CC-RV32ST-C RISC-V PROCESSOR CORE

USER MANUAL

VERSION 1.1

1. FEATURES

- **RV32ST Processor Core**

    - RV32IMAFD (RV32G) instruction set

    - high-performance, in-order, single-issue pipeline

    - PHT, GHR, BTB dynamic branch prediction

- **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-RV32ST-C instruction set architecture.

The CC-RV32ST-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-RV32ST-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-RV32ST-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 1.51 DMIPS/MHz/Core
Coremark	up to 2.51 CoreMark/MHz/Core

Table 3.2 shows the example 4 processor cluster utilization with Xiling Ultrascale FPGA and ASIC gate equivalent. The processor system was configured with typical parameters without Branch Target Buffer, floating point unit and Physical Memory Protection. Register file and cache memories are not included.

Table 3.2: Xilinx Ultrascale and ASIC resource utilization.

Block	CLB LUTs	CLB REGs	DSPs	ASIC Gates [kGE]
Processor cluster:	31599	21444	20	333.9
Core 0	3192	2073	5	39.9
Core 1	3192	2073	5	39.9
Core 2	3192	2073	5	39.9
Core 3	3192	2073	5	39.9
I-Cache Controller (shared)	489	562	0	14.7
D-Cache Controller 0	563	848	0	13.1
D-Cache Controller 1	563	848	0	13.1
D-Cache Controller 2	563	848	0	13.1
D-Cache Controller 3	563	848	0	13.1
CSRs (shared)	1247	1375	0	16.3
PLIC	1295	796	0	6.8
CLINT	911	753	0	6.1
Multicore Controller	543	583	0	3.5
Power Management	51	153	0	2.7
Internal Interconnect	2491	1020	0	9.2
Debug System (shared)	9552	4671	0	62.6

4. BLOCK DIAGRAM
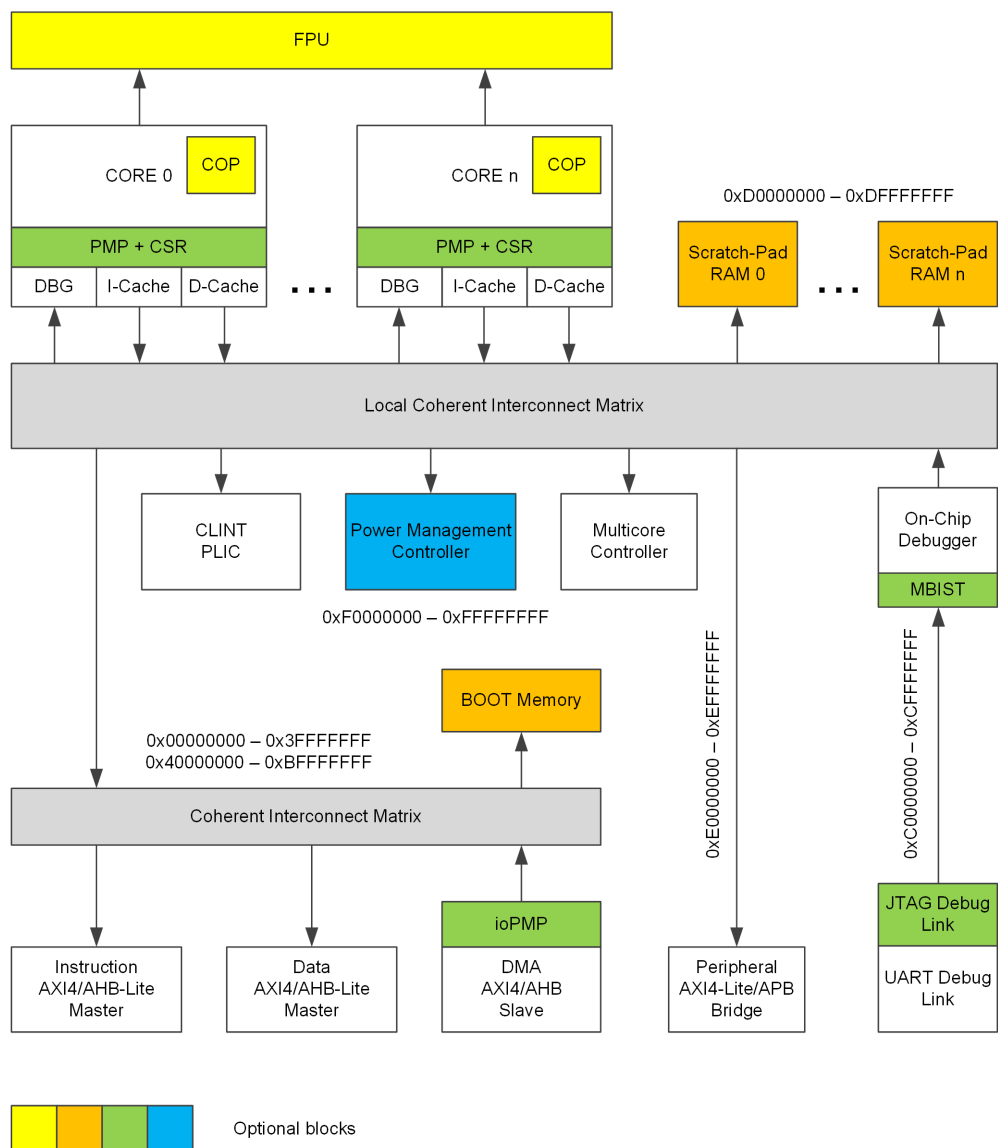
Figure 4.1: Processor block diagram.

5. MEMORY MAP

	Instruction Bus	Data Bus	Debug Bus

0xFFFFFFFF

Tightly-Coupled Peripherals
(Machine-Level Access)

0xF0000000
0xEFFFFFFF

AMBA APB
(Machine-Level Access)

0xE0000000
0xDFFFFFFF

SCRATCH-PAD RAM
(Uncachable Data)

0xD0000000
0xCFFFFFFF

DEBUG

0xC0000000
0xBFFFFFFF

RAM (Cachable Data)

0x40000000
0x3FFFFFFF

ROM (Cachable Instruction)

0x00000000
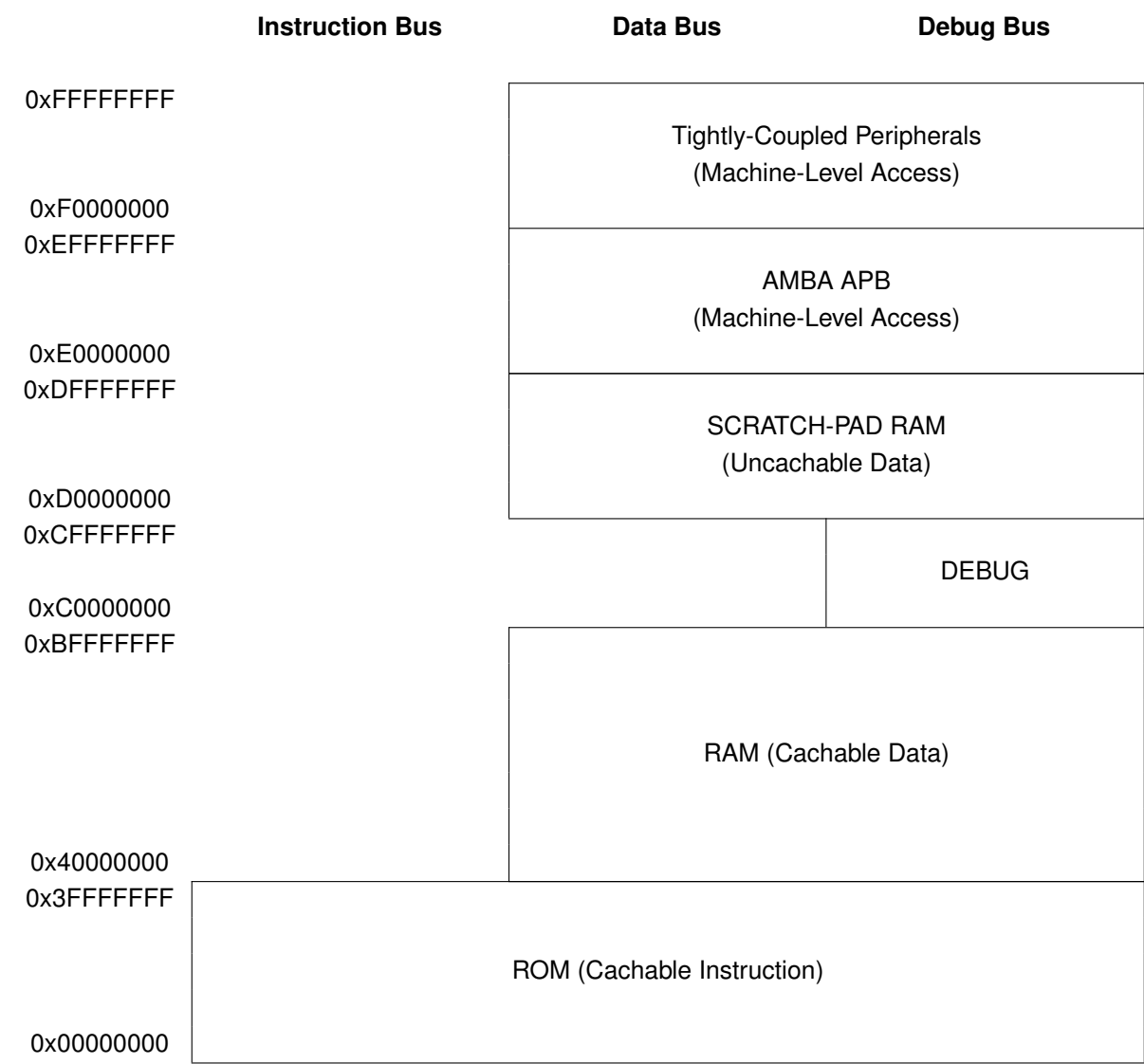
5.1. REGION DESCRIPTIONS

CC-RV32ST-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

31	7 6	0
BANK		OFFSET
R		R
0x09		0x7D

6.1.2. MACHINE ARCHITECTURE ID REGISTER

31	30	···	···	···	18	17	16
		···	···	···			
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

15	14	13 BTB[2:0] 11	10 ARCH[2:0] 8
		BTB[2:0]	ARCH[2:0]
R	R	R	R
0	0	5	2

7 BPRED[2:0] 5	4 MULFAST	3 MULIMP[1:0] 2	1 AROPT	0
BPRED[2:0]	MULFAST	MULIMP[1:0]	AROPT	
R	R	R	R	R
4	1	0	0	1

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.

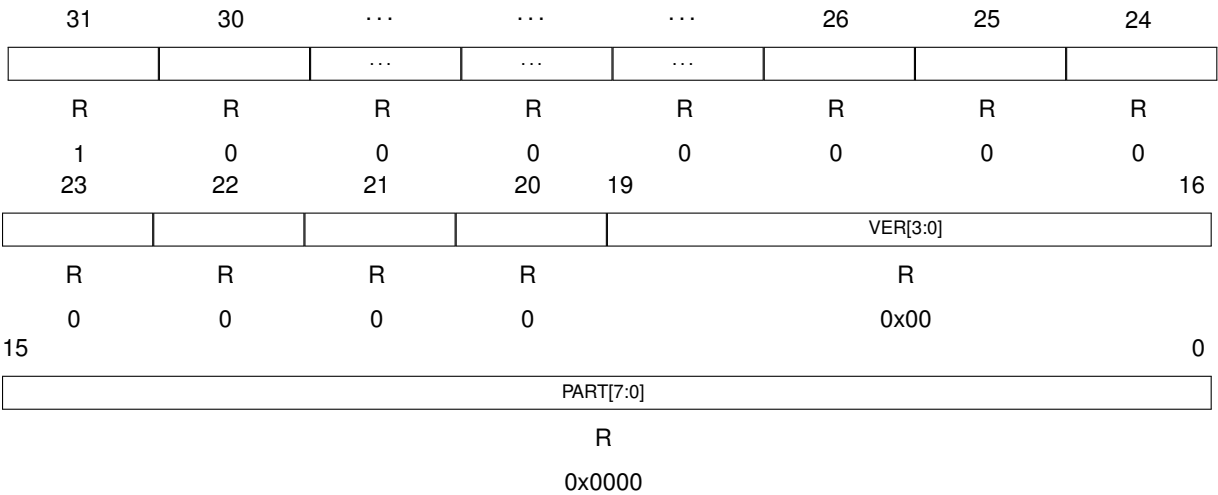   **3** PHT.

    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

31	30	$\cdots$	$\cdots$	$\cdots$	26	25	24
R	R	R	R	R	R	R	R
1	0	0	0	0	0	0	0

23	22	21	20	19	VER[3:0]		16
R	R	R	R	R			
0	0	0	0	0x00			

15	PART[7:0]	0
	R	
	0x0000	

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

31	30	0
INTR	EXC_CODE[30:0]	
R	R	
0	0	

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	≥ 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

31		29	28	27	26	25	24
ICWAY[2:0]			SPROT		PMP	USER	IRQ
R			R	R	R	R	R
2			1	0	1	1	1

23	22		18	17	16
PWD	SPRSIZE[4:0]			SPRAM	MCTRL
R	R			R	R
1	13			1	1

15		11	10	9		8
ICSIZE[4:0]			ICACHE	DMSIZE[4:3]		
R			R	R		
11			1	0		

7		5	4		0
DMSIZE[2:0]			IMSIZE[4:0]		
R			R		
0			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

31	30	29	28	27	26	25	24
							COMPR
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

23		21	20	19	18		17	16
BREP[2:0]			MULFAST	FPGA	MULIMP[1:0]			AROPT
R			R	R	R			R
4			1	0	0			0

15			12	11	10	9	8
FPUNUM[3:0]				PERFCNT	MBIST	ENDIAN	DCWAY[2]
R				R	R	R	R
1				1	0	0	2

7	6	5			1	0
DCWAY[1:0]		DCSIZE[4:0]				DCACHE
R		R				R
2		11				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

31	···	14	12 11	10	9	8
	···	BTB[2:0]	SYSBUS[1:0]	XPRAM	VITERBI	
R	R	R	R	R	R	
0	0	5	2	0	0	

7	4	3	2	0
GNSS_BANKS[3:0]		DCLS	ARCH[2:0]	
R		R	R	
0		0	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

31							24
CORE_ID[7:0]							
R							
CORE_ID							

23		20	11	10	9	8
BOOTREG[3:0]			GHRDIS	BTBDIS	NMIDIS	PHTDIS
R/W			R/W	R/W	R/W	R/W
0			0	0	1	0

7	6	5	4	3	2	1	0
		DCLS		SPROTEN			EXC
R	R	R	R	R/W	R	R	R
0	0	0	0	0	0	0	1

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

31	0
STACK_MIN[31:0]	

<div align="center">

R/W

0

</div>
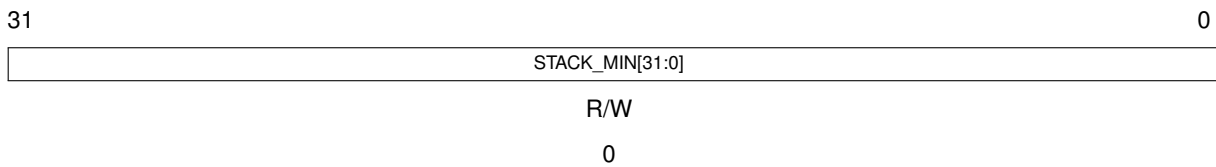
STACK_MIN[31:0] *Minimum Stack Pointer Value*

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

31	0
STACK_MAX[31:0]	

<div align="center">

R/W

0

</div>
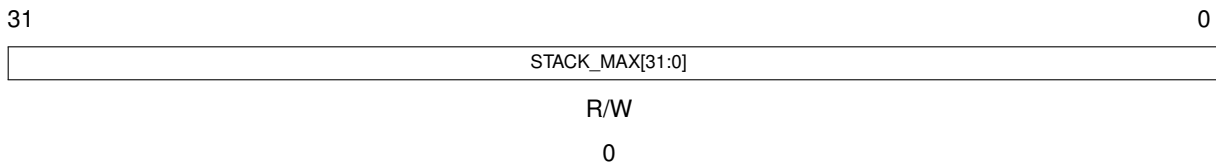
STACK_MAX[31:0] *Maximum Stack Pointer Value*

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

31	30	...	...	...	26	25	24
		...	...	...			
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

23	22	21	20	19			16
				BAUD_FRAC[3:0]			
R	R	R	R	R/W			
0	0	0	0	0			

15	8
BAUD_MANT[15:8]	
R/W	

7	0
BAUD_MANT[7:0]	
R/W	
36	

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

31							24
UNLOCK_KEY[7:0]							
W							
0							

23	22	···	···	···	2	1	0
		···	···	···			UNLOCK
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	1
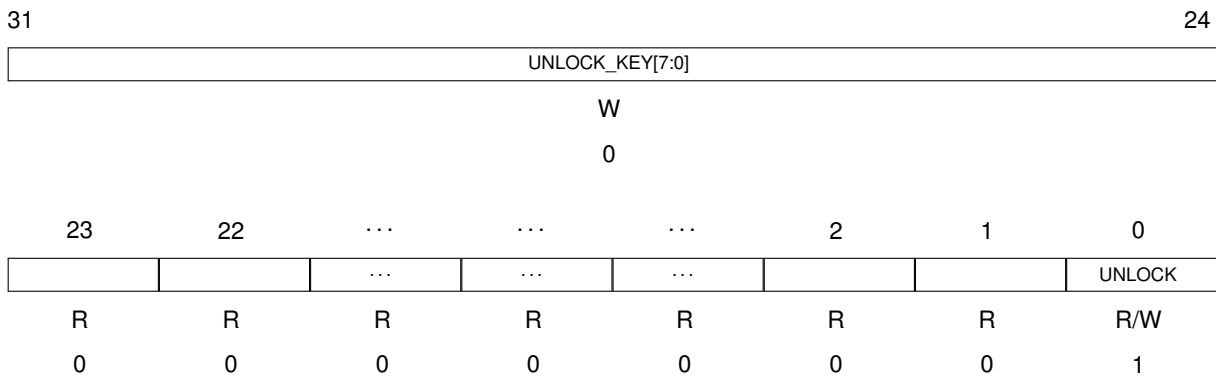
UNLOCK *ROM Unlock*

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. TIGHTLY-COUPLED PERIPHERALS

7.1. 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

7.2. MEMORY MAP

Table 7.1 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 7.1: 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

7.3. 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.
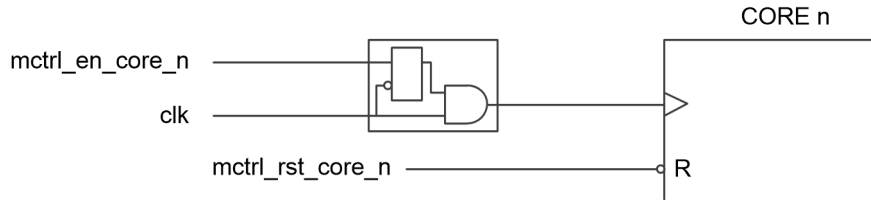
Figure 7.1: Simplified processor cores clock gating scheme.

7.3.1. REGISTERS LIST

Table 7.2: 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
. . .	. . .	. . .

7.3.2. 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.

7.3.3. CORE NUMBER REGISTER

Address: 0x04

Type: shared

31 0
CORE_NUM[31:0]
R
1

CORE_NUM[31:0] *Core Number*

Stores the number of processor cores.

7.3.4. SHUTDOWN REGISTER

Address: 0x08

Type: shared

31							24
SHDN_KEY[7:0]							
W							
0							

23	22	21	20	19	18	17	16
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

15	14	···	···	···	2	1	0
SHDN15	SHDN14	···	···	···	SHDN2	SHDN1	
W	W	W	W	W	W	W	R
0	0	0	0	0	0	0	0
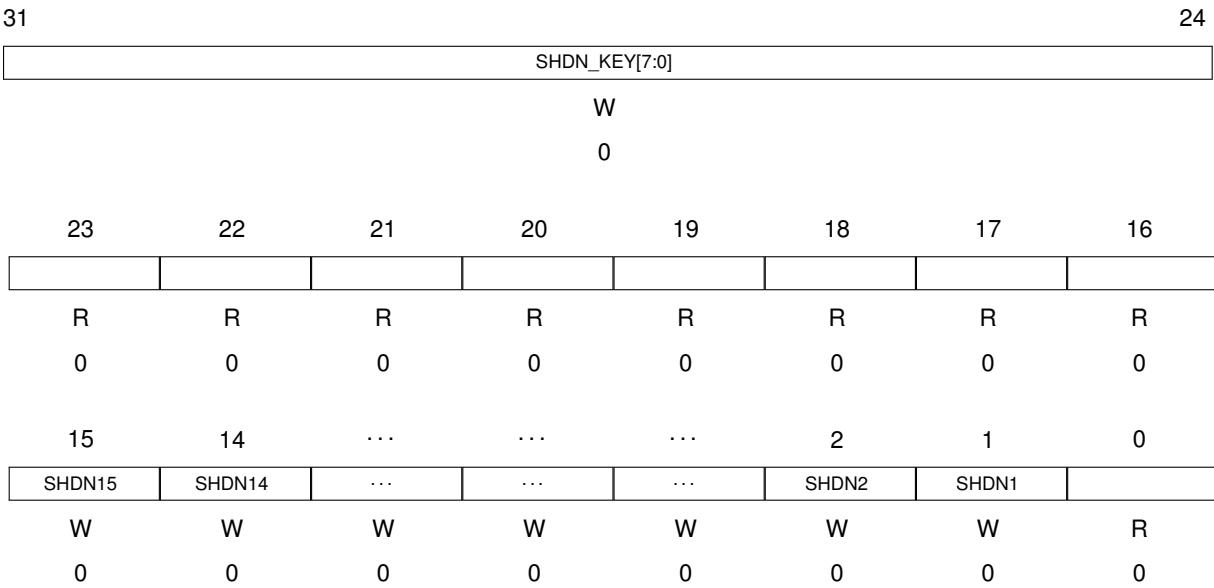
SHDN*n* *Shut down Core n*

   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.

7.3.5. START ADDRESS REGISTERS

Address: 0x10

31	0
CORE_0_ADDR[31:0]	

N/A

0

Address: 0x14

Type: shared

31	0
CORE_1_ADDR[31:0]	

R/W

0

Address: 0x18

Type: shared

31	0
CORE_2_ADDR[31:0]	

R/W

0

Address: . . .

31	0
CORE_n_ADDR[31:0]	

R/W

0

CORE_n_ADDR[31:0] *Core n Start Address*

Stores the Core *n* start address.

7.3.6. CORE RUN REGISTERS

Address: 0xD8

31	30	$\cdots$	$\cdots$	8	7	0
		$\cdots$	$\cdots$		CORE_0_RUN[7:0]	
R	R	R	R	R	N/A	
0	0	0	0	0	0	

Address: 0xDC

Type: shared

31	30	$\cdots$	$\cdots$	8	7	0
		$\cdots$	$\cdots$		CORE_1_RUN[7:0]	
R	R	R	R	R	W	
0	0	0	0	0	0	

Address: 0xE0

Type: shared

31	30	$\cdots$	$\cdots$	8	7	0
		$\cdots$	$\cdots$		CORE_2_RUN[7:0]	
R	R	R	R	R	W	
0	0	0	0	0	0	

Address: . . .

31	30	$\cdots$	$\cdots$	8	7	0
		$\cdots$	$\cdots$		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.

7.4. 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.

7.4.1. REGISTERS LIST

Table 7.3: 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

7.4.2. RESET CAUSE REGISTER

Address: 0x04

Type: shared

31	30	···	···	···	10	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
				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.

7.4.3. RESET REGISTER

Address: 0x08

Type: shared

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

PWDRST[7:0] *Power Management Reset*

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

7.4.4. DEEP RESET REGISTER

Address: 0x0C

Type: shared

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

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.

7.4.5. INFO REGISTER

Address: 0x14

Type: shared

31	30	···	···	···	10	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
					SYSPWDEN	COREPWDEN	MAINPWDEN
R	R	R	R	R	R	R	R
0	0	0	0	1	0	0	0

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.

7.5. 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).

7.5.1. 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 7.2 presents the simplified stack exception generation datapath.
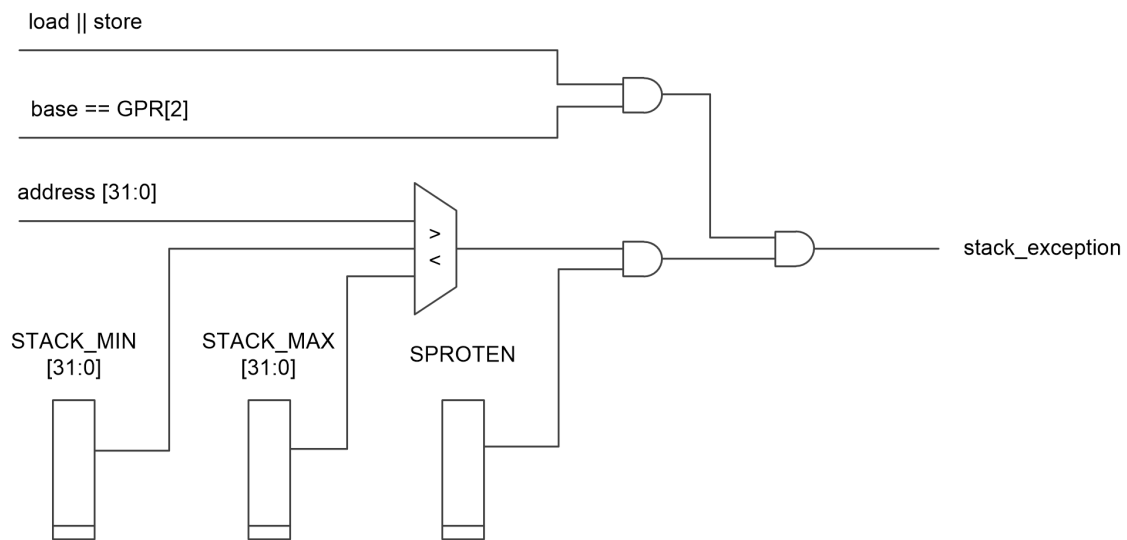
Figure 7.2: Simplified stack exception generation datapath.

7.5.2. 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.

Platform-Level Interrupt Controller prioritizes and distributes global interrupts in a processor system. Figure 7.3 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.
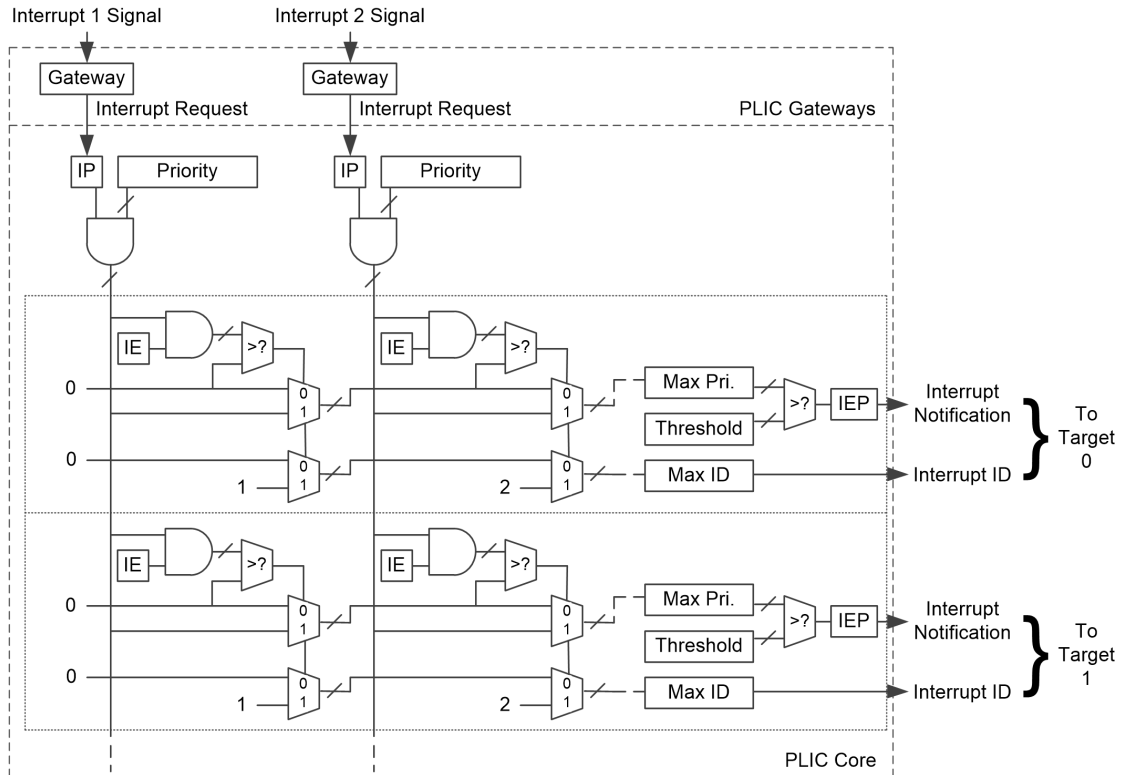
Figure 7.3: Platform-Level Interrupt Controller (PLIC) conceptual block diagram.

7.6.1. REGISTERS LIST

Table 7.4: 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

7.6.2. INTERRUPT PRIORITY REGISTERS

Address: 0x04

Type: shared

31	. . .	. . .	. . .	3	2		0
	. . .	. . .	. . .		IRQ_PRIOR1[2:0]		
R	R	R	R	R	R/W		
0	0	0	0	0	1		

Address: 0x08

Type: shared

31	. . .	. . .	. . .	3	2		0
	. . .	. . .	. . .		IRQ_PRIOR1[2:0]		
R	R	R	R	R	R/W		
0	0	0	0	0	1		

Address: . . .

31	. . .	. . .	. . .	3	2		0
	. . .	. . .	. . .		IRQ_PRIORn[2:0]		
R	R	R	R	R	R/W		
0	0	0	0	0	1		

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.

7.6.3. 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.

7.6.4. 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.

7.6.5. 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.

7.6.6. 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.

7.7. 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.

7.7.1. REGISTERS LIST

Table 7.5: 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

7.7.2. 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.

7.7.3. INTER-CORE INTERRUPT TRIGGER REGISTER

Address: 0x04

Type: exclusive

31	30	$\cdots$	$\cdots$	$\cdots$	2	1	0
CT31	CT30	$\cdots$	$\cdots$	$\cdots$	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.

7.7.4. INTER-CORE INTERRUPT FLAGS REGISTER

Address: 0x08

Type: exclusive

31	30	$\cdots$	$\cdots$	$\cdots$	2	1	0
CTF31	CTF30	$\cdots$	$\cdots$	$\cdots$	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.

7.8. CORE LOCAL INTERRUPTOR CONTROLLER

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

7.8.1. REGISTERS LIST

Table 7.6: 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

7.8.2. 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.

7.8.3. MACHINE TIME COMPARE

Address: 0x080

Type: shared

31	0
MTIMECMP_LO[31:0]	

R/W

0
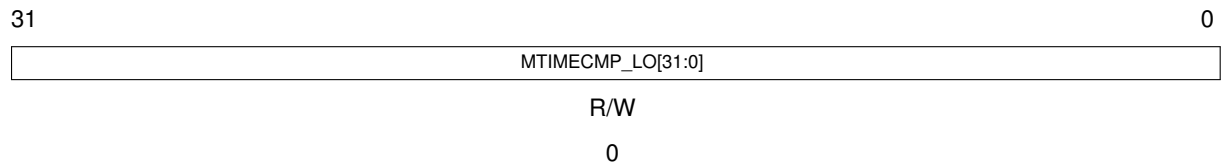
Address: 0x084

Type: shared

31	0
MTIMECMP_HI[31:0]	

R/W

0
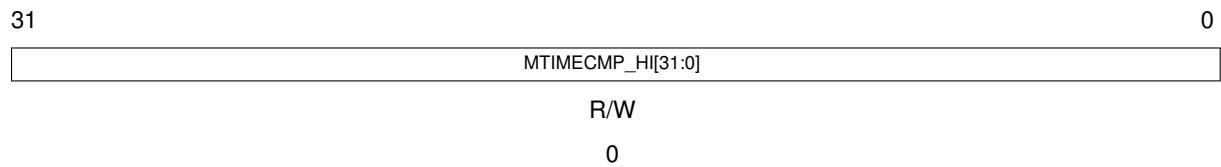
MTIMECMP[63:0] *Machine Time Compare*

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.

7.8.4. MACHINE TIME REGISTER

Address: 0x0180

Type: exclusive

31	0
MTIME_LO[31:0]	

R/W

0
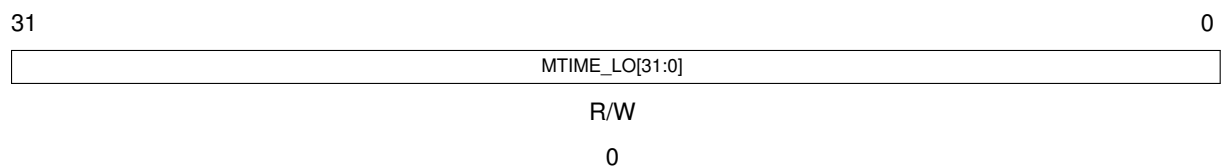
Address: 0x0184

Type: exclusive

31	0
MTIME_HI[31:0]	

R/W

0
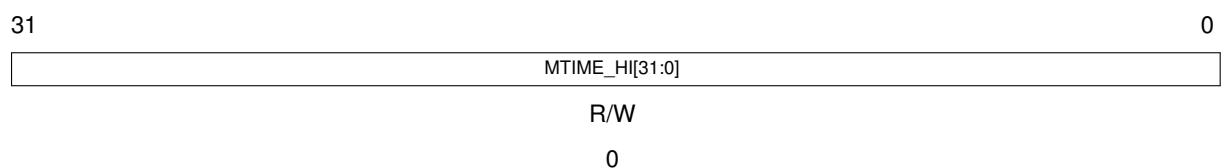
MTIME[63:0] *Machine Time Register*

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

7.8.5. MACHINE TIME CONFIG REGISTER

Address: 0x188

Type: exclusive

31	30	29	28	27	26	25	24
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

23	22			MTIMEBITS[6:0]			16
R				R			
0				56			

15	14	13	12	11	10	9	8
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

7			MTIMESRC[3:0]	4	3	2	1	0 MTIMEEN
R/W					R	R	R	R/W
0					0	0	0	0

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.

7.9. 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 7.4 and 7.5 presents the cache address mapping examples.

31		10 9	5 4	0
	TAG		INDEX	OFFSET

Figure 7.4: 1 KiB/way, 32 bytes/line address mapping example.

31		12 11	4 3	0
	TAG		INDEX	OFFSET

Figure 7.5: 4 KiB/way, 16 bytes/line address mapping example.

Figures 7.6 and 7.7 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.

TAGSIZE+1		2	1	0
	TAG		LRR	VALID

Figure 7.6: Tag memory entry with LRR bit.

TAGSIZE		1	0
	TAG		VALID

Figure 7.7: 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.

7.9.1. REGISTERS LIST

Table 7.7: Instruction Cache Controller registers list.

Address Offset	Register	Name
0xF0070000	STCR	Status and Control Register
0xF0070004	FLUSH	Flush Register
0xF0070008	INFO	Info Register

7.9.2. STATUS AND CONTROL REGISTER

Address: 0x00

Type: exclusive

31	···	···	4	3	2	1	0
	···	···					ICEN
R	R	R	R	R	R	R	R/W
0	0	0	0	0	0	0	0

ICEN *Instruction Cache Enable*

Setting this bit enables instruction cache.

7.9.3. FLUSH REGISTER

Address: 0x04

Type: exclusive

31	0
FLUSH[31:0]	
W	
N/A	

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

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

7.9.4. INFO REGISTER

Address: 0x08

Type: shared

31			28	27	26	25	24
VER[3:0]							
R				R	R	R	R
0				0	0	0	0

23	22	21		16
		TAGSIZE[6:1]		
R	R	R		
0	0	22		

15	14	13	12	11	8
TAGSIZE[0]	ICALG[1:0]		VALID	ICLSIZE[4:0]	
R	R		R	R	
22	0		1	6	

7	3	2	0
ICSIZE[4:0]		ICWAY[2:0]	
R		R	
11		2	

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.

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 7.8 and 7.9 presents the cache address mapping examples.

31		10	9	5	4	0
	TAG			INDEX		OFFSET

Figure 7.8: 1 KiB/way, 32 bytes/line address mapping example.

31		12	11	4	3	0
	TAG			INDEX		OFFSET

Figure 7.9: 4 KiB/way, 16 bytes/line address mapping example.

Figures 7.10 and 7.11 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.

TAGSIZE+1		2	1	0
	TAG		LRR	VALID

Figure 7.10: Tag memory entry with LRR bit.

TAGSIZE		1	0
	TAG		VALID

Figure 7.11: 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.

7.10.1. REGISTERS LIST

Table 7.8: Data Cache Controller registers list.

Address Offset	Register	Name
0xF0072000	STCR	Status and Control Register
0xF0072004	FLUSH	Flush Register
0xF0072008	INFO	Info Register

7.10.2. STATUS AND CONTROL REGISTER

Address: 0x00

Type: exclusive

31	⋯	⋯	4	3	2	1	0
	⋯	⋯			BUSY	FLUSH	DCEN
R	R	R	R	R	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.

7.10.3. FLUSH REGISTER

Address: 0x04

Type: exclusive

31 0
FLUSH[31:0]
W
N/A

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

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

7.10.4. INFO REGISTER

Address: 0x08

Type: shared

31		28	27		24
VER[3:0]			STOREBUF		
R			R		
0			6		

23	22	21		16
		TAGSIZE[6:1]		
R	R	R		
0	0	22		

15	14	13	12	11	8
TAGSIZE[0]	DCALG[1:0]		VALID	DCLSIZE[4:0]	
R	R		R	R	
22	0		1	6	

7	3	2	0
DCSIZE[4:0]		DCWAY[2:0]	
R		R	
11		2	

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.

7.11. 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.

7.11.1. REGISTERS LIST

Table 7.9: 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

7.11.2. COMMAND REGISTER

Address: 0x00

Type: shared

31	30	$\cdots$	$\cdots$	$\cdots$	10	9	8
		$\cdots$	$\cdots$	$\cdots$			
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

7	0
CMD[7:0]	
R/W	
0	

CMD[7:0] *Command*

Executed command:

0 Idle.

109 Read.

119 Write.

160 Context.

7.11.3. ADDRESS REGISTER

Address: 0x04

Type: shared

31	0
ADDR[31:0]	

R/W

0
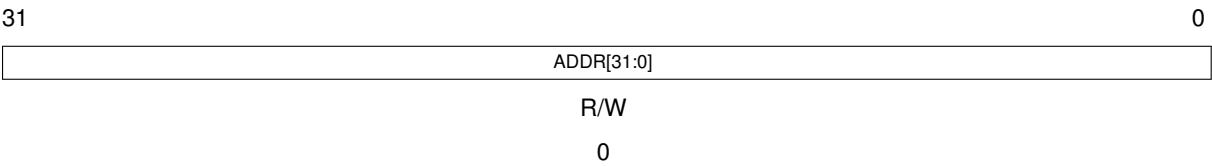
ADDR[31:0] *Address*

Memory address to read/write.

7.11.4. WRITE DATA REGISTER

Address: 0x08

Type: shared

31	0
DATA[31:0]	

R/W

0
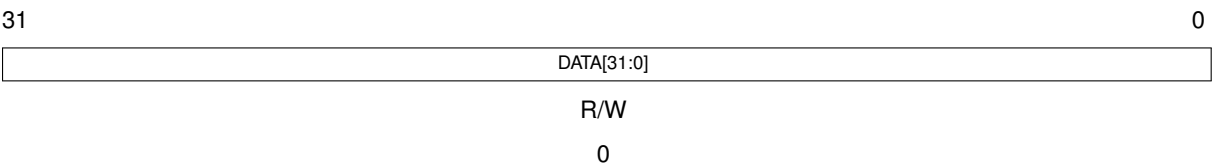
DATA[31:0] *Write Data*

Data to be written.

7.11.5. READ DATA REGISTER

Address: 0x0C

Type: shared

31   0

DATA[31:0]

R

0
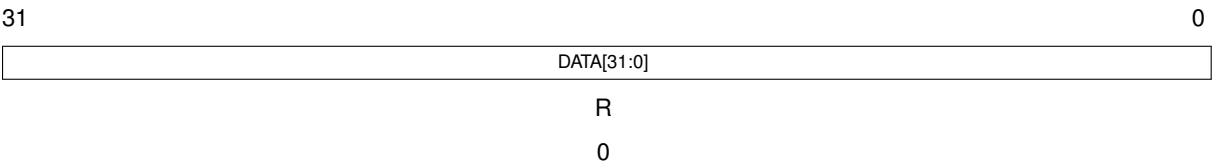
DATA[31:0] *Read Data*

Data read from the given address.

8. ON-CHIP DEBUGGER

CC-RV32ST-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 8.1 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 8.1: 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

8.1. VERSION REGISTER

Address: 0xC0000024

31			28	27			24
DEBUG_START[3:0]				AMBA_START[3:0]			
R				R			
0xC				0xE			

23			20	19			16
PERIPH_START[3:0]				RAM_START[3:0]			
R				R			
0xF				0x4			

15			12	11			8
ROM_START[3:0]				PROC_ARCH[3:0]			
R				R			
0x0				2			

7			4	3	2	1	0
BURST_SIZE[3:0]						MBIST_EN	BURST_EN
R				R	R	R	R
12				0	1	0	1

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.

8.2. 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.

8.2.1. 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.

HOST	BREAK_FRAME
TARGET	0x55 (U)
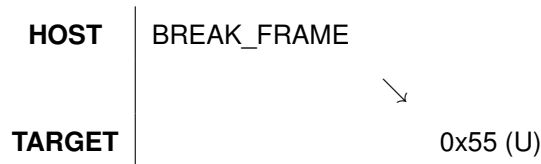
8.2.2. DEBUG VERSION

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.

HOST	0x76 (v)
TARGET	VERSION
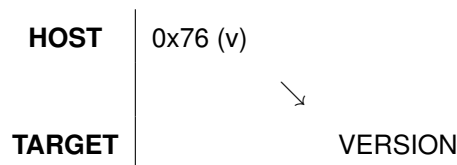
8.2.3. DEBUG CONTEXT

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.

HOST	8'b10100001		**HOST**	8'b10100110
TARGET	0x06 (<ACK>)		**TARGET**	0x06 (<ACK>)
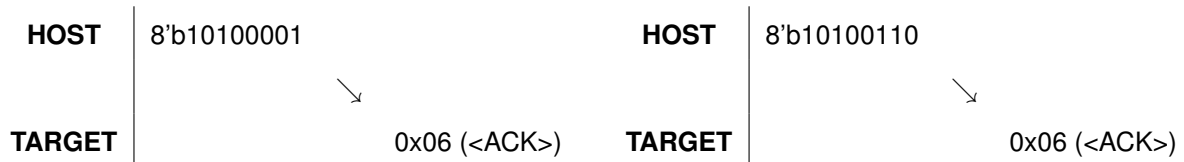
8.2.4. DETECTING DEBUG MODE

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.

HOST	0x42 (B)
TARGET	0x06 (<ACK>)
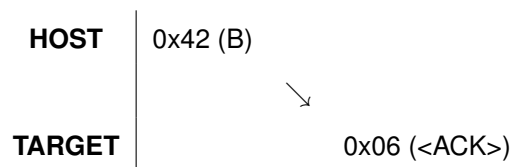
Otherwise the target will send one state frame for every processor core in the system.

HOST	0x42 (B)
TARGET	CPU_0_FRAME(t) $\rightarrow$ CPU_1_FRAME(t) $\rightarrow$ ... $\rightarrow$ CPU_n_FRAME(t)
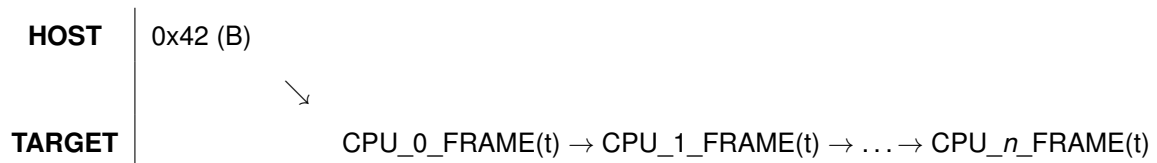
The 0x42 (B) char can also be used to resend the current processor state frame after losing sync with the target.

HOST	0x42 (B)	0x42 (B)
TARGET	CPU_0_FRAME(t) ...	CPU_0_FRAME(t) ...
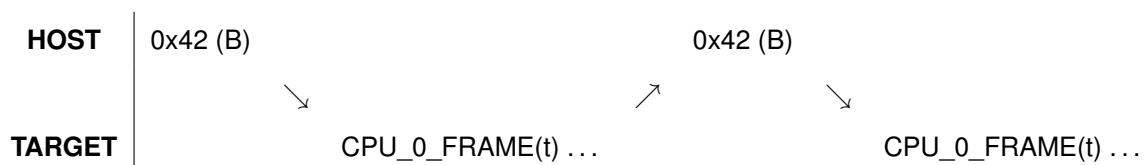
8.2.5. PROCESSOR CORE STATE FRAME

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.

0	1	2 5	6 9	10 13	14 17	18 21	22 25
0x7E (~)	ID	PC	INST	RESULT	WDATA	ADDR	LRESULT

0	1
0x7E (~)	ID
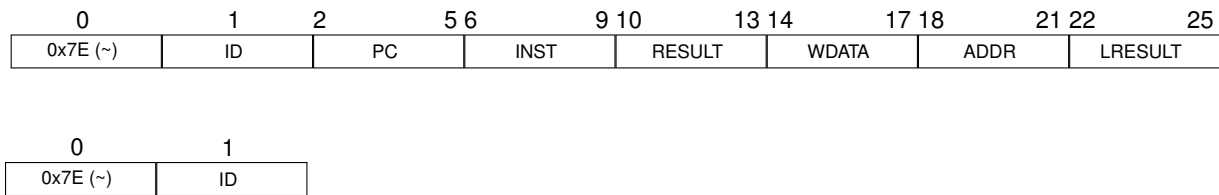
Figure 8.1: Long and short processor core state frame structure.

Table 8.2: 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

8.2.6. 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.

HOST

TARGET | CPU_0_FRAME(t) $\rightarrow$ CPU_1_FRAME(t) $\rightarrow$ ... $\rightarrow$ CPU_n_FRAME(t)

8.2.7. 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.

HOST | 0x62 (b)

TARGET | CPU_0_FRAME(t) $\rightarrow$ CPU_1_FRAME(t) $\rightarrow$ ... $\rightarrow$ CPU_n_FRAME(t)
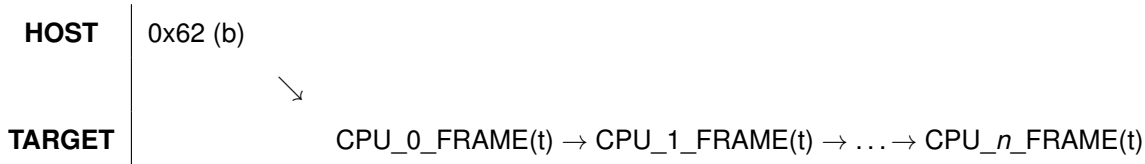
8.2.8. STEP COMMAND

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.

HOST | 0x73 (s)   0x73 (s)

TARGET | CPU_0_FRAME(t+1) ...   CPU_0_FRAME(t+2) ...
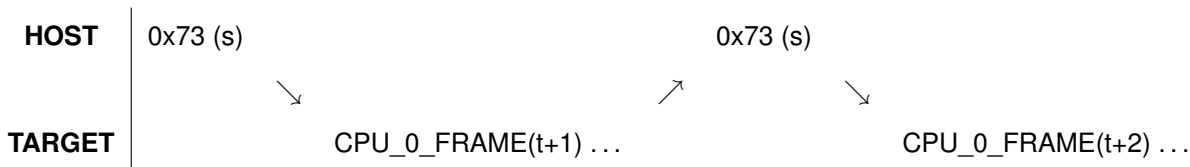
8.2.9. LEAVING DEBUG MODE

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

8.2.10. 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.

HOST	0x66 (f)
TARGET	0x06 (<ACK>)

8.2.11. 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.

HOST	0x72 (r)
TARGET	0x06 (<ACK>)

8.2.12. 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.

HOST	0x52 (R)
TARGET	0x06 (<ACK>)

8.2.13. 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.

```
HOST    | 0x61 (a) → ADDRESS
        |                            ↘
TARGET  |                                0x06 (<ACK>)
```

8.2.14. 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.
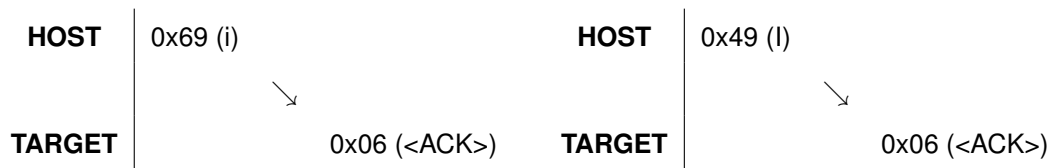
```
HOST    | 0x69 (i)                        HOST    | 0x49 (I)
        |              ↘                           |              ↘
TARGET  |                  0x06 (<ACK>)   TARGET  |                  0x06 (<ACK>)
```

8.2.15. 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).

```
HOST    | 0x61 (a) → ADDRESS                      0x6D (m)
        |              ↘              ↗              ↘
TARGET  |                  0x06 (<ACK>)                DATA(ADDRESS)
```
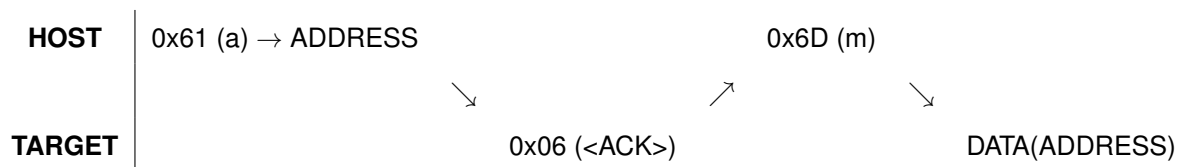
When the autoincrementation is enabled every Read Memory Command execution will dump consecutive memory location content.

HOST	0x6D (m)		0x6D (m)	
		↘	↗	↘
TARGET		DATA(ADDRESS)		DATA(ADDRESS+4)

8.2.16. 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).

HOST	0x77 (w) → DATA	
		↘
TARGET		0x06 (<ACK>)
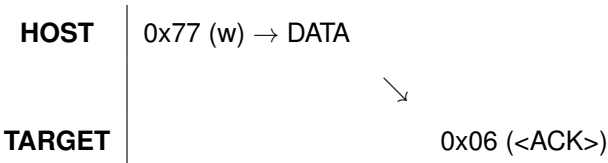
When the autoincrementation is enabled every Memory Write Command execution will store data to the consecutive memory location.

HOST	0x77 (w) → DATA_0		0x77 (w) → DATA_1	
		↘	↗	↘
TARGET		0x06 (<ACK>)		0x06 (<ACK>)
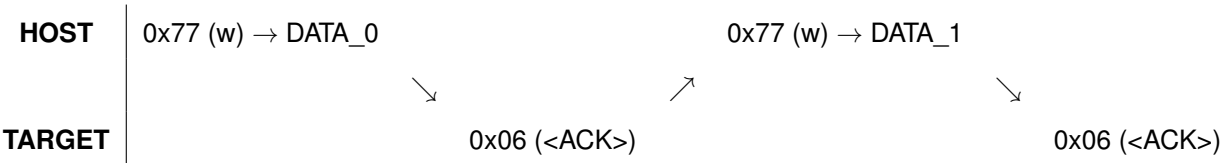
8.2.17. BURST COUNTER REGISTER

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.

HOST	0x61 (a) → 0xC0000020		0x77 (w) → 0x0000000C	
		↘	↗	↘
TARGET		0x06 (<ACK>)		0x06 (<ACK>)

HOST	0x61 (a) → ADDRESS
TARGET	0x06 (<ACK>)

HOST	0x6D (m)
TARGET	DATA(ADDRESS) → DATA(ADDRESS+4) → DATA(ADDRESS+8)
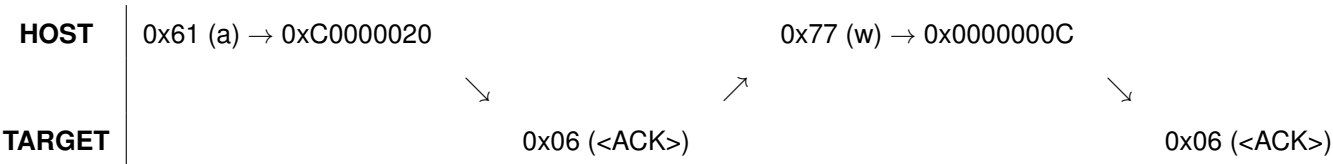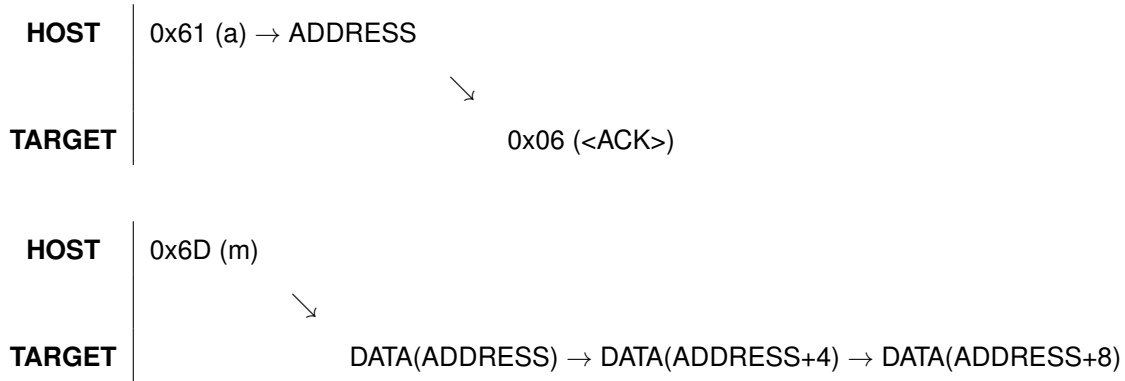
Similarly, when autoincrementation is enabled and Burst Counter Register is set to the value greater than 4, the memory write operation can be instantly followed by a number of 32-bit data that will be stored in consecutive memory locations.

HOST	0x61 (a) → 0xC0000020	0x77 (w) → 0x00000014
TARGET	0x06 (<ACK>)	0x06 (<ACK>)

HOST	0x61 (a) → ADDRESS
TARGET	0x06 (<ACK>)

HOST	0x77 (w) → DATA_0 → DATA_1 → DATA_2 → DATA_3 → DATA_4
TARGET	0x06 (<ACK>)
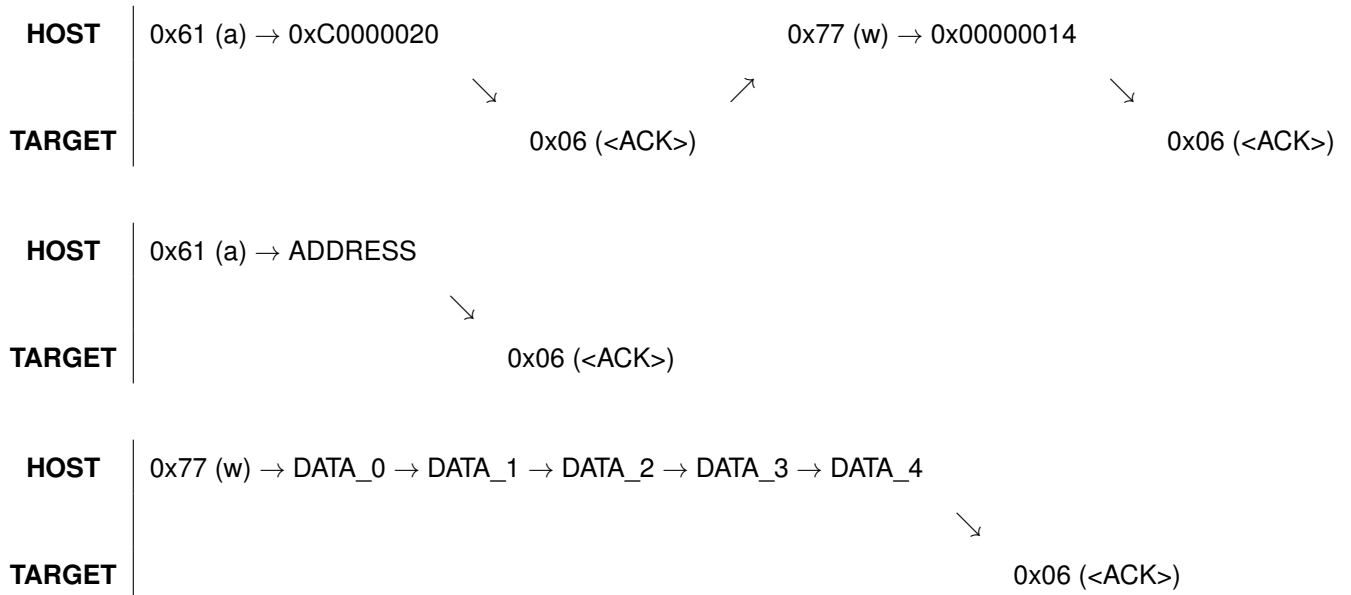
8.2.18. MBIST RUN COMMAND

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.

HOST	0x4D (M)
TARGET	0x06 (<ACK>)
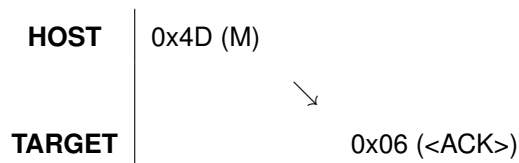
8.2.19. UART DEBUG LINK PINOUT

Table 8.3: 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	

8.3. JTAG DEBUG LINK

JTAG interface provides a communication link with the debug host using simple communication protocol which translates JTAG instructions to On-Chip Debugger commands. The JTAG interface supports the 4-bit instructions shown in Table 8.4.

Table 8.4: 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.

8.3.1. 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 8.2.

47	40 39	32 31	0
CRC-8	COMMAND	DATA	

TDI →

47	8 7	0
	RESPONSE	

TDO →
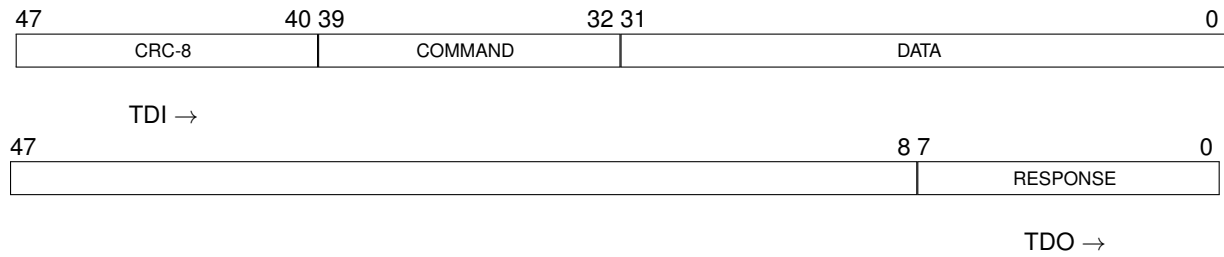
Figure 8.2: Debug Command Register structure.

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.

8.3.2. 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 8.3.

39	32	31	0
CRC-8		DATA	

TDO →
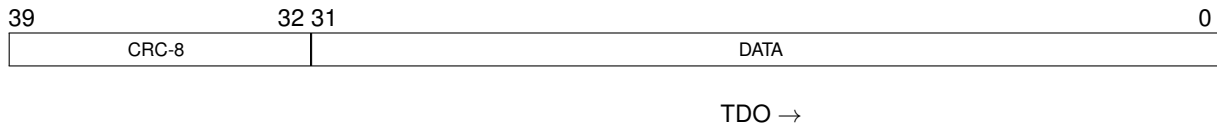
Figure 8.3: Debug Data Register structure.

8.3.3. 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:

x31	x00	x99	x00	x99	x00	199	0
CRC-32		CPU_0_FRAME		...		CPU_*n*_FRAME	

TDO →
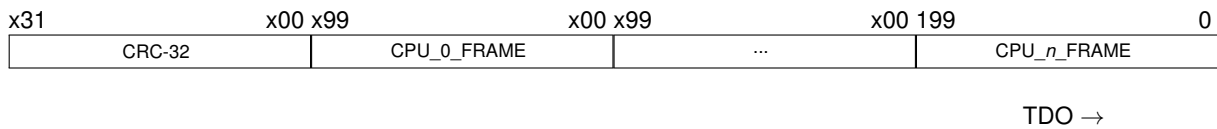
Figure 8.4: Debug Stauts Register structure.

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.

199	192	191	160	159	128	127	96	95	64	63	32	31	0
STATUS		PC		INST		RESULT		WDATA		ADDR		LRESULT	

TDO →
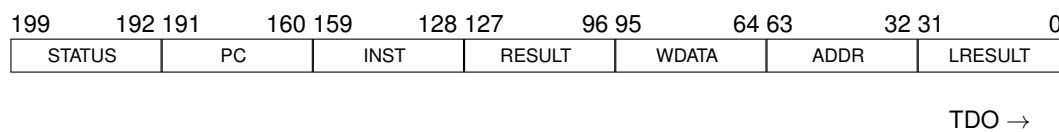
Figure 8.5: JTAG processor core state frame structure.

Table 8.5: 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

8.3.4. 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.

47	40 39	32 31	0
8'b00011010	8'b01110110 (v)	32'd0	

TDI →

47	8 7	0
		8'b00000110

TDO →
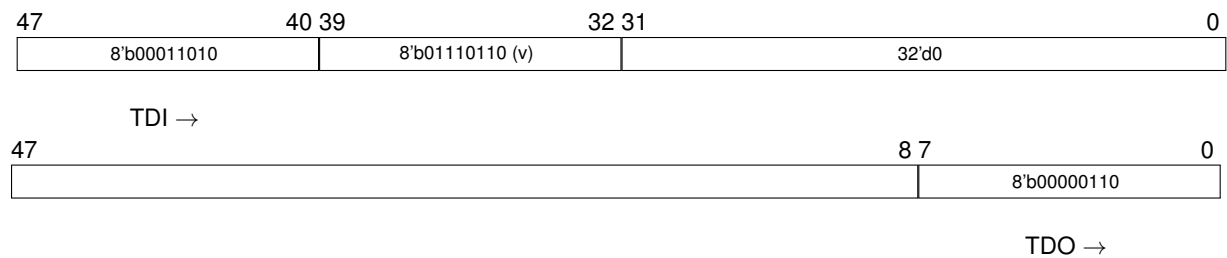
Figure 8.6: Debug Version Command structure.

8.3.5. 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

8.3.6. 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).
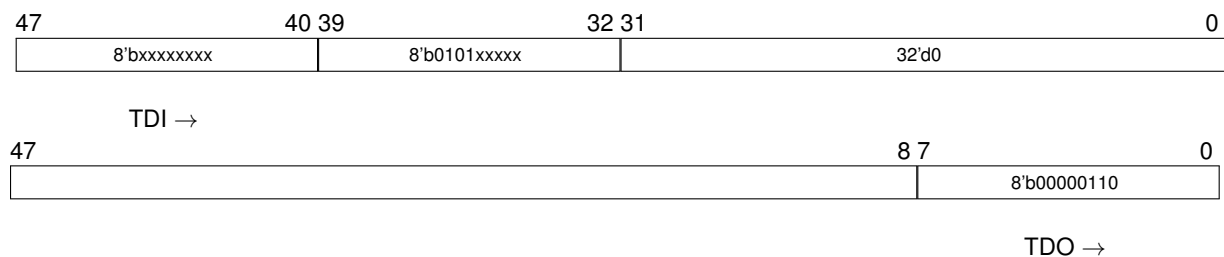
Figure 8.7: Debug Context Command structure.

8.3.7. 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.

47	40 39	32 31	0
8'b10100100	8'b01100010 (b)	32'd0	

TDI →

47	8 7	0
	8'b00000110	

TDO →
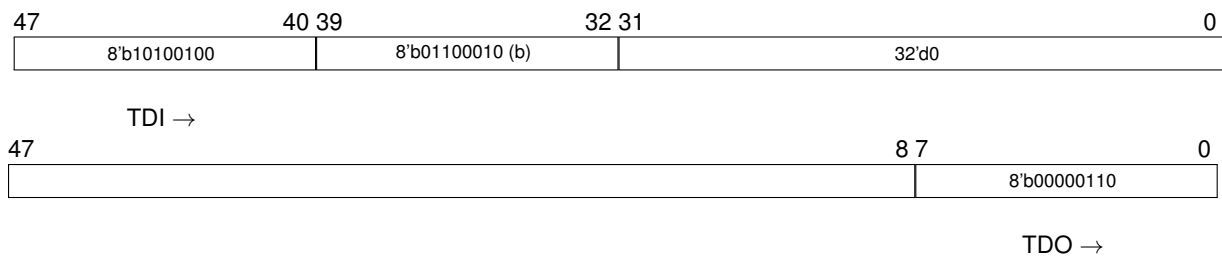
Figure 8.8: Break Command structure.

8.3.8. 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.

47	40 39	32 31	0
8'b11101100	8'b01110011 (s)	32'd0	

TDI →

47	8 7	0
	8'b00000110	

TDO →
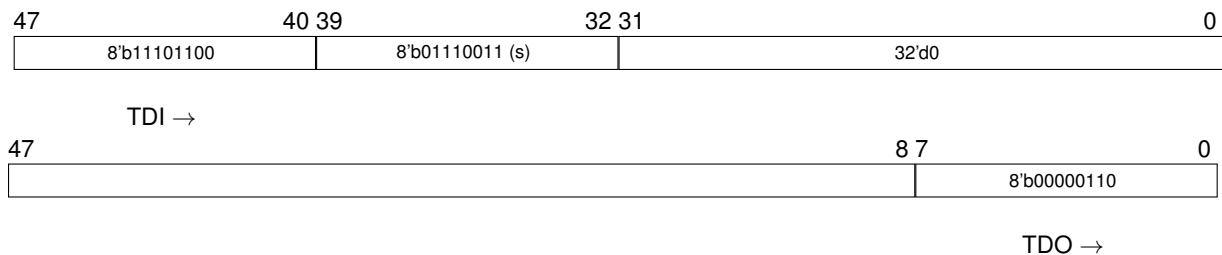
Figure 8.9: Step Command structure.

8.3.9. 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

8.3.10. 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.

47		40 39		32 31		0
8'b10000010		8'b01100110 (f)		32'd0		

TDI →

47			8 7		0
				8'b00000110	

TDO →
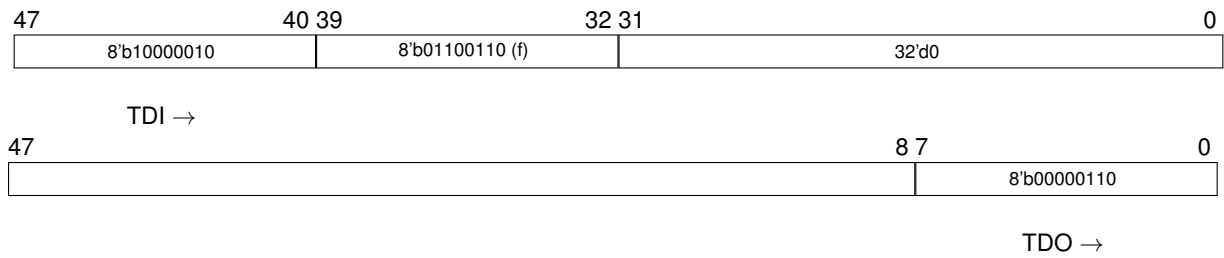
Figure 8.10: Free Running Command structure.

8.3.11. 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.

47		40 39		32 31		0
8'b00111100		8'b01110010 (r)		32'd0		

TDI →

47			8 7		0
				8'b00000110	

TDO →
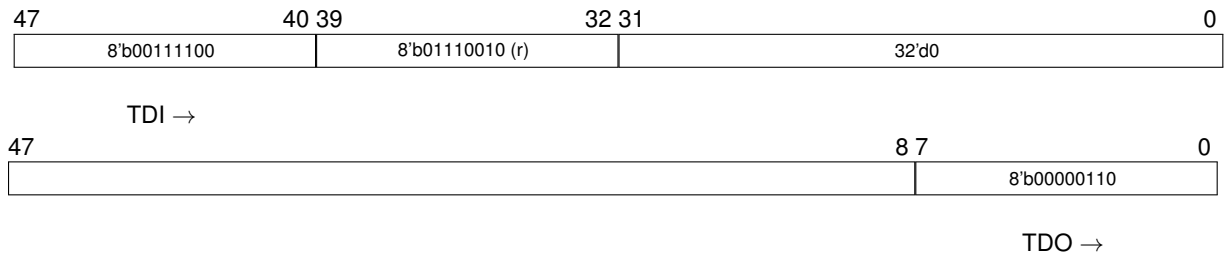
Figure 8.11: Reset Command structure.

8.3.12. 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.

47		40 39		32 31		0
8'b10111111		8'b01010010 (R)		32'd0		

TDI →

47			8 7		0
				8'b00000110	

TDO →
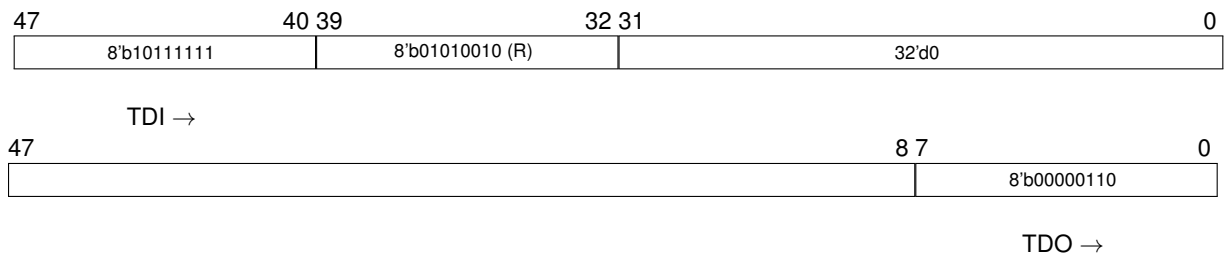
Figure 8.12: Debugger Reset Command structure.

8.3.13. 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.

47	40 39	32 31	0
8'b01100111	8'b01100001 (a)	32'd0	

TDI →

47	8 7	0
	8'b00000110	

TDO →
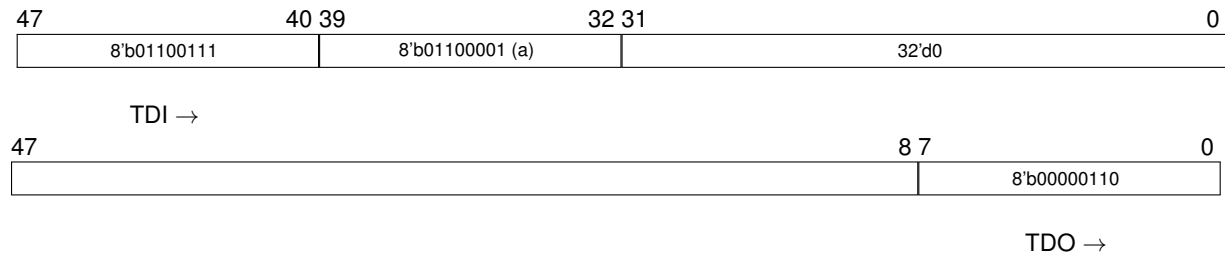
Figure 8.13: Address Command structure.

8.3.14. 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.

47	40 39	32 31	0
8'b00101011	8'b01101001 (i)	32'd0	

TDI →

47	8 7	0
	8'b00000110	

TDO →
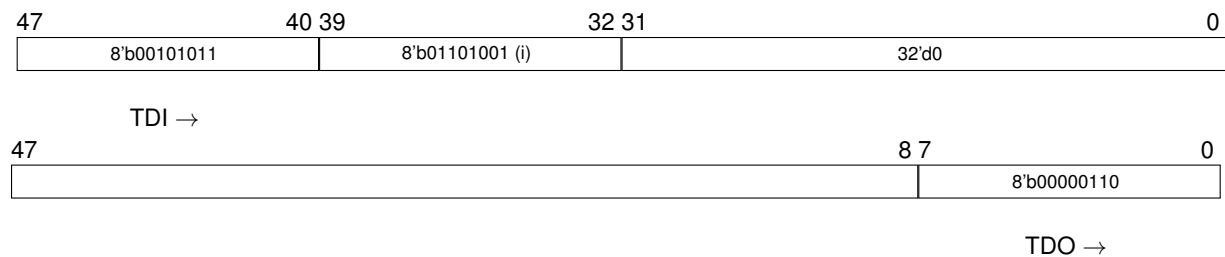
Figure 8.14: Auto-Increment Off Command structure.

47	40 39	32 31	0
8'b10101000	8'b01001001 (I)	32'd0	

TDI →

47	8 7	0
	8'b00000110	

TDO →
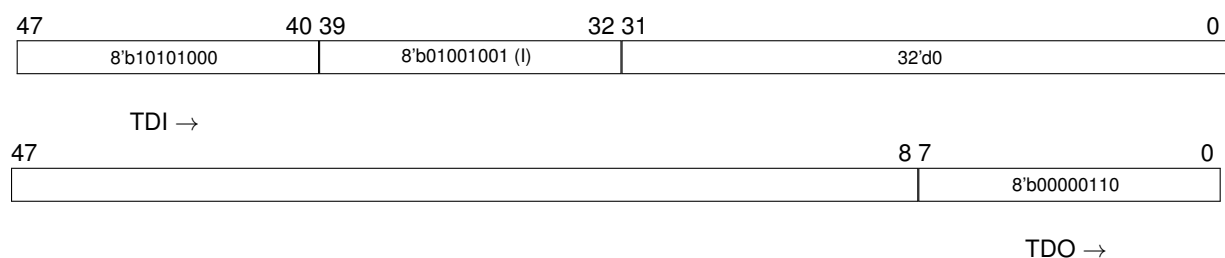
Figure 8.15: Auto-Increment On Command structure.

8.3.15. 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).

47 40	39 32	31 0
8'b00001101	8'b01101101 (m)	32'd0

TDI →

47 8	7 0
	8'b00000110

TDO →
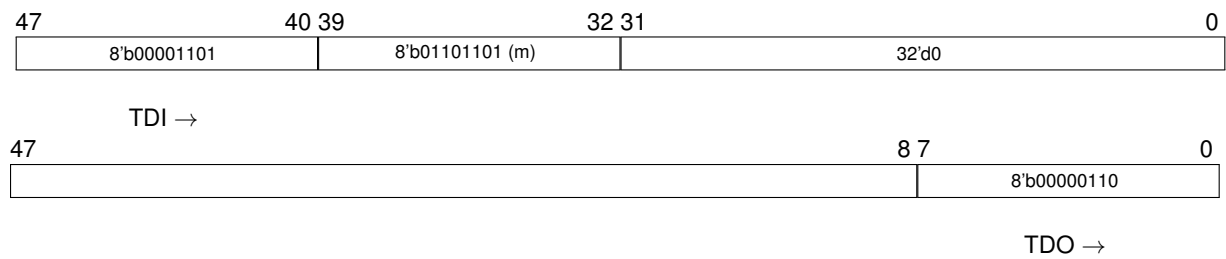
Figure 8.16: Memory Read Command structure.

8.3.16. 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.

47 40	39 32	31 0
CRC-8	8'b01110111 (w)	DATA

TDI →

47 8	7 0
	8'b00000110

TDO →
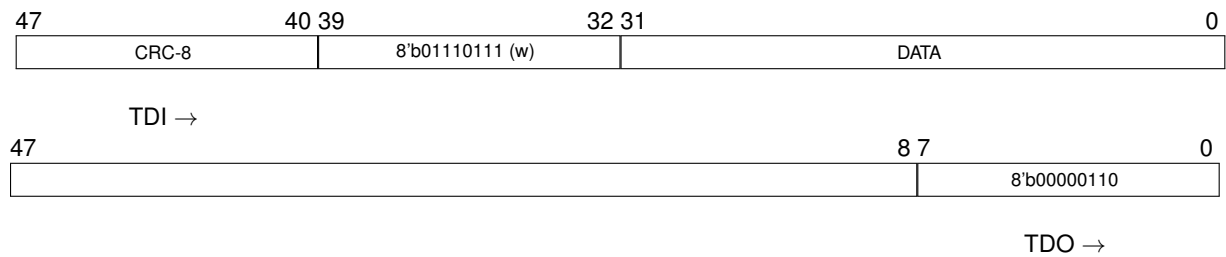
Figure 8.17: Memory Write Command structure.

8.3.17. 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.

47	40 39	32 31	0
CRC-8	8'b01001101 (M)	DATA	

TDI →

47	8 7	0
	8'b00000110	

TDO →
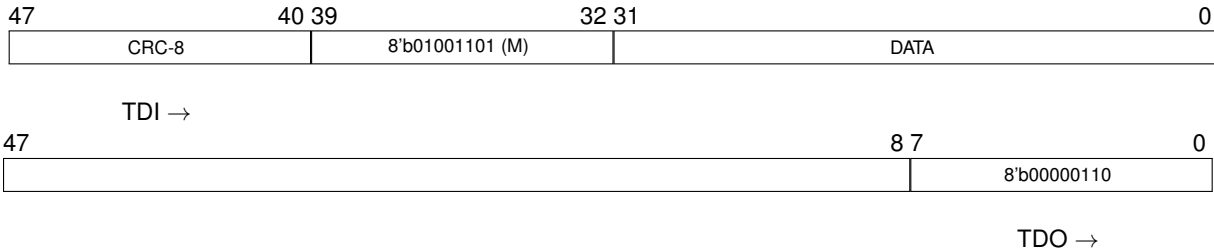
Figure 8.18: MBIST Run Command structure.

8.3.18. JTAG DEBUG LINK PINOUT

Table 8.6: 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	

9. REVISION HISTORY

Table 9.1: Revision history.

Doc. Rev.	Date	Comments
1.0	08-2022	First Issue.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

© 2023 ChipCraft Sp. z o.o.
All rights reserved

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

www.chipcraft-ic.com