8.6: Tag memory entry with LRR bit.

Figure 8.7: Tag memory entry without LRR bit.

When the instruction cache is disabled, every time the processor requests for new instruction, the whole cache line is read. However only the demanding instruction is forwarded to the processor and cache tag

memory is not updated. Because of high performance penalty caches should be disabled only when necessary.

Processor core instruction cache is disabled after reset. Before it can be enabled for the first time or again after disabling, the flush must be done using *Flush Register*. Instruction caches are automatically disabled after the corresponding processor core has stopped.
Detailed registers description is shown below.

8.9.1. PARITY SUPPORT

Instruction Cache Controller supports parity module that contains coder/decoder circuitry for parity bits calculation. Coder module appends parity bits to the data word (both memory and tag), while the decoder checks stored data agains erros. One parity bit is generated for each data byte, what means capability of single error detection per data byte. Errors are counted and causes controller to re-fetch the data line. If the retry count reaches programmed value, hardware reset error (Reset Cause Register 8.4.2) or NMI is issued, depending on the configuration.

8.9.2. MEMORY SCRAMBLING

Instruction Cache Controller supports memory scrambling. Each time the cache is flushed, controller generates new patterns that will be used to scramble memory and tags data. Pattern is generated using internal LFSR register that can be aided with external TRNG.

8.9.3. REGISTERS LIST

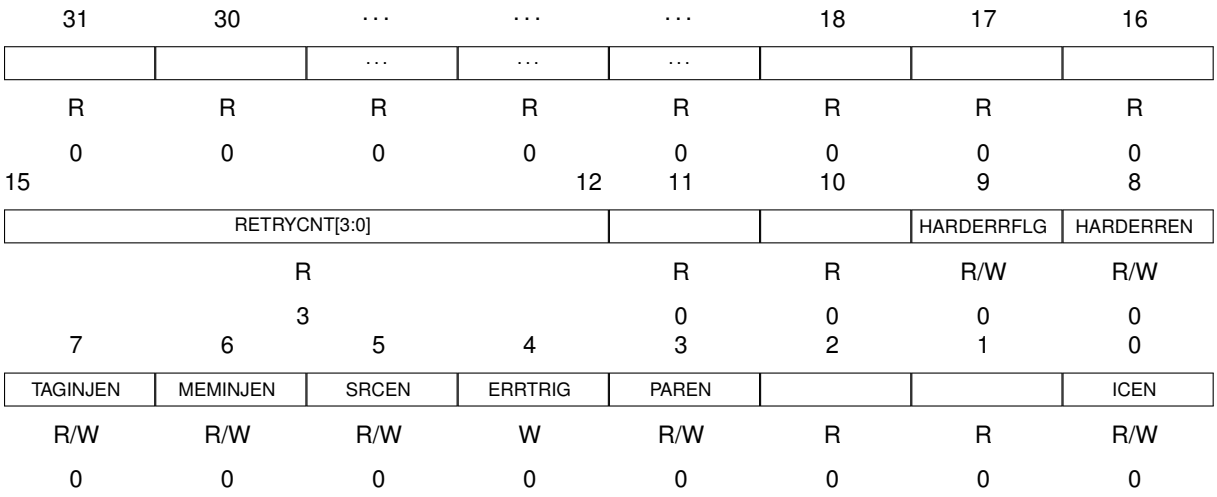
Table 8.7: Instruction Cache Controller registers list.

Address Offset	Register	Name
0xF0070000	STCR	Status and Control Register
0xF0070004	FLUSH	Flush Register
0xF0070008	INFO	Info Register
0xF007000C	ERR_CNT_0	Error Counter 0
0xF0070010	ERR_CNT_1	Error Counter 1
0xF0070014	INJECT_MASK_LO	Error Injection Mask Low
0xF0070018	INJECT_MASK_HI	Error Injection Mask High

8.9.4. STATUS AND CONTROL REGISTER

Address: 0x00

Type: exclusive



ICEN *Instruction Cache Enable*

Setting this bit enables instruction cache.

PAREN *Parity Enable*

Enable parity control logic.

ERRTRIG *Error Counter Trigger*

Used to synchronize error counters in lockstep mode.

SRCEN *Scrambling Enable*

Enable scrambling logic.

MEMINJEN *Data Memory Error Injection Enable*

Enable error injection to data memory.

TAGINJEN *Tag Memory Error Injection Enable*

Enable error injection to tag memory.

HARDERREN *Hard Error Enable*

Enable hard error generation. This field is shared across cores.

HARDERRFLG *Hard Error Flag*

Enable hard error flag.

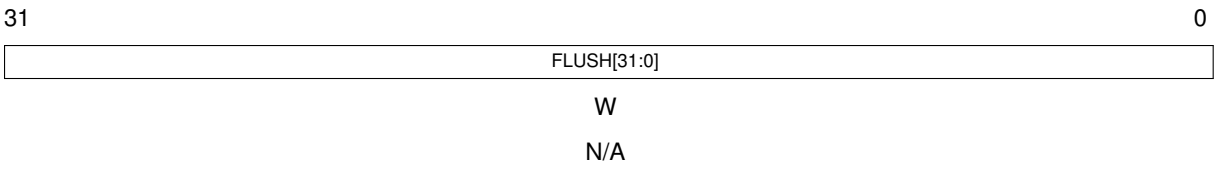
RETRYCNT[3:0] *Retry Count*

Number of retries of instruction cache line fetch. This field is shared across cores.

8.9.5. FLUSH REGISTER

Address: 0x04

Type: exclusive



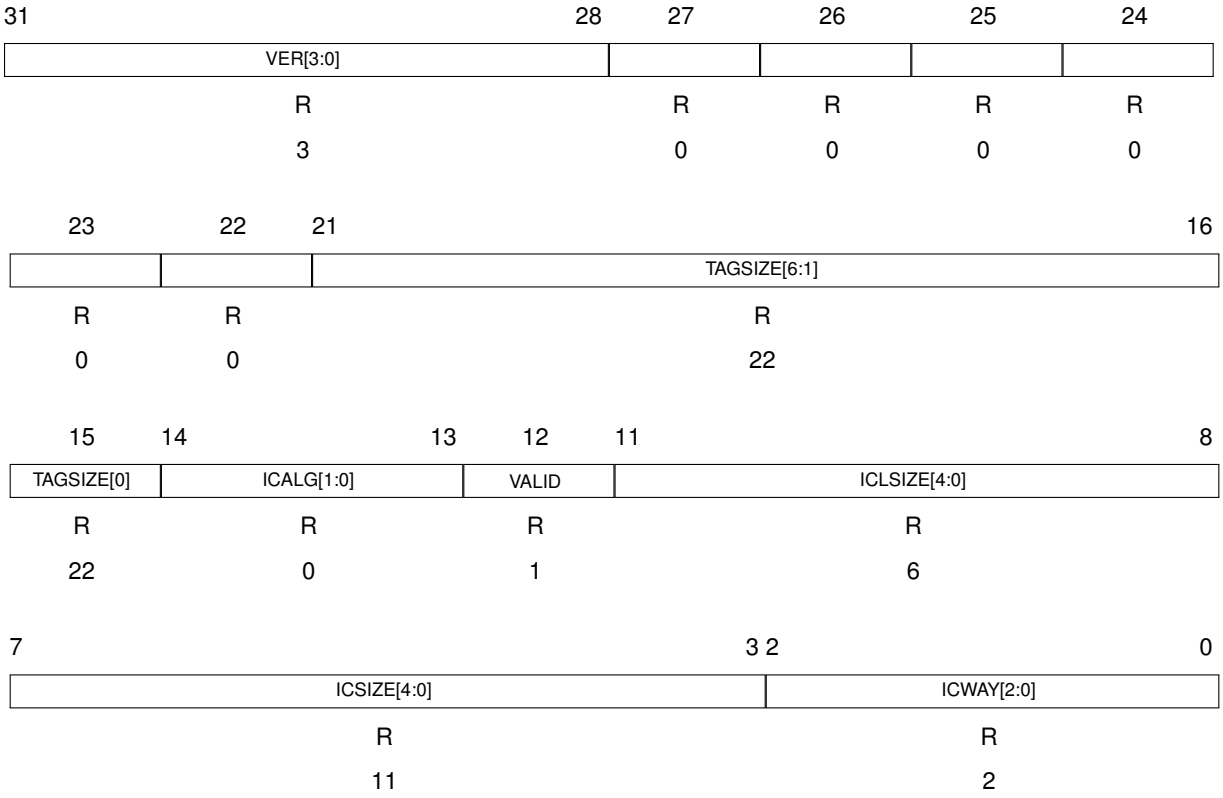
FLUSH[31:0] *Flush Instruction Tag Memory*

Writing any value to this register will start flush operation of instruction cache tag memory.

8.9.6. INFO REGISTER

Address: 0x08

Type: shared



ICWAY[2:0] *Instruction Cache Ways*

Number of instruction cache ways.

ICSIZE[4:0] *Instruction Cache Size*

Size of instruction cache way – $2^{ICSIZE}b$.

ICLSIZE[3:0] *Instruction Cache Line Size*

Size of instruction cache line – $2^{ICLSIZE}b$.

VALID *Valid Bit*

Valid bit.

ICALG *Replacement Algorithm*

Line replacement algorithm in multiway instruction cache configuration:

0 Pseudo-random,

1 Least Recently Replaced.

TAGSIZE[6:0] *Tag Address Part Size*

Number of address bits in instruction cache tag memory record.

VER[3:0] *Instruction Cache Version*

Implemented instruction cache version:

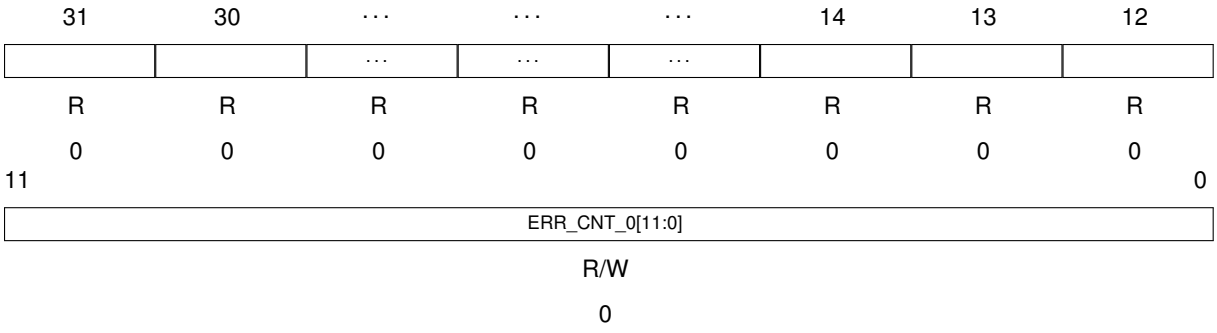
0 High-performance.

3 Fault-tolerant.

8.9.7. PARITY ERROR COUNT 0

Address: 0x0C

Type: exclusive



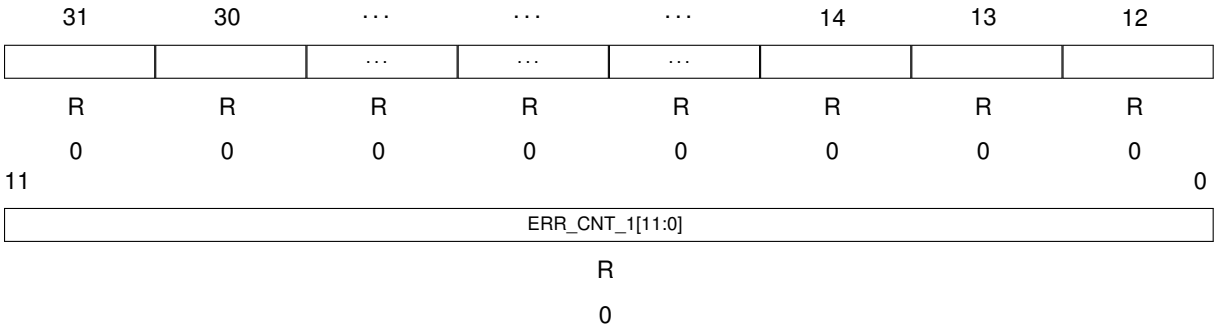
ERR_CNT_0[11:0] *Core 0 Parity Error Count*

Saturating counter of encountered parity errors of Core 0.

8.9.8. PARITY ERROR COUNT 1

Address: 0x10

Type: shared



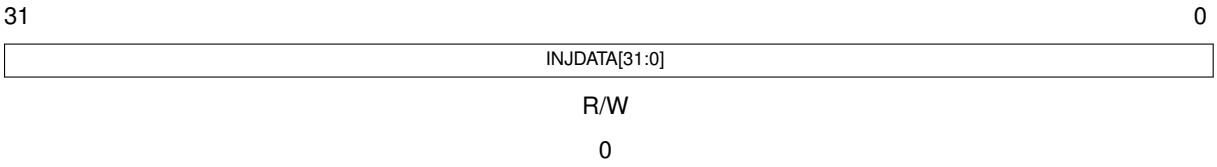
ERR_CNT_1[11:0] *Core 1 Parity Error Count*

Saturating counter of encountered parity errors of Core 1.

8.9.9. ERROR INJECTION MASK LOW

Address: 0x14

Type: exclusive



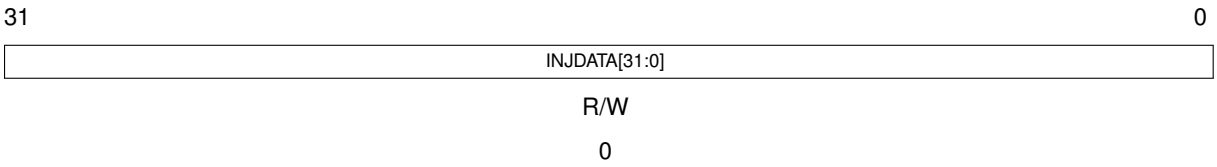
INJDATA[31:0] *Error Injection Data Low*

Low part of error injection data. Should be used to self-test of parity controller. Both low and hi part registers forms Error Injection Data that is xored with data to be written to corresponding memory. If MEMINJEN is set, data is injected to the memory. If TAGINJEN is set, data is injected to the tags.

8.9.10. ERROR INJECTION MASK HIGH

Address: 0x18

Type: exclusive



INJDATA[31:0] *Error Injection Data High*

Low part of error injection data. Should be used to self-test of parity controller. Both low and hi part registers forms Error Injection Data that is xored with data to be written to corresponding memory. If MEMINJEN is set, data is injected to the memory. If TAGINJEN is set, data is injected to the tags.

8.10. DATA CACHE CONTROLLER

The data cache is a small, fast memory which stores copies of program data from frequently used main memory locations. The addresses seen by the processor core are divided into tag, index and offset bits. The index is used to select the set in the cache, therefore only a limited number of cache lines with the same index part can be stored at one time in the cache. The tag is stored in the cache and compared upon read. Figures 8.8 and 8.9 presents the cache address mapping examples.

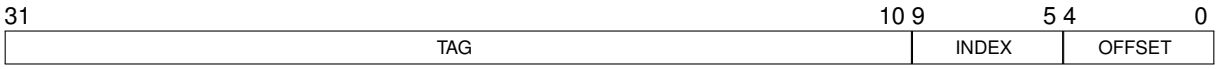


Figure 8.8: 1 KiB/way, 32 bytes/line address mapping example.

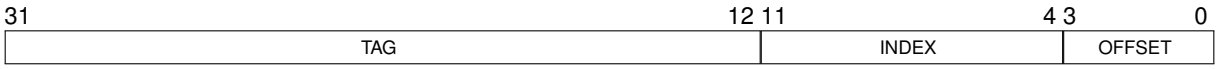


Figure 8.9: 4 KiB/way, 16 bytes/line address mapping example.

Figures 8.10 and 8.11 presents the single cache tag entry examples. The first bit is used to determine if set is valid. The LRR bit is used in Least Recently Replaced algorithm. The remaining bits store the tag. When a read from cache is performed, the tags and data for all cache ways of the corresponding set are read out in parallel, the tags and valid bits are compared to the desired address and the matching way is selected. In the hit case, the data cache can deliver single 32-bit word every clock cycle. In the miss case, the processor core will be stalled till the entire data cache line will be replaced.

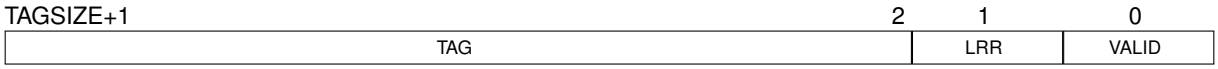


Figure 8.10: Tag memory entry with LRR bit.

Figure 8.11: Tag memory entry without LRR bit.

When the data cache is disabled, every time the processor requests for new data from cachable region, the whole cache line is read. However only the demanding data is forwarded to the processor and cache

tag memory is not updated. Because of high performance penalty caches should be disabled only when necessary.

Processor core data cache is disabled after reset. Before it can be enabled for the first time or again after disabling, the flush must be done using *Flush Register*. Data caches are automatically disabled after the corresponding processor core has stopped.
Detailed registers description is shown below.

8.10.1. EDAC SUPPORT

Data cache controller supports EDAC (Error Detection And Correction) module that contains coder/decoder circuitry for parity bits calculation. Coder module appends parity bits to the data word (both memory and tag), while the decoder checks stored data agains erros. The EDAC controller supports SEC-DED, what means capability of correcting single error and detection of double error in a data block. Single errors are counted and corrected transparently to the software. Double errors are counted and can cause hardware reset error (Reset Cause Register 8.4.2) or issue NMI, depending on the configuration.

8.10.2. MEMORY SCRAMBLING

Data cache controller supports memory scrambling. Each time the cache is flushed, controller generates new patterns that will be used to scramble memory and tags data. Pattern is generated using internal LFSR register that can be aided with external TRNG.

8.10.3. REGISTERS LIST

Table 8.8: Data Cache Controller registers list.

Address Offset	Register	Name
0xF0072000	STCR	Status and Control Register
0xF0072004	FLUSH	Flush Register
0xF0072008	INFO	Info Register
0xF007200C	ERR_CNT_0	Error Counter 0
0xF0072010	ERR_CNT_1	Error Counter 1
0xF0072014	INJECT_MASK_LO	Error Injection Mask Low
0xF0072018	INJECT_MASK_HI	Error Injection Mask High
0xF007201C	ERR_ADDR	Last Error Address
0xF0072020	ERR_STAT	Last Error Statistics

8.10.4. STATUS AND CONTROL REGISTER

Address: 0x00

Type: exclusive

31	30	···	···	···	18	17	16
		···	···	···			
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
15	14	13	12	11	10	9	8
						HARDERRFLG	HARDERREN
R	R	R	R	R	R	R/W	R/W
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
TAGINJEN	MEMINJEN	SRCEN	ERRTRIG	ECCEN	BUSY	FLUSH	ICEN
R/W	R/W	R/W	W	R/W	R	R	R/W
0	0	0	0	0	0	0	0

DCEN *Data Cache Enable*

Setting this bit enables data cache.

FLUSH *Data Cache Flush*

Bit indicates if data cache flush is in progress.

BUSY *Write Buffer Busy*

Bit is set if data cache write buffer is busy.

ECCEN *ECC Enable*

Enable ECC control logic.

ERRTRIG *Error Counter Trigger*

Used to synchronize error counters in lockstep mode.

SRCEN *Scrambling Enable*

Enable scrambling logic.

MEMINJEN *Data Memory Error Injection Enable*

Enable error injection to data memory.

TAGINJEN *Tag Memory Error Injection Enable*

Enable error injection to tag memory.

HARDERREN *Hard Error Enable*

Enable hard error generation. With hardware error generation enabled, detected double error with cause hardware error generation. When hardware error generation is disabled, double error issues NMI.

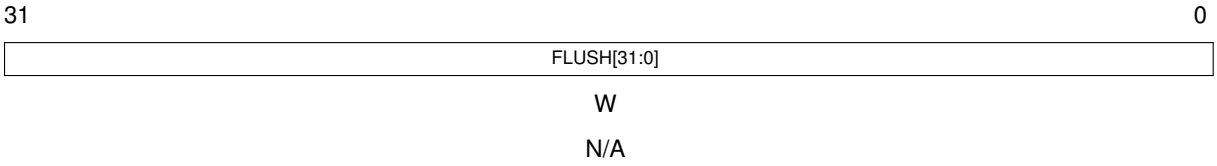
HARDERRFLG *Hard Error Flag*

Hard error flag.

8.10.5. FLUSH REGISTER

Address: 0x04

Type: exclusive



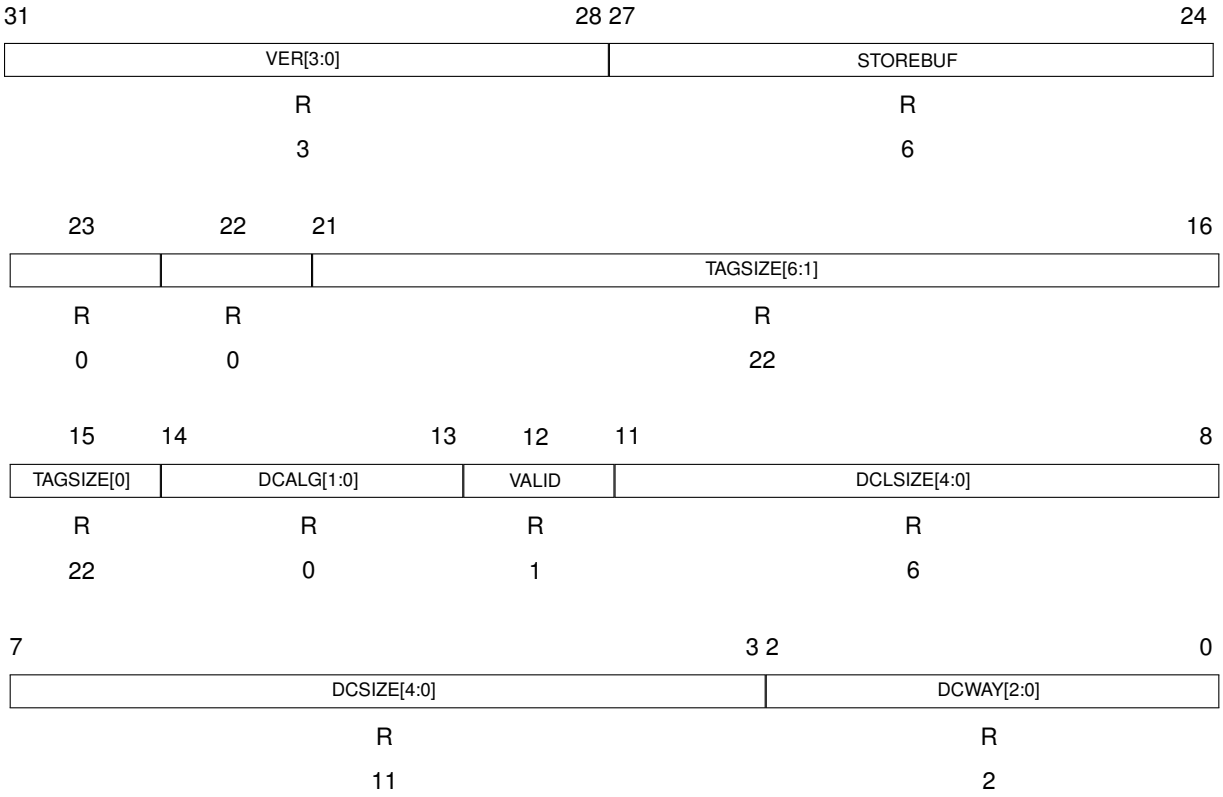
FLUSH[31:0] *Flush Data Tag Memory*

Writing any value to this register will start flush operation of data cache tag memory.

8.10.6. INFO REGISTER

Address: 0x08

Type: shared



DCWAY[2:0] *Data Cache Ways*

Number of data cache ways.

DCSIZE[4:0] *Data Cache Size*

Size of data cache way – $2^{DCSIZE}b$.

DCLSIZE[3:0] *Data Cache Line Size*

Size of data cache line – $2^{DCLSIZE}b$.

VALID *Valid Bit*

Valid bit.

DCALG *Replacement Algorithm*

Line replacement algorithm in multiway data cache configuration:

0 Pseudo-random,

1 Least Recently Replaced.

TAGSIZE[6:0] *Tag Address Part Size*

Number of address bits in data cache tag memory record.

STOREBUF[3:0] *Store Buffer Size*

Number of store buffer entries.

VER[3:0] *Data Cache Version*

Implemented data cache version:

0 High-performance.

1 High-speed.

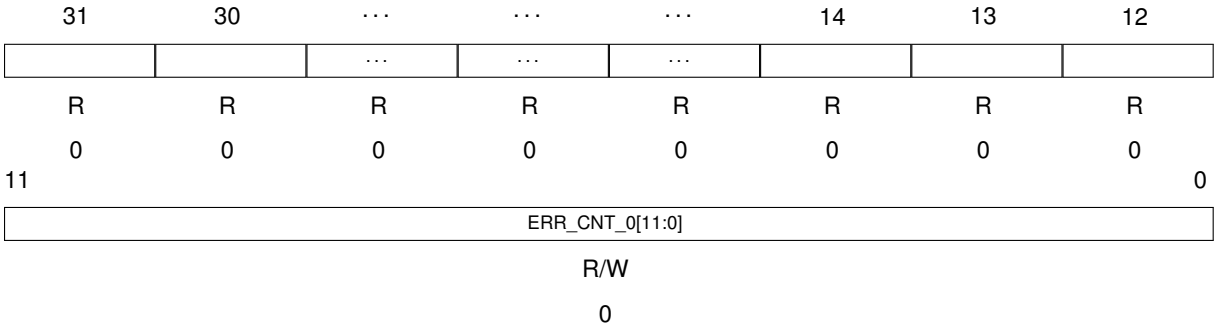
2 Low-power.

3 Fault-tolerant.

8.10.7. ECC ERROR COUNT 0

Address: 0x0C

Type: exclusive



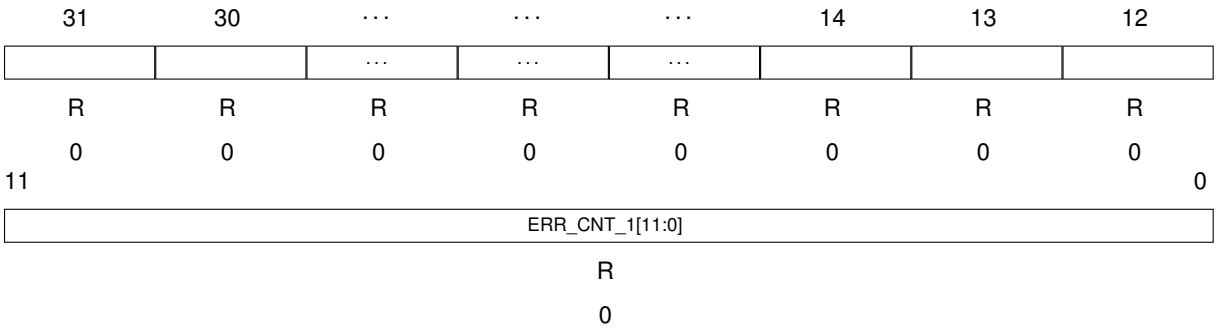
ERR_CNT_0[11:0] *Core 0 ECC Error Count*

Saturating counter of encountered ECC errors of Core 0.

8.10.8. ECC ERROR COUNT 1

Address: 0x10

Type: shared



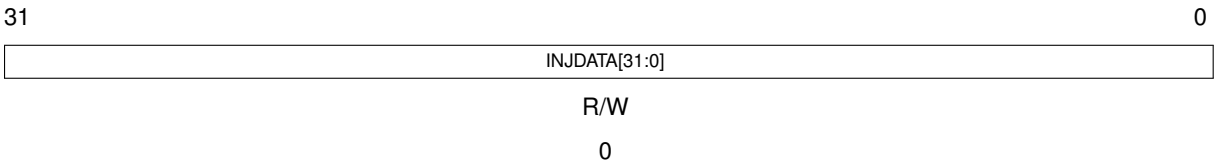
ERR_CNT_1[11:0] *Core 1 ECC Error Count*

Saturating counter of encountered ECC errors of Core 1.

8.10.9. ERROR INJECTION MASK LOW

Address: 0x14

Type: exclusive



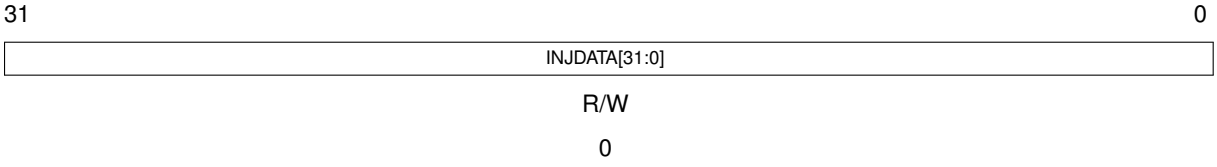
INJDATA[31:0] *Error Injection Data Low*

Low part of error injection data. Should be used to self-test of EDAC controller. Both low and hi part registers forms Error Injection Data that is xored with data to be written to corresponding memory. If MEMINJEN is set, data is injected to the memory. If TAGINJEN is set, data is injected to the tags.

8.10.10. ERROR INJECTION MASK HIGH

Address: 0x18

Type: exclusive



INJDATA[31:0] *Error Injection Data High*

High part of error injection data. Should be used to self-test of EDAC controller. Both low and hi part registers forms Error Injection Data that is xored with data to be written to corresponding memory. If MEMINJEN is set, data is injected to the memory. If TAGINJEN is set, data is injected to the tags.

8.10.11. LAST ERROR ADDRESS

Address: 0x1C

31	0
ERR_ADDR[31:0]	
R	
0	

ERR_ADDR[31:0] *Error Address*

Error injection address.

8.10.12. LAST ERROR STATISTICS

Address: 0x20

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
		...	...	DBL	PORT	MEM_TAG	RDWR
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

RDWR *Read/Write Access*

0 Read Access.

1 Write Access.

MEM_TAG *Memory/Tag Access*

0 Tag Access.

1 Memory Access.

PORT *Tag Port*

0 Tag Port0.

0 Tag Port1.

DBL *Double Error*

0 Single Error.

1 Double Error.

8.11. DEBUG INTERFACE CONTROLLER

The Debug Interface Controller allows running software to execute read/write command from the On-Chip Debugger perspective. Running command will force processor to the debug mode, cause the On-Chip Debugger to execute command and resume work.

8.11.1. REGISTERS LIST

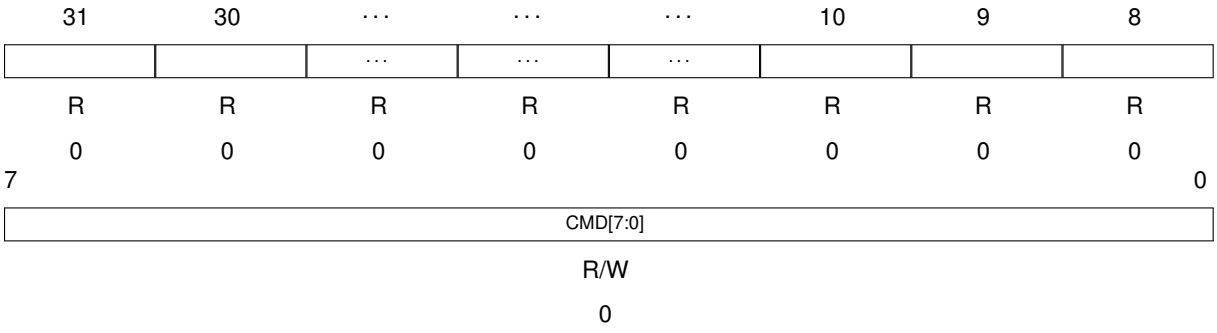
Table 8.9: Debug Interface Controller registers list.

Address Offset	Register	Name
0xF00B0200	CMD	Command Register
0xF00B0204	ADDR	Address Register
0xF00B0208	WDATA	Write Data Register
0xF00B020C	RDATA	Read Data Register

8.11.2. COMMAND REGISTER

Address: 0x00

Type: shared



CMD[7:0] *Command*

Executed command:

0 Idle.

109 Read.

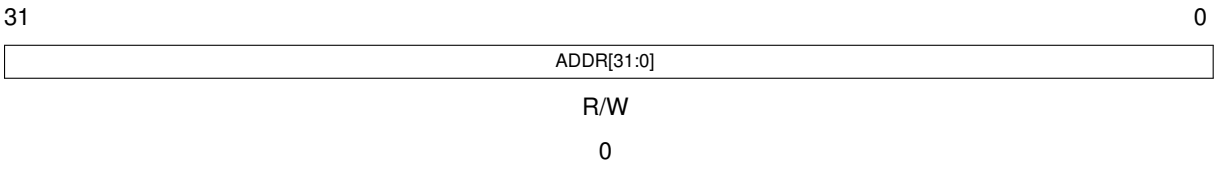
119 Write.

160 Context.

8.11.3. ADDRESS REGISTER

Address: 0x04

Type: shared



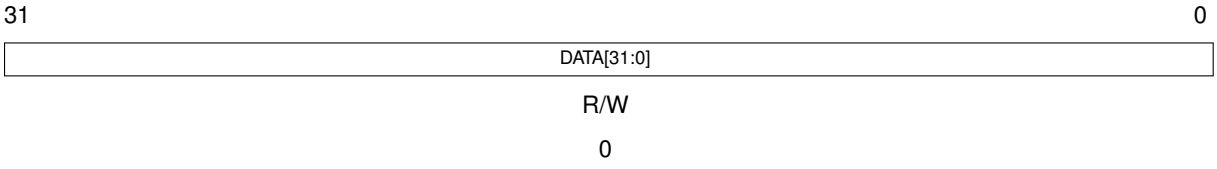
ADDR[31:0] *Address*

Memory address to read/write.

8.11.4. WRITE DATA REGISTER

Address: 0x08

Type: shared



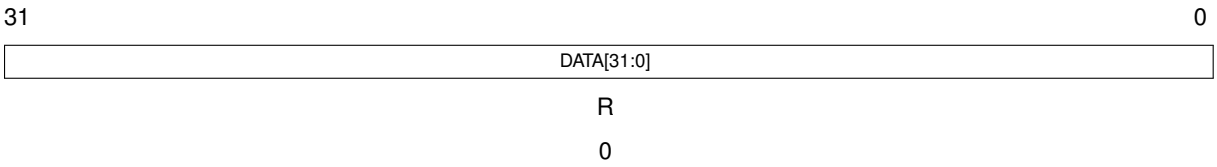
DATA[31:0] *Write Data*

Data to be written.

8.11.5. READ DATA REGISTER

Address: 0x0C

Type: shared



DATA[31:0] *Read Data*

Data read from the given address.

9. ON-CHIP DEBUGGER

CC-RV32HR-C processor implements a debug mode during which the processor pipeline is stalled. The On-Chip Debugger is used to control the processor cores during debug mode. The debug hardware have access to every memory and peripheral location defined in memory map. The On-Chip Debugger performs only 32-bit operations with big-endian byte order. Access to locations that are exclusive to every processor core, like scratch-pad ram, tightly-coupled peripherals or internal core memories, is possible by switching processor core context with debug context register. Before switching to another processor context, the host have to ensure that the interesting core is running. Otherwise, the On-Chip Debugger behavior is undefined. An external debug host can access On-Chip Debugger using UART or JTAG debug link.

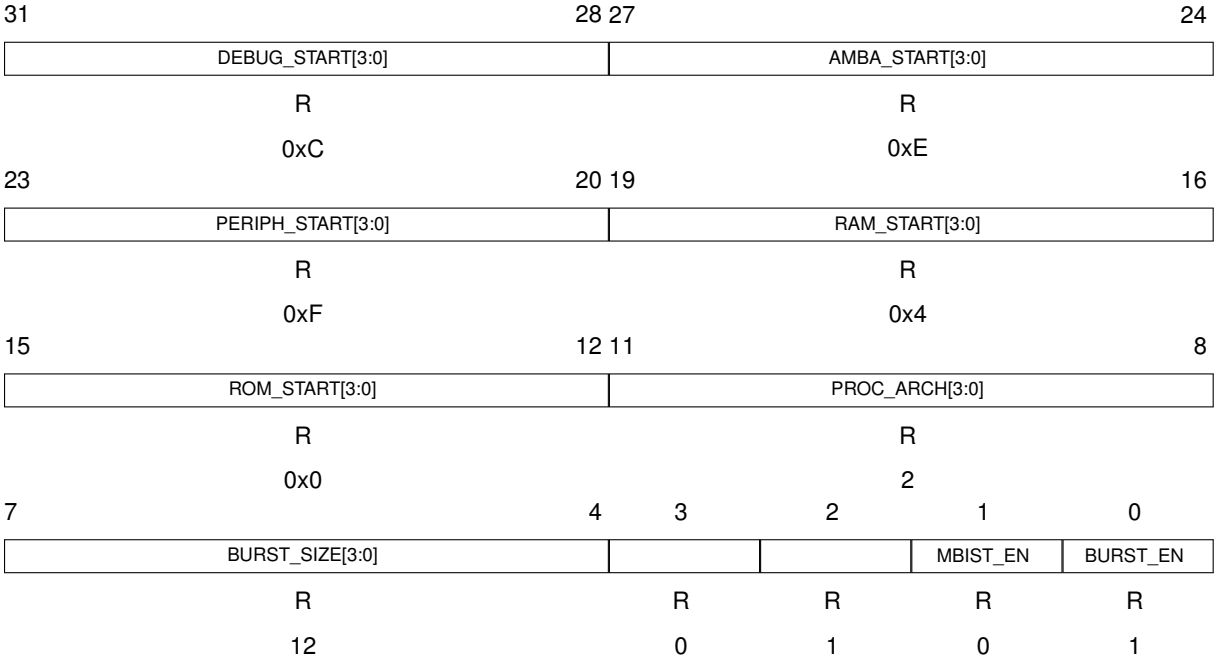
Table 9.1 shows the detailed description of Debug Region memory map. The visible content of locations starting from address 0xC1000000 depends on current debug context which points to the particular processor core.

Table 9.1: On-Chip Debugger memory map.

Address	Description
0xC0000000 - 0xC000000C	4 x Breakpoint
0xC0000010 - 0xC000001C	4 x Watchpoint
0xC0000020 - 0xC0000020	Burst Counter Register
0xC0000024 - 0xC0000024	Debug Version Register
0xC1000000 - 0xC100007C	Integer Register-File
0xC1000080 - 0xC10000FC	Floating-Point Register-File
0xC2000000 - 0xC20xxxxx	Instruction Cache Data Way 0
	Instruction Cache Data Way 1
	...
	Instruction Cache Data Way *n*
0xC2100000 - 0xC21xxxxx	Instruction Cache Tag Way 0
	Instruction Cache Tag Way 1
	...
	Instruction Cache Tag Way *n*
0xC2200000 - 0xC22xxxxx	Data Cache Data Way 0
	Data Cache Data Way 1
	...
	Data Cache Data Way *n*
0xC2300000 - 0xC23xxxxx	Data Cache Tag Way 0
	Data Cache Tag Way 1
	...
	Data Cache Tag Way *n*

9.1. VERSION REGISTER

Address: 0xC0000024



BURST_EN *Burst Enable*

Bit is set if debug burst transactions are allowed.

MBIST_EN *MBIST Enable*

Bit is set if memory BIST module is implemented.

BURST_SIZE[3:0] *Burst Counter Size*

Number of burst counter bits.

PROC_ARCH[3:0] *Processor Architecture*

Processor architecture code.

ROM_START[3:0] *ROM Region Start*

ROM region start address.

RAM_START[3:0] *RAM Region Start*

RAM region start address.

PERIPH_START[3:0] *Peripherals Region Start*

Tightly-coupled peripherals region start address.

AMBA_START[3:0] *AMBA Region Start*

AMBA peripherals region start address.

DEBUG_START[3:0] *Debug Region Start*

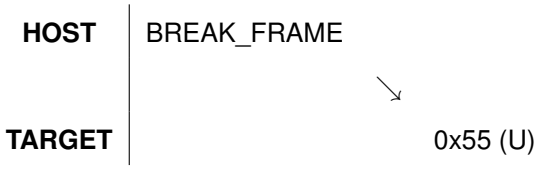
Debug region start address.

9.2. UART DEBUG LINK

UART Debug Link consists of a 2-wire UART communication with debug host using simple communication protocol with 8-bit, no parity, 1 stop bit frames. The On-Chip Debugger baud rate depends of current processor clock speed and content of the *On-Chip Debugger Baud Register* in Interrupt Controller. Before using the On-Chip Debugger, the user have to ensure that the processor is not in deep power down mode and its main clock is running. Otherwise the On-Chip Debugger will not respond.

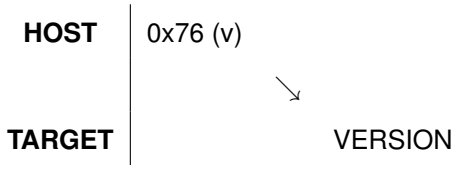
9.2.1. BAUD RATE DETECTION

If not known, the On-Chip Debugger baud rate detection can be achieved by sending break frame by a debug host. As the result, the target will send 0x55 (U) char, which can be used to calculate On-Chip Debugger baud rate.



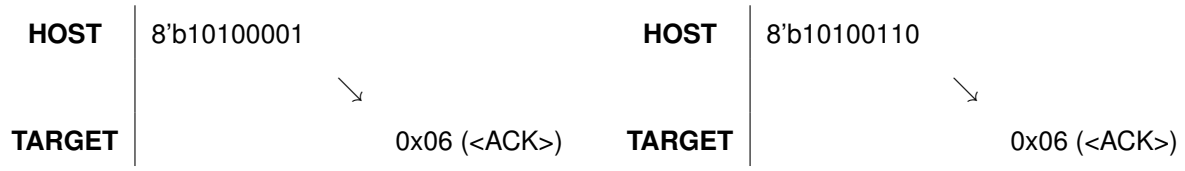
9.2.2. DEBUG VERSION

The Debug Version Register can be directly readout be sending 0x76 (v) command. Target will respond with Debug Version Register data. Before switching to another processor context, the host have to ensure that the interesting core is running. Otherwise, the On-Chip Debugger behavior is undefined.



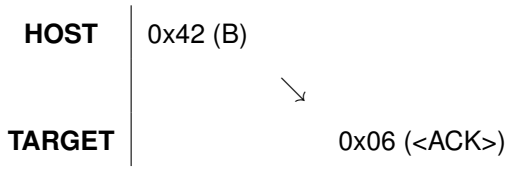
9.2.3. DEBUG CONTEXT

After reset the debug context register points to the processor Core 0. This means that access to scratch-pad ram, tightly-coupled memories or internal core memories like caches or register files will be forwarded to the Core 0 exclusive resources. Debug context switch can be done at any moment by sending 8'b101xxxxx char, where the last 5 bits represent the core index. Target will respond with 0x06 (<ACK>) char. Before switching to another processor context, the host have to ensure that the interesting core is running. Otherwise, the On-Chip Debugger behavior is undefined.

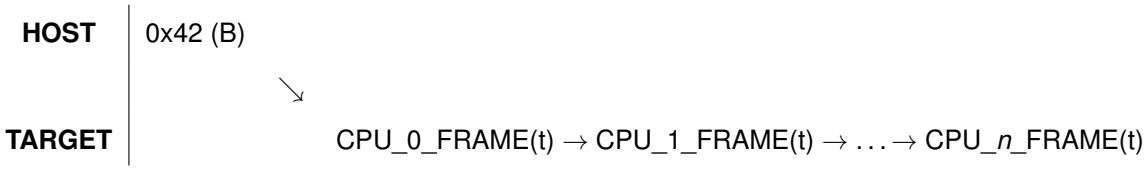


9.2.4. DETECTING DEBUG MODE

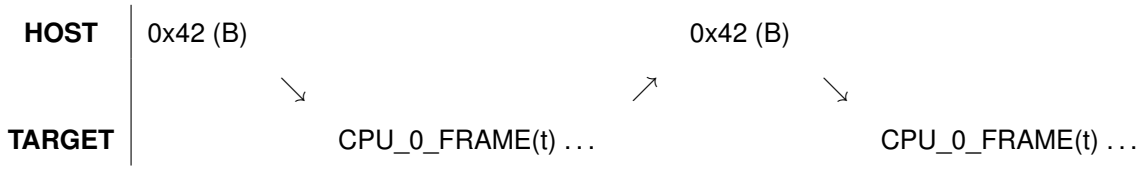
The detection of current processor mode can be done by sending 0x42 (B) char. If processor is not in the debug mode, the target will respond with 0x06 (<ACK>) char.



Otherwise the target will send one state frame for every processor core in the system.



The 0x42 (B) char can also be used to resend the current processor state frame after losing sync with the target.



9.2.5. PROCESSOR CORE STATE FRAME

There are two types of processor core state frames: long and short. The long frame consists fo 26 bytes and is sent for a running processor core and a 2 byte short frame is sent for the stopped one. The structure of a long state frame is presented below.

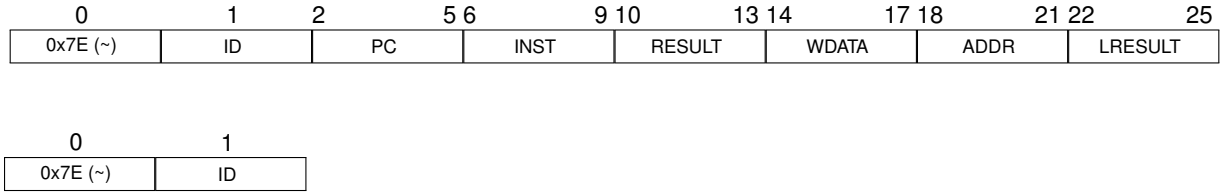


Figure 9.1: Long and short processor core state frame structure.

Table 9.2: Processor core state frame.

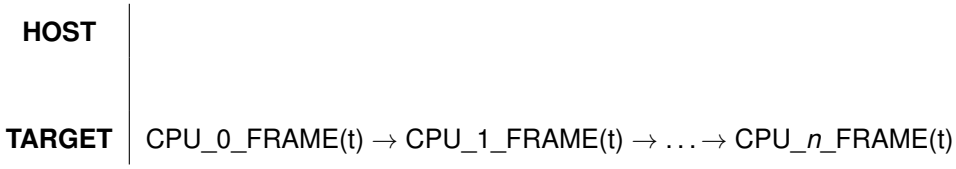
Field	Description
0x7E (~)	Start of frame
ID	ID[7] = 0 - processor core is running ID[7] = 1 - processor core is stopped ID[6:0] - processor core index
PC	Current processor core program counter
INST	Current instruction opcode
RESULT	Result of arithmetic, logical, move and store conditional instructions
WDATA	Write data of store instruction
ADDR	Address of load and store instructions
LRESULT	Load instruction result

9.2.6. ENTERING DEBUG MODE

There are several methods of entering the debug mode. The user program can force processor to enter the debug mode by executing EBREAK instruction. The debug mode can be caused by executing certain program address or by performing access to the certain memory location. Finally, the debug mode can be caused by executing On-Chip Debugger command or asserting DBG_BREAK pin.
The list of events causing the processor to enter the debug mode is listed below:

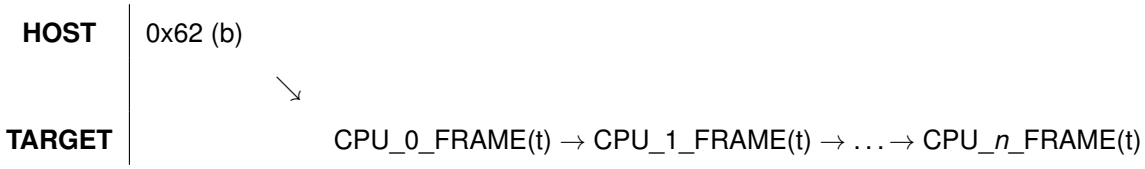
- executing a EBREAK instruction
- hardware breakpoint/watchpoint hit
- executing debugger Break Command
- asert DBG_BREAK pin

When the processor core hit any of breakpoints or watchpoints, or after execution a BREAK instruction with 0x0F code, the processor will enter the debug mode and dump its state frames.



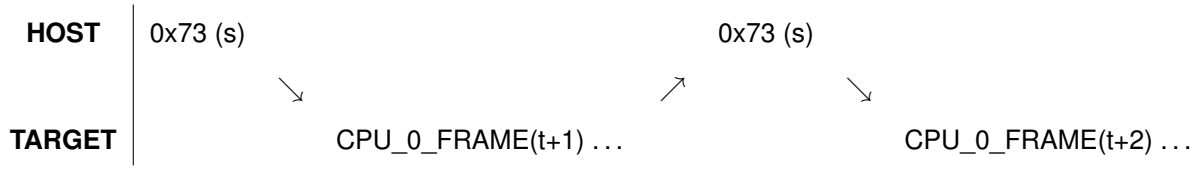
9.2.7. BREAK COMMAND

One of the possible events that will cause processor to enter the debug mode is executing the break command by the On-Chip Debugger. The command will have no effect if processor is already in the debug mode.



9.2.8. STEP COMMAND

Step command will force processor cores to execute next single instruction and dump its new state frames. Command will have no effect if processor is not in the debug mode.



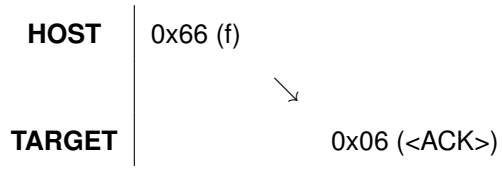
9.2.9. LEAVING DEBUG MODE

There are three On-Chip Debugger command that will cause processor to leave the debug mode:

- Free Running Command
- Processor Reset Command
- Debugger Reset Command

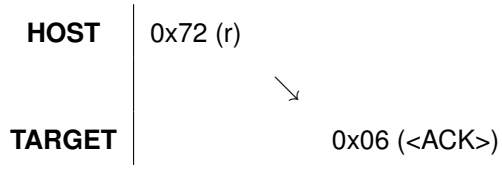
9.2.10. FREE RUNNING COMMAND

After executing this command, the processor will leave the debug mode. Processor will enter debug mode again after occurring one of described earlier events. The command will have no effect if processor is not in the debug mode.



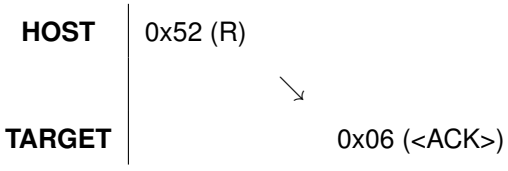
9.2.11. PROCESSOR RESET COMMAND

After executing this command, the On-Chip Debugger will reset the processor. If processor was in the debug mode it will leave this state and than reset. Processor will enter debug mode again after occurring one of the described earlier events.



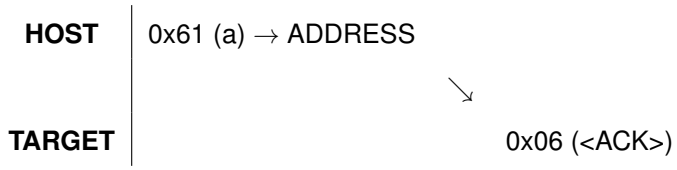
9.2.12. DEBUGGER RESET COMMAND

After executing this command, the On-Chip Debugger will reset its internal registers. Breakpoints and watchpoints will be cleared (set to 0xFFFFFFFF value) and internal Address Register will be zeroed. If processor was in the debug mode it will leave this state. Processor will enter debug mode again after occurring one of the described earlier events.



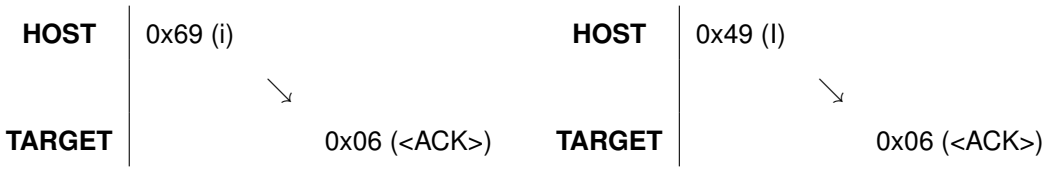
9.2.13. ADDRESS COMMAND

This command is used to load 32-bit value into the internal On-Chip Debugger Address Register. This register value will be used to perform read and write operations on the memory map locations. This command is accessible weather the processor is in the debug mode or not.



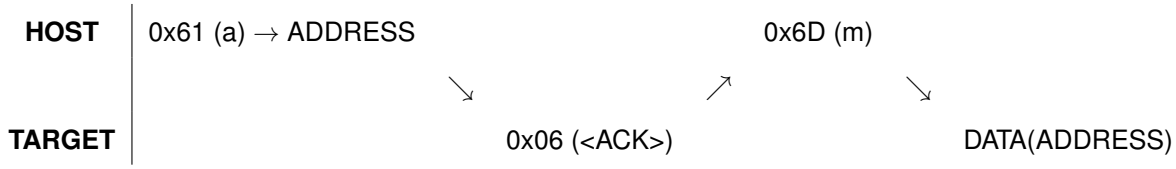
9.2.14. ADDRESS AUTO-INCREMENT ON/OFF COMMAND

After reset the address autoincrementation is enabled. This means that the internal Address Register is incremented by the value of 4 after each read or write operation. Autoincrementation can be disabled be sending 0x69 (i) char and turned on again after sending 0x49 (I) char. This command is accessible weather the processor is in the debug mode or not.

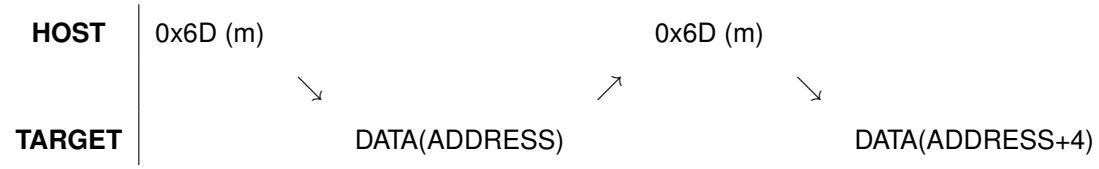


9.2.15. MEMORY READ COMMAND

This command is used to read the content of memory location pointed by the internal Address Register. The command is accessible in the debug mode or if the Address Register points to the On-Chip Debugger internal registers (Breakpoint, Watchpoint, Burst Counter Register).

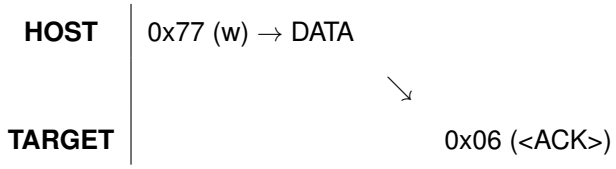


When the autoincrementation is enabled every Read Memory Command execution will dump consecutive memory location content.

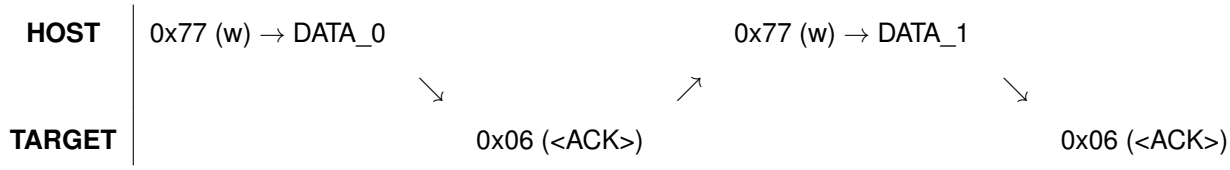


9.2.16. MEMORY WRITE COMMAND

This command is used to write data to the memory location pointed by the internal Address Register. The command is accessible in the debug mode or if the Address Register points to the On-Chip Debugger internal registers (Breakpoint, Watchpoint, Burst Counter Register).

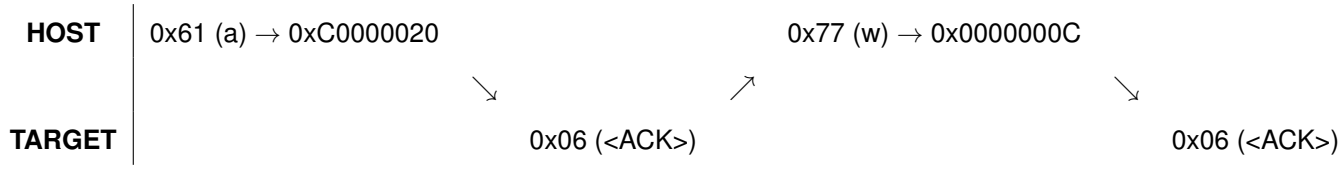


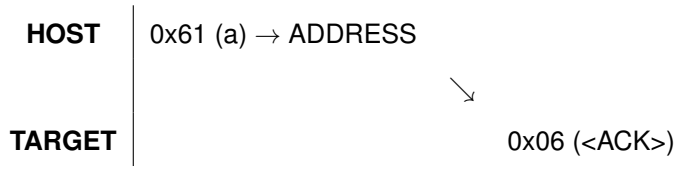
When the autoincrementation is enabled every Memory Write Command execution will store data to the consecutive memory location.

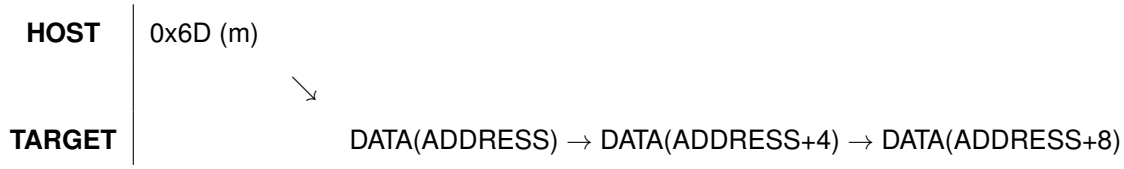


9.2.17. BURST COUNTER REGISTER

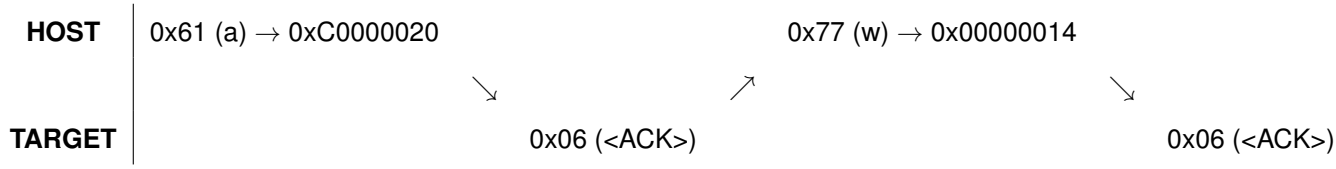
The On-Chip Debugger Burst Counter Register is located at address 0x91000000. The maximum allowed value of 0x1000 can result in instant reading or writing of 1024 words containing 32 bits each. When autoincrementation is enabled and Burst Counter Register is set to the value greater than 4, the memory read operation will stream programmed number of 32-bit words from consecutive locations pointed by the Address Register.

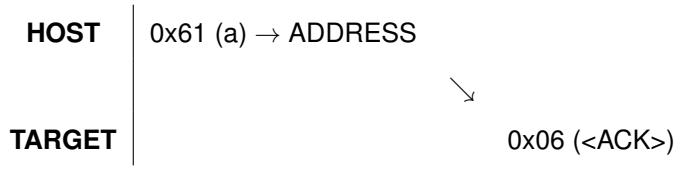


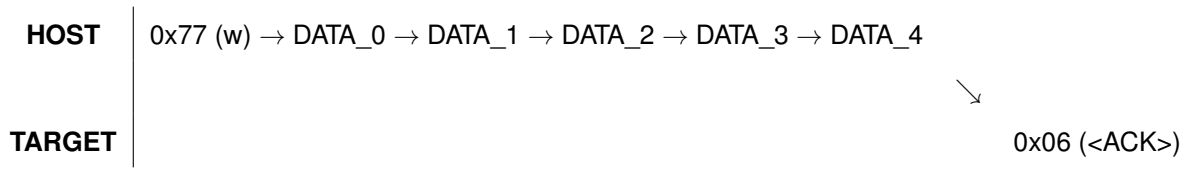




Similarly, when autoincrementation is enabled and Burst Counter Register is set to the value greater than 4, the memory write operation can be instantly followed by a number of 32-bit data that will be stored in consecutive memory locations.

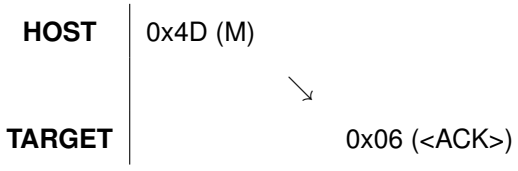






9.2.18. MBIST RUN COMMAND

After executing this command, if MBIST is present, the On-Chip Debugger will run the MBIST controller test procedure. The command returns <ACK> when all tested memories are functional and <NACK> otherwise.



9.2.19. UART DEBUG LINK PINOUT

Table 9.3: UART Debug Link pinout.

Pin Mame	Description	Type	Comment
DBG_EN	On-Chip Debugger Enable Input	Input	Active high
DBG_BREAK	On-Chip Debugger Break Input	Input	Force entering debug mode, active high
DBG_TX	Debug Transmit Data	Output	
DBG_RX	Debug Receive Data"	Input	

9.3. JTAG DEBUG LINK

JTAG interface provides a communication link with the debug host using simple communitation protocol which translates JTAG instructions to On-Chip Debugger commands. The JTAG interface supports the 4-bit instructions shown in Table 9.4.

Table 9.4: JTAG On-Chip Debugger commands.

Command	Opcode	Description
EXTEST	4'b0000	Instruction performs a PCB interconnect test, places an IEEE 1149.1 compliant device into an external boundary test mode, and selects the boundary scan register to be connected between TDI and TDO.
SAMPLE_PRELOAD	4'b0001	Instruction allows an IEEE 1149.1 compliant device to remain in its functional mode and selects the boundary scan register to be connected between the TDI and TDO pins.
IDCODE	4'b0010	Instruction is associated with a 32-bit identification register. Its data uses a standardized format that includes a manufacturer code.
HIGHZ	4'b0011	Instruction forces all output pins drivers into high impedance (High-Z) states.
DEBUG_COMMAND	4'b1000	Instruction is used to enter On-Chip Debugger command. Debug Command Register is selected to be connected between TDI and TDO.
DEBUG_DATA	4'b1001	Instruction is used to readout processor memory map content. Debug Data Register is selected to be connected between TDI and TDO.
DEBUG_STATUS	4'b1010	Instruction is used to readout processor core state frames. Debug Status register is selected to be connected between TDI and TDO.
BYPASS	4'b1111	Using this instruction, a device's boundary scan chain can be skipped, allowing the data to pass through the bypass register.

9.3.1. JTAG DEBUG COMMAND INSTRUCTION

The Debug Command Instruction selects Debug Command Register to be connected between TDI and TDO pins. The Debug Command Register is 48-bit length on write and 8-bit read. Its purpose is to enter the On-Chip Debugger command. After entering the command, readout of this register can result in the value of 0x06 (<ACK>) if entering command was successful (CRC field matched), or 0x15 (<NACK>) otherwise. JTAG TAP Update-DR State is used to trigger CRC comparison, loading Debug Command Register with ACK or NACK char and execute the command on CRC match. The Debug Command Register is using CRC-8 calculated over COMMAND and DATA field to ensure the correct operation. CRC-8 uses $x^8 + x^7 + x^4 + x^3 + x^1 + x^0$ polynomial. The Debug Command Register structure is shown in figure 9.2.

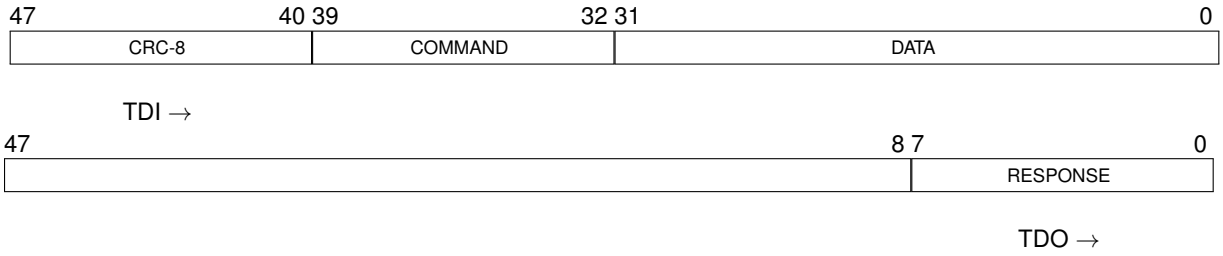


Figure 9.2: Debug Command Register structure.

Note, that during the same time the readout of 8-bit response is done, a potentially new command is shifted in through the TDI pin. Therefore it is strongly recommended to readout the entire 48-bit register or shift in new command.

9.3.2. JTAG DEBUG DATA INSTRUCTION

The Debug Data Instruction selects Debug Data Register to be connected between TDI and TDO pins. The Debug Data Register is 40-bit length. The 32-bit data field is followed by the CRC-8 field to ensure the correct operation. CRC-8 uses $x^8 + x^7 + x^4 + x^3 + x^1 + x^0$ polynomial. The Debug Data Register is read-only and is loaded with stored Read Data Register on every JTAG TAP Capture-DR State. Next, when in the debug mode or if the Address Register points to the On-Chip Debugger internal registers, the On-Chip Debugger reads the memory content of location pointed by the current Address Register value and stores it in the Read Data Register. Finally, when the autoincrementation is enabled, the Address Register is incremented by the value of 4. The Read Data Register is cleared on the reset. The Debug Data Register structure is shown in figure 9.3.

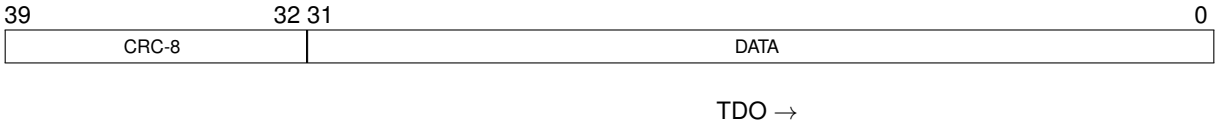


Figure 9.3: Debug Data Register structure.

9.3.3. JTAG DEBUG STATUS INSTRUCTION

The Debug Status Instruction selects Debug Status Register to be connected between TDI and TDO pins. The Debug Status Register is composed of a 32-bit CRC-32 and 200-bit processor core status frame for every processor core in the system. CRC-32 is calculated over all remaining bits ans uses $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$ polynomial. The Debug Status Register is read-only and is loaded with new data on every JTAG TAP Capture-DR State. The Debug Status Register structure is presented below:

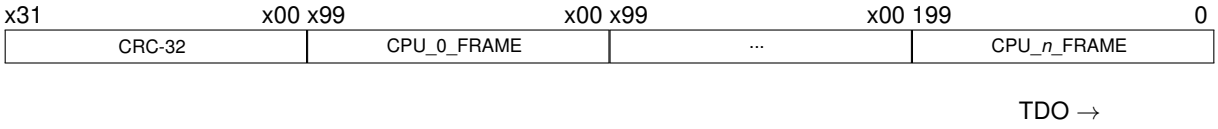


Figure 9.4: Debug Stauts Register structure.

The structure of a processor core state frame is presented below. As can be seen, readout of the Debug Status Register can be used to determine if the processor is in the debug state or not.

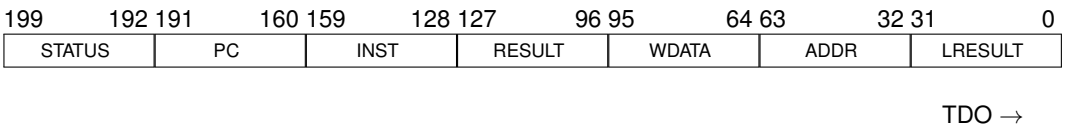


Figure 9.5: JTAG processor core state frame structure.

Table 9.5: Processor core state frame.

Field	Description
STATUS	STATUS[7:0] = 0xFF - processor is not in the debug mode Otherwise: STATUS[7] = 0 - processor core is running STATUS[7] = 1 - processor core is stopped STATUS[6:0] - processor core index
PC	Current processor core program counter
INST	Current instruction opcode
RESULT	Result of arithmetic, logical, move and store conditional instructions
WDATA	Write data of store instruction
ADDR	Address of load and store instructions
LRESULT	Load instruction result

9.3.4. DEBUG VERSION

The Debug Version Register can be directly readout using Debug Version command. Read data is placed in the Read Data Register and can be readout using Debug Data Instruction.

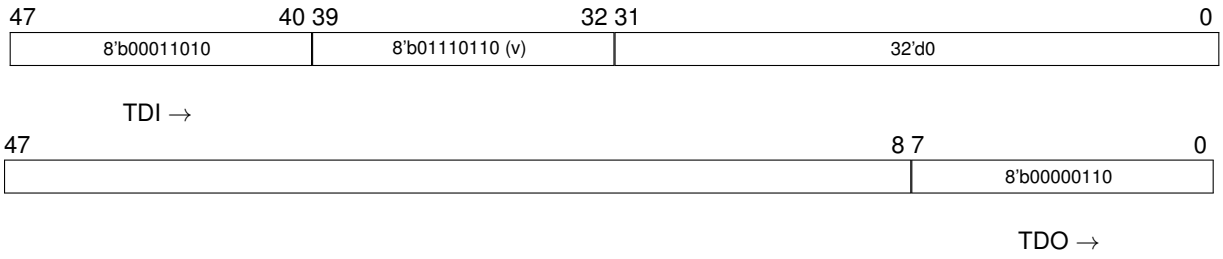


Figure 9.6: Debug Version Command structure.

9.3.5. ENTERING DEBUG MODE

There are several methods of entering the debug mode. The user program can force processor to enter the debug mode by executing BREAK instruction with 0x0F code. The debug mode can be caused by executing certain program address or by performing access to the certain memory location. Finaly, the debug mode can be caused by executing On-Chip Debugger command.
The list of events causing the processor to enter the debug mode is listed below:

- executing a BREAK instruction with 0x0F code
- hardware breakpoint/watchpoint hit
- executing debugger Break Command

9.3.6. DEBUG CONTEXT

After reset the debug context register points to the processor Core 0. This means that access to scratch-pad ram, tightly-coupled memories or internal core memories like caches or register files will be forwarded to the Core 0 exclusive resources. Debug context switch can be done at any moment by using 8'b101xxxxx command, where the last 5 bits represent the core index. Before switching to another processor context, the host have to ensure that the interesting core is running. Otherwise, the On-Chip Debugger behavior is undefined.
The command is accessible in both debug mode or if the Address Register points to the On-Chip Debugger internal registers (Breakpoint, Watchpoint, Burst Counter Register).

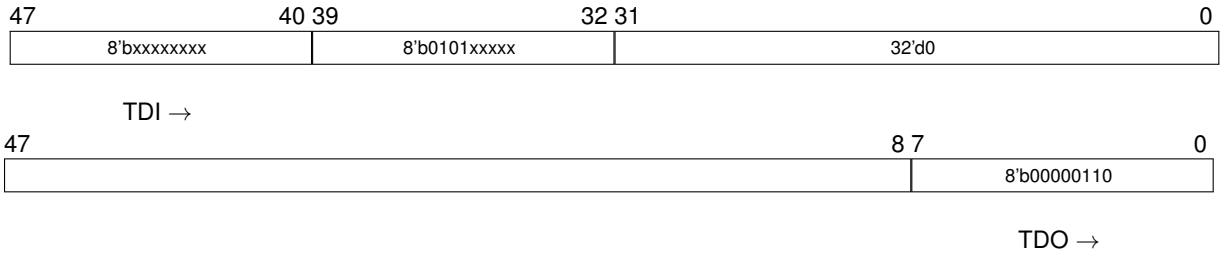


Figure 9.7: Debug Context Command structure.

9.3.7. BREAK COMMAND

One of the possible events that will cause processor to enter the debug mode is executing the break command by the On-Chip Debugger. The command will have no effect if processor is already in the debug mode. Host can check the current processor state by reading Debug Status Register using Debug Status Instruction.

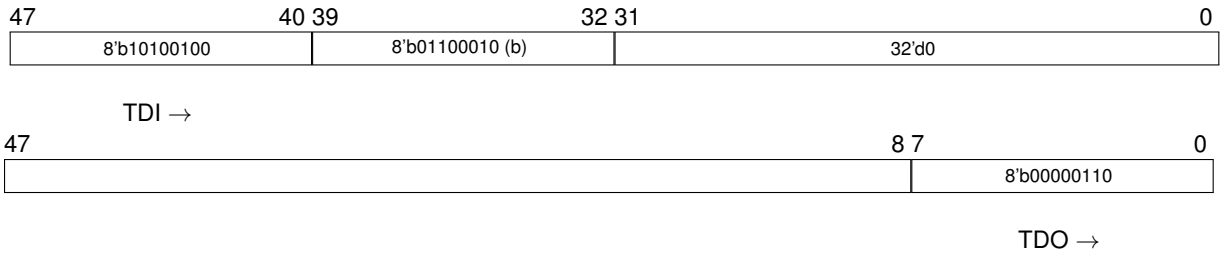


Figure 9.8: Break Command structure.

9.3.8. STEP COMMAND

Step command will force processor cores to execute next single instruction. Command will have no effect if processor is not in the debug mode. Host can check the current processor state by reading Debug Status Register using Debug Status Instruction.

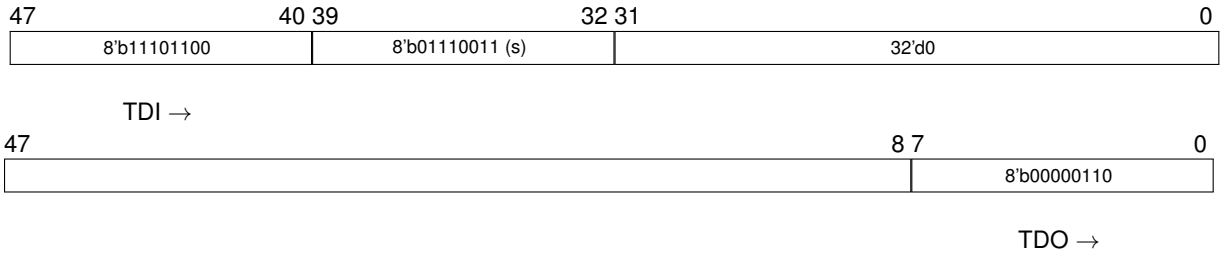


Figure 9.9: Step Command structure.

9.3.9. LEAVING DEBUG MODE

There are three On-Chip Debugger command that will cause processor to leave the debug mode:

- Free Running Command
- Processor Reset Command
- Debugger Reset Command

9.3.10. FREE RUNNING COMMAND

After executing this command, the processor will leave the debug mode. Processor will enter debug mode again after occurring one of described earlier events. The command will have no effect if processor is not in the debug mode.

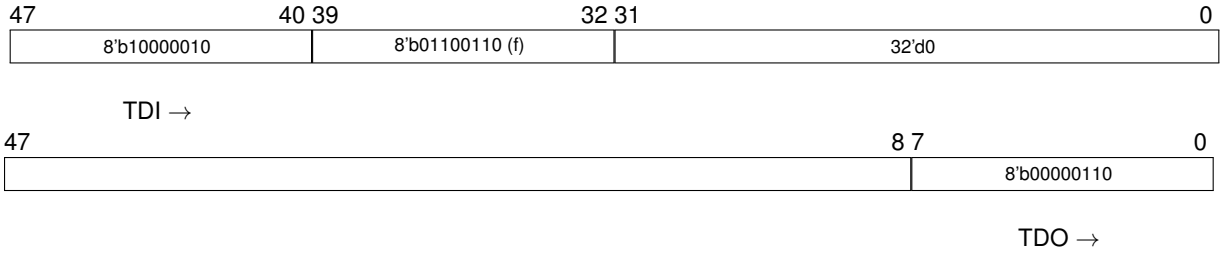


Figure 9.10: Free Running Command structure.

9.3.11. PROCESSOR RESET COMMAND

After executing this command, the On-Chip Debugger will reset the processor. If processor was in the debug mode it will leave this state and than reset. Processor will enter debug mode again after occurring one of the described earlier events.

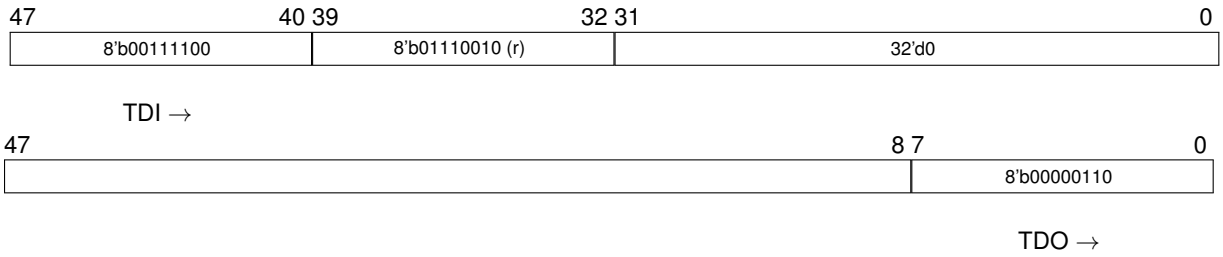


Figure 9.11: Reset Command structure.

9.3.12. DEBUGGER RESET COMMAND

After executing this command, the On-Chip Debugger will reset its internal registers. Breakpoints and watchpoints will be cleared (set to 0xFFFFFFFF value) and internal Address Register will be zeroed. If processor was in the debug mode it will leave this state. Processor will enter debug mode again after occurring one of the described earlier events.

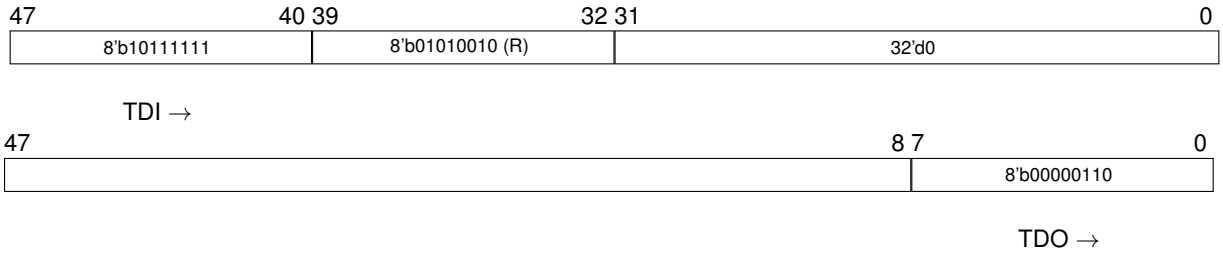


Figure 9.12: Debugger Reset Command structure.

9.3.13. ADDRESS COMMAND

This command is used to load 32-bit value into the internal On-Chip Debugger Address Register. This register value will be used to perform read and write operations on the memory map locations. This command is accessible weather the processor is in the debug mode or not.

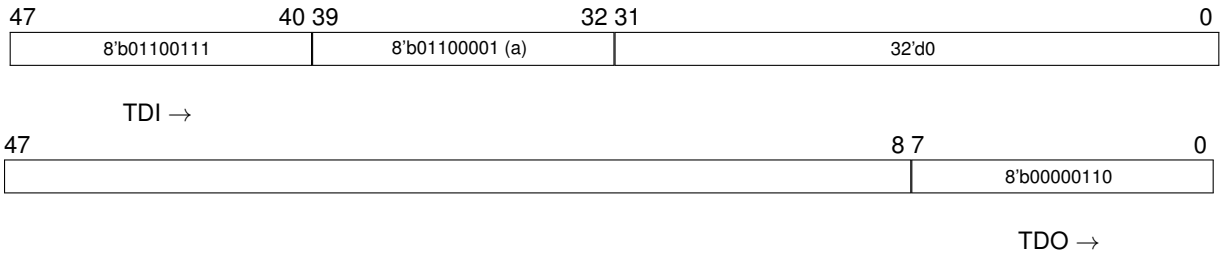


Figure 9.13: Address Command structure.

9.3.14. ADDRESS AUTO-INCREMENT ON/OFF COMMAND

After reset the address autoincrementation is enabled. Autoincrementation can be enabled or disabled by executing the following commands. This command is accessible weather the processor is in the debug mode or not.

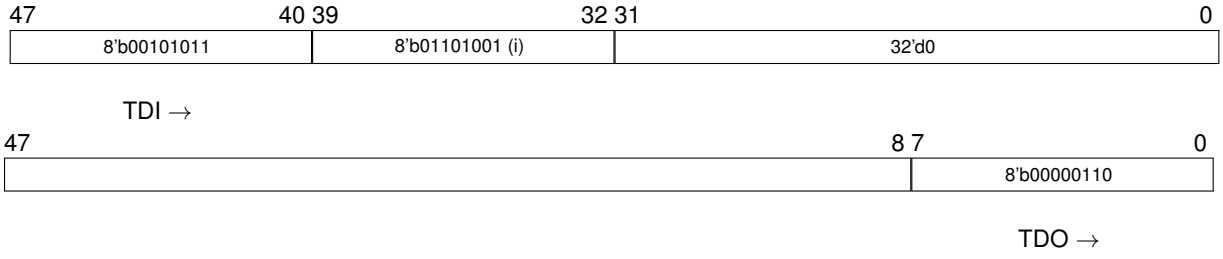


Figure 9.14: Auto-Increment Off Command structure.

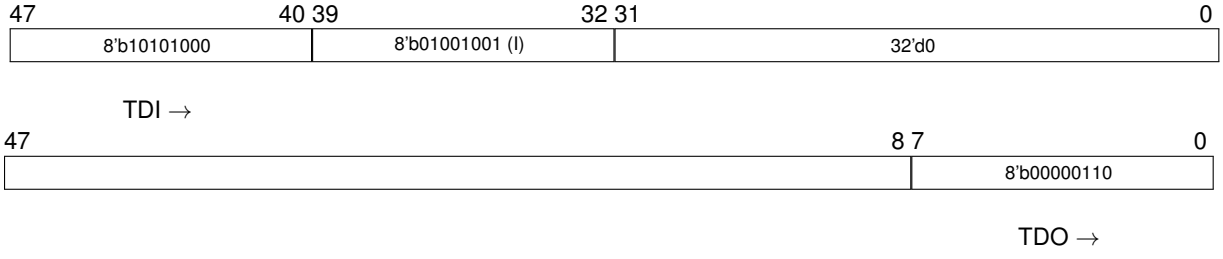


Figure 9.15: Auto-Increment On Command structure.

9.3.15. MEMORY READ COMMAND

This command is used to read the content of memory location pointed by the internal Address Register. Read data is placed in the Read Data Register and can be readout using Debug Data Instruction. After command execution, Address Register will be incremented if autoincrement option is enabled. The Memory Read Command is used to perform initial memory content read. After that, repeatedly reading of Debug Data Register can be used to read consecutive memory locations. The command is accessible in the debug mode or if the Address Register points to the On-Chip Debugger internal registers (Breakpoint, Watchpoint, Burst Counter Register).

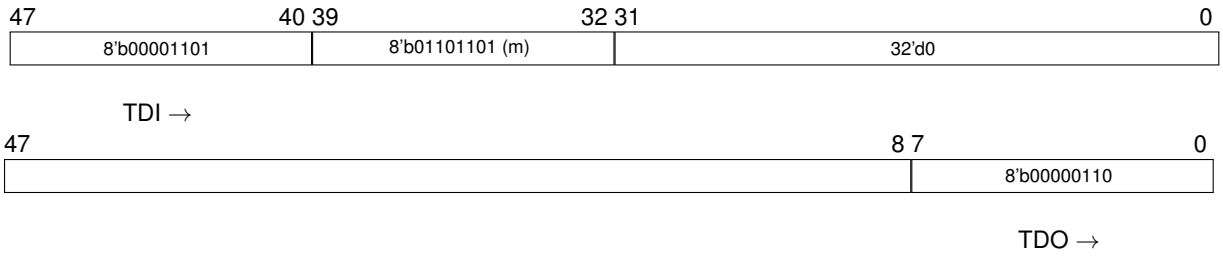


Figure 9.16: Memory Read Command structure.

9.3.16. MEMORY WRITE COMMAND

This command is used to write data to the memory location pointed by the internal Address Register. The command is accessible in the debug mode or if the Address Register points to the On-Chip Debugger internal registers (Breakpoint, Watchpoint, Burst Counter Register). After command execution, Address Register will be incremented if autoincrement option is enabled.

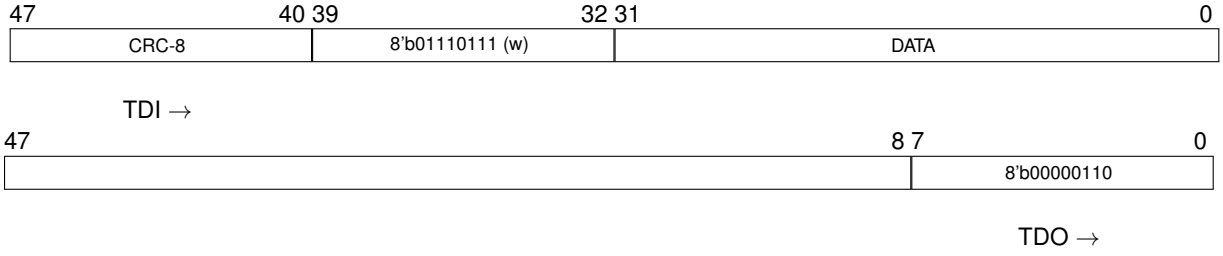


Figure 9.17: Memory Write Command structure.

9.3.17. MBIST RUN COMMAND

After executing this command, if MBIST is present, the On-Chip Debugger will run the MBIST controller test procedure. The results can be readout from the MBIST controller peripheral.

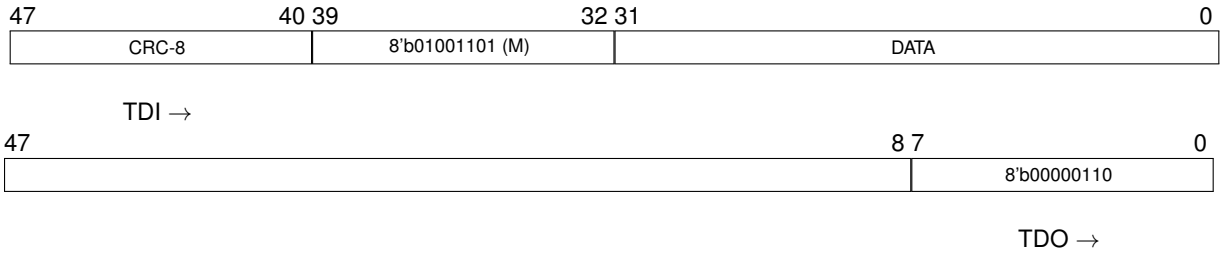


Figure 9.18: MBIST Run Command structure.

9.3.18. JTAG DEBUG LINK PINOUT

Table 9.6: JTAG Debug Link pinout.

Pin Mame	Description	Type	Comment
DBG_EN	On-Chip Debugger Enable Input	Input	Active high
TDO	JTAG Test Data Output	Output	
RTCK	JTAG Return TCK Output	Output	JTAG return TCK
TCK	JTAG Test Clock Input	Input	
TMS	JTAG Test Mode Select Input	Input	
TDI	JTAG Test Data Input	Input	

10. REVISION HISTORY

Table 10.1: Revision history.

Doc. Rev.	Date	Comments
1.0	01-2023	First Issue.

ChipCraft®

TABLE OF CONTENTS

1. Features 2

2. Instruction Set 3

3. Performance and Utilization 4

4. Block Diagram 5

5. Memory Map 6
   - 5.1. Region Descriptions 7
     - 5.1.1. ROM Region 7
     - 5.1.2. Scratch-pad RAM Region 7
     - 5.1.3. RAM Region 7
     - 5.1.4. Debug Region 8

6. Control and Status Registers 9
   - 6.1. Machine-Level CSRs 10
     - 6.1.1. Machine Vendor ID Register 12
     - 6.1.2. Machine Architecture ID Register 12
     - 6.1.3. Machine Implementation ID Register 13
     - 6.1.4. Machine Cause Register 14
   - 6.2. User-Level CSRs 16
   - 6.3. Custom Machine-Level CSRs 17
     - 6.3.1. Processor Configuration Register 0 18
     - 6.3.2. Processor Configuration Register 1 19
     - 6.3.3. Processor Configuration Register 2 21
     - 6.3.4. Processor Control Register 23
     - 6.3.5. Minimum Stack Pointer Value Register 24
     - 6.3.6. Maximum Stack Pointer Value Register 24
     - 6.3.7. On-Chip Debugger Baud Register 25
     - 6.3.8. ROM Unlock Register 26

7. Scratch-Pad RAM Controller 27

8. Tightly-Coupled Peripherals 28
   - 8.1. Register Description Convention 28
   - 8.2. Memory Map 29
   - 8.3. Multicore Controller 30
     - 8.3.1. Lockstep Controller 30
     - 8.3.2. Registers List 31
     - 8.3.3. Status Register 32
     - 8.3.4. Core Number Register 32
     - 8.3.5. Shutdown Register 33
     - 8.3.6. Start Address Registers 34
     - 8.3.7. Core Run Registers 35

8.3.8. Core 0 Injection Mask Register 36
8.3.9. Core 1 Injection Mask Register 36
8.3.10. Core Deadlock Counter Max Register 37
8.3.11. Error Program Counter Register 37
8.3.12. Error Statistics Register 38
8.3.13. Lockstep Control Register 38
8.4. Power Management Controller 39
8.4.1. Registers List 39
8.4.2. Reset Cause Register 40
8.4.3. Reset Register 41
8.4.4. Deep Reset Register 42
8.4.5. Info Register 42
8.5. CSR Controller 44
8.5.1. Stack Protection Unit 44
8.5.2. Non-maskable Interrupt 44
8.6. Platform-Level Interrupt Controller 45
8.6.1. Registers List 46
8.6.2. Interrupt Priority Registers 46
8.6.3. Interrupt Pending Register 47
8.6.4. Interrupt Enable Register 47
8.6.5. Interrupt Threshold Registers 48
8.6.6. Interrupt Claim Registers 48
8.7. Inter-Core Interrupt Controller 49
8.7.1. Registers List 49
8.7.2. Inter-Core Interrupt Mapping Register 49
8.7.3. Inter-Core Interrupt Trigger Register 50
8.7.4. Inter-Core Interrupt Flags Register 50
8.8. Core Local Interruptor Controller 51
8.8.1. Registers List 51
8.8.2. Machine Software Interrupt Pending 51
8.8.3. Machine Time Compare 52
8.8.4. Machine Time Register 52
8.8.5. Machine Time Config Register 53
8.9. Instruction Cache Controller 54
8.9.1. Parity Support 55
8.9.2. Memory Scrambling 55
8.9.3. Registers List 55
8.9.4. Status and Control Register 56
8.9.5. Flush Register 57
8.9.6. Info Register 58
8.9.7. Parity Error Count 0 59
8.9.8. Parity Error Count 1 60
8.9.9. Error Injection Mask Low 60
8.9.10. Error Injection Mask High 61
8.10. Data Cache Controller 62

8.10.1. EDAC Support 63
8.10.2. Memory Scrambling 63
8.10.3. Registers List 64
8.10.4. Status and Control Register 64
8.10.5. Flush Register 66
8.10.6. Info Register 66
8.10.7. ECC Error Count 0 68
8.10.8. ECC Error Count 1 68
8.10.9. Error Injection Mask Low 69
8.10.10. Error Injection Mask High 69
8.10.11. Last Error Address 70
8.10.12. Last Error Statistics 70
8.11. Debug Interface Controller 72
8.11.1. Registers List 72
8.11.2. Command Register 72
8.11.3. Address Register 73
8.11.4. Write Data Register 73
8.11.5. Read Data Register 74

9. On-Chip Debugger 75
9.1. Version Register 77
9.2. UART Debug Link 79
9.2.1. Baud Rate Detection 79
9.2.2. Debug Version 79
9.2.3. Debug Context 79
9.2.4. Detecting Debug Mode 80
9.2.5. Processor Core State Frame 80
9.2.6. Entering Debug Mode 81
9.2.7. Break Command 82
9.2.8. Step Command 82
9.2.9. Leaving Debug Mode 82
9.2.10. Free Running Command 83
9.2.11. Processor Reset Command 83
9.2.12. Debugger Reset Command 83
9.2.13. Address Command 84
9.2.14. Address Auto-Increment On/Off Command 84
9.2.15. Memory Read Command 84
9.2.16. Memory Write Command 85
9.2.17. Burst Counter Register 85
9.2.18. MBIST Run Command 86
9.2.19. UART Debug Link Pinout 87
9.3. JTAG Debug Link 88
9.3.1. JTAG Debug Command Instruction 89
9.3.2. JTAG Debug Data Instruction 89
9.3.3. JTAG Debug Status Instruction 90

9.3.4. Debug Version 91
9.3.5. Entering Debug Mode 92
9.3.6. Debug Context 92
9.3.7. Break Command 92
9.3.8. Step Command 93
9.3.9. Leaving Debug Mode 93
9.3.10. Free Running Command 93
9.3.11. Processor Reset Command 94
9.3.12. Debugger Reset Command 94
9.3.13. Address Command 95
9.3.14. Address Auto-Increment On/Off Command 95
9.3.15. Memory Read Command 96
9.3.16. Memory Write Command 96
9.3.17. MBIST Run Command 97
9.3.18. JTAG Debug Link Pinout 97

10. Revision History 98

List of Tables 103

List of Figures 104

ChipCraft®

LIST OF TABLES

Table 3.1 Integer unit performance. 4
Table 3.2 Xilinx Ultrascale resource utilization. 4

Table 5.1 Debug Mode regions. 8

Table 6.1 Machine-level CSR registers. 10
Table 6.2 Exception codes. 14
Table 6.3 Machine-level CSR registers. 16
Table 6.4 Machine-level custom CSR registers. 17

Table 8.1 Tightly-Coupled Peripherals memory map. 29
Table 8.2 Multicore Controller registers list. 31
Table 8.3 Power Management Controller registers list. 39
Table 8.4 Platform-Level Interrupt Controller registers list. 46
Table 8.5 Inter-Core Interrupt Controller registers list. 49
Table 8.6 Core Local Interruptor Controller registers list. 51
Table 8.7 Instruction Cache Controller registers list. 55
Table 8.8 Data Cache Controller registers list. 64
Table 8.9 Debug Interface Controller registers list. 72

Table 9.1 On-Chip Debugger memory map. 76
Table 9.2 Processor core state frame. 81
Table 9.3 UART Debug Link pinout. 87
Table 9.4 JTAG On-Chip Debugger commands. 88
Table 9.5 Processor core state frame. 91
Table 9.6 JTAG Debug Link pinout. 97

Table 10.1 Revision history. 98

ChipCraft®

LIST OF FIGURES

Figure 4.1 Processor block diagram. 5

Figure 8.1 Simplified processor cores clock gating scheme. 30
Figure 8.2 Simplified stack exception generation datapath. 44
Figure 8.3 Platform-Level Interrupt Controller (PLIC) conceptual block diagram. 45
Figure 8.4 1 KiB/way, 32 bytes/line address mapping example. 54
Figure 8.5 4 KiB/way, 16 bytes/line address mapping example. 54
Figure 8.6 Tag memory entry with LRR bit. 54
Figure 8.7 Tag memory entry without LRR bit. 54
Figure 8.8 1 KiB/way, 32 bytes/line address mapping example. 62
Figure 8.9 4 KiB/way, 16 bytes/line address mapping example. 62
Figure 8.10 Tag memory entry with LRR bit. 62
Figure 8.11 Tag memory entry without LRR bit. 62

Figure 9.1 Long and short processor core state frame structure. 81
Figure 9.2 Debug Command Register structure. 89
Figure 9.3 Debug Data Register structure. 90
Figure 9.4 Debug Stauts Register structure. 90
Figure 9.5 JTAG processor core state frame structure. 90
Figure 9.6 Debug Version Command structure. 91
Figure 9.7 Debug Context Command structure. 92
Figure 9.8 Break Command structure. 93
Figure 9.9 Step Command structure. 93
Figure 9.10 Free Running Command structure. 94
Figure 9.11 Reset Command structure. 94
Figure 9.12 Debugger Reset Command structure. 94
Figure 9.13 Address Command structure. 95
Figure 9.14 Auto-Increment Off Command structure. 95
Figure 9.15 Auto-Increment On Command structure. 95
Figure 9.16 Memory Read Command structure. 96
Figure 9.17 Memory Write Command structure. 96
Figure 9.18 MBIST Run Command structure. 97

ChipCraft Sp. z o.o.
Bukowinska 22 lok. 9B
02-703 Warsaw, POLAND

www.chipcraft-ic.com