Template Datasheet

CC100-C Processor Core Template

1.1

Scope

This document contains the CC100-C Processor Core Template Datasheet. Fixed default configuration options and features are described. Configuration registers of on-chip debugger and a fixed set of tightly-coupled peripherals are described.

Contents

1. Block Diagram

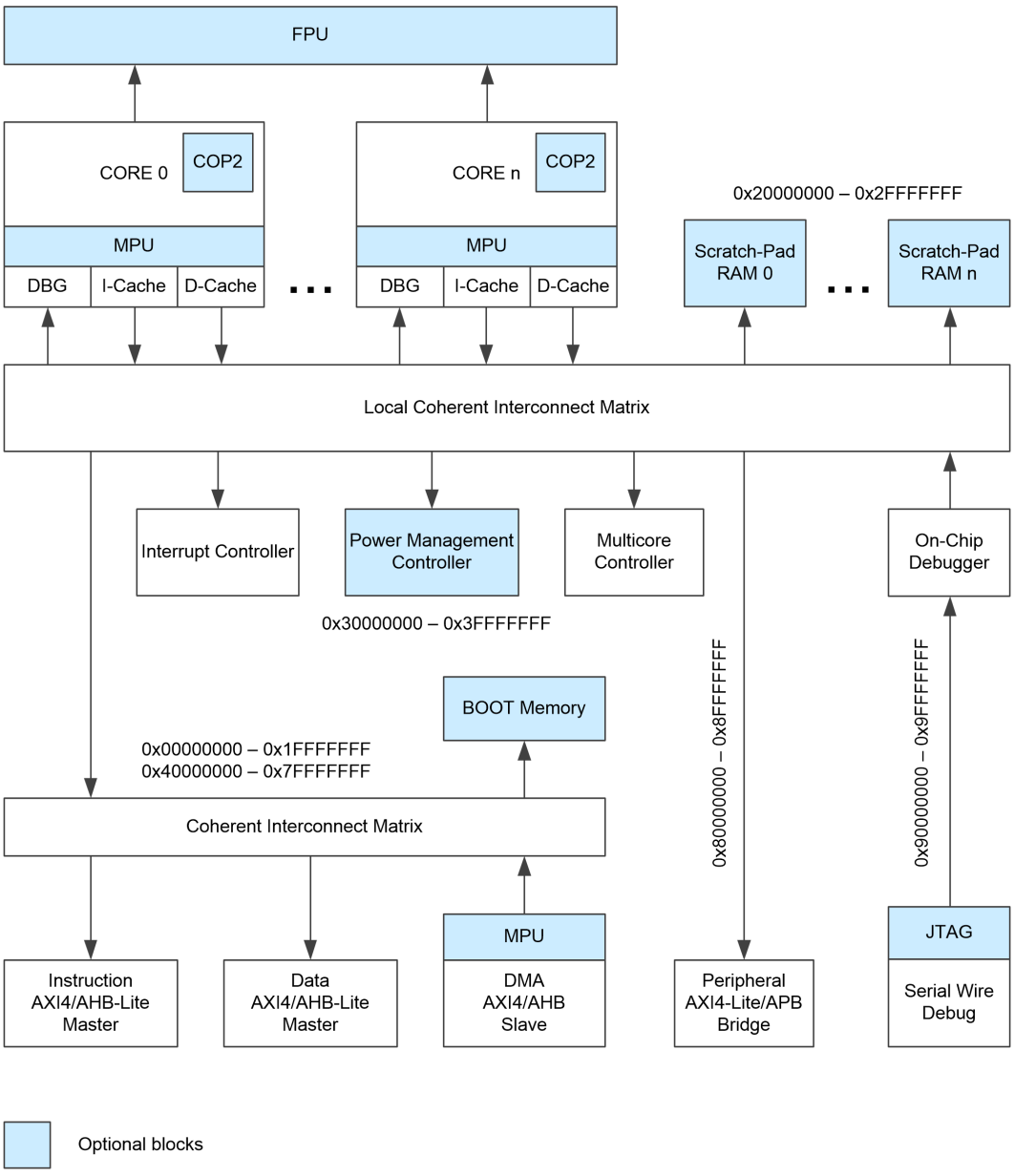
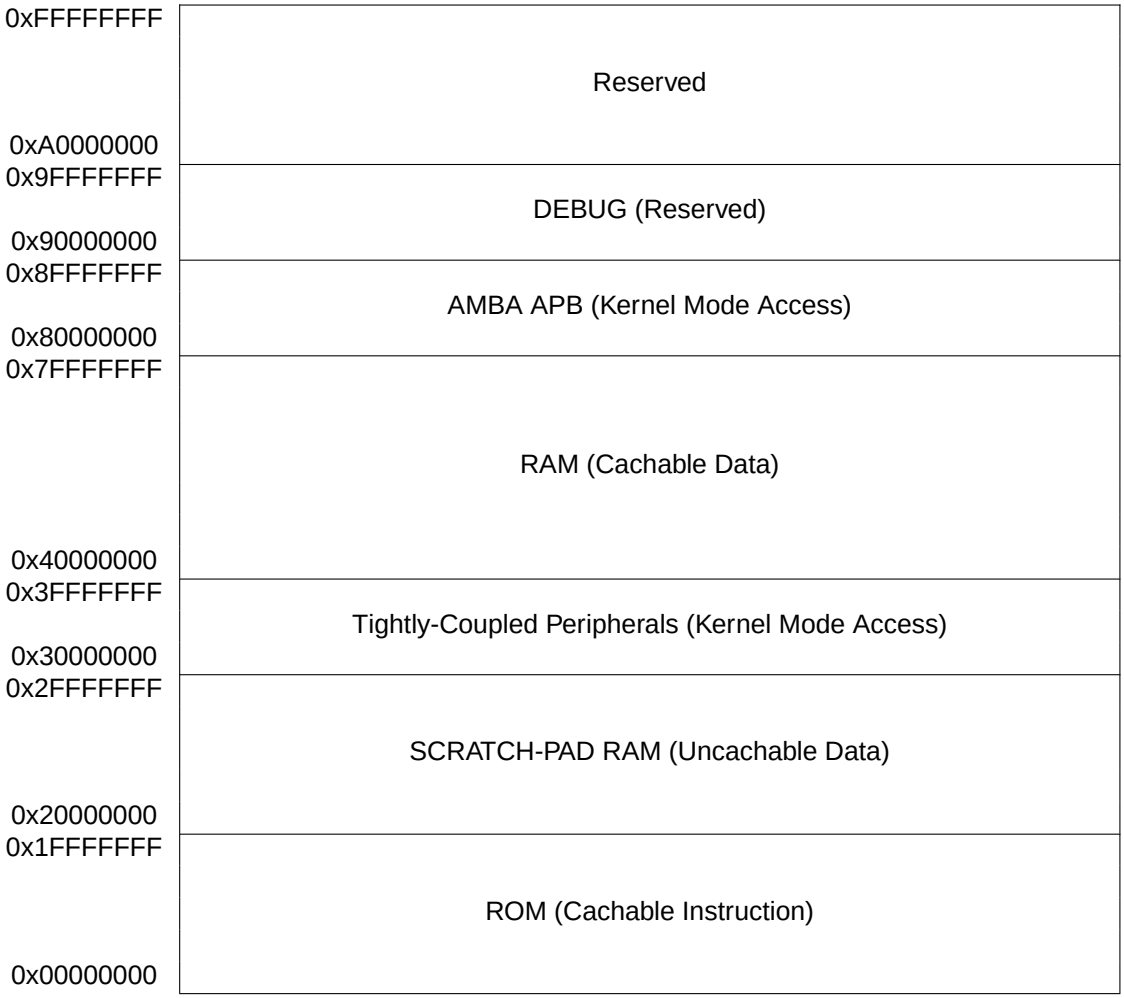


Figure 1.1. Processor block diagram.

2. Memory Map


0xFFFFFFFF
Reserved
0xA0000000
0x9FFFFFFF
DEBUG (Reserved)
0x90000000
0x8FFFFFFF
AMBA APB (Kernel Mode Access)
0x80000000
0x7FFFFFFF
RAM (Cachable Data)
0x40000000
0x3FFFFFFF
Tightly-Coupled Peripherals (Kernel Mode Access)
0x30000000
0x2FFFFFFF
SCRATCH-PAD RAM (Uncachable Data)
0x20000000
0x1FFFFFFF
ROM (Cachable Instruction)
0x00000000

2.1 Region Descriptions

CC100-C processor implements big-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.

2.1.1 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 *Unlock Register* in Interrupt Controller. 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.

2.1.2 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.

2.1.3 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 linked (LL) and store conditional (SC) atomic instructions. To increase system bandwidth, store operation that results in RAM bus error will silently fail without raising exception.

2.1.4 Tightly-Coupled Peripherals

The region is reserved for tightly-coupled processor peripherals such as Interrupt Controller or Power Management Controller. Each processor core has its own private bus connecting it with tightly-coupled peripherals. Depending on peripheral, particular registers can be shared for all cores or each core can have its exclusive version. 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.

2.1.5 AMBA APB Region

The region is dedicated to standard peripherals connected using AMBA APB bus. Only word-size access is allowed. Otherwise APB peripheral exception will be raised. Access to non-existing peripherals will generate APB peripheral exception. Attempt to perform a store operation on read-only address can silently fail or generate an exception depending on peripheral. Peripherals are allowed to fail access an generate an exception for their internal reason. The region is shared for all processor cores and is accessed using dedicated arbiter.

2.1.6 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 below.

Address	Description
0x90000000 - 0x9000000C	4 x Breakpoint
0x90000010 - 0x9000001C	4 x Watchpoint
0x90000020 - 0x90000020	Burst Counter Register
0x91000000 - 0x9100007C	Integer Register-File
0x91000080 - 0x910000FC	Floating-Point Register-File
0x92000000 - 0x920xxxxx	Instruction Cache Data
0x92100000 - 0x921xxxxx	Instruction Cache Tag
0x92200000 - 0x922xxxxx	Data Cache Data
0x92300000 - 0x923xxxxx	Data Cache Tag

3. Instruction Set

This chapter contains the brief introduction to the CC100-C instruction set architecture.

The instruction set is derived from the MIPS®-II [1] architecture. All differences are described in detail herein.

The CC100-C processing unit pipeline is fully interlocked and there are no architectural delayed loads. Programs will execute correctly when the loaded data is used by the instructions following the loads, but this may require extra clock cycles.

All branches have architectural delay of one instruction. When a branch is taken, the instruction immediately following the branch instruction, in the branch delay slot, is executed before the branch to the target instruction takes place.

For detailed informations about instruction set list and its description please refer to the **CC100-ISA** document.

[1] MIPS is registered trademark of Imagination Technologies LLC. ChipCraft Sp. z .o.o. is not associated with Imagination Technologies LLC in any way.

4. Tightly-Coupled Peripherals

4.1 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: 0x300A0014

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

4.2 Memory Map

The table below 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.

Address	Peripheral
0x30010000 - 0x30010FFF	Multicore Controller
0x30020000 - 0x30020FFF	Power Management Controller
0x30030000 - 0x300301FF	Interrupt Controller
0x30040000 - 0x30040FFF	GNSS Controller
0x30070000 - 0x30070FFF	Instruction Cache Controller
0x30072000 - 0x30072FFF	Data Cache Controller

4.3 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.

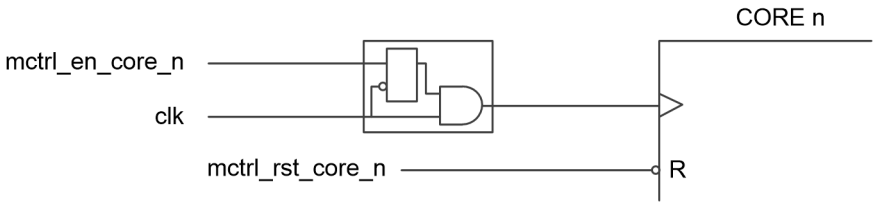


Figure 4.1. Simplified processor cores clock gating scheme.

4.3.1 Registers List

Address Offset	Register	Name
0x30010000	STATUS	Status Register
0x30010004	CORE_NUM	Core Number Register
0x30010008	CORE_SHDN	Shutdown Register
0x30010010	CORE_0_ADDR	Core 0 Start Address Register
0x30010014	CORE_1_ADDR	Core 1 Start Address Register
0x30010018	CORE_2_ADDR	Core 2 Start Address Register
. . .	. . .	. . .
0x300100D8	CORE_0_RUN	Core 0 Run Register
0x300100DC	CORE_1_RUN	Core 1 Run Register
0x300100E0	CORE_2_RUN	Core 2 Run Register
. . .	. . .	. . .

4.3.2 Status Register

Address: 0x00

Type: shared

31	30	···	···	···	2	1	0
STAT31	STAT30	···	···	···	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.

4.3.3 Core Number Register

Address: 0x04

Type: shared

31	0
CORE_NUM[31:0]	
R	
4	

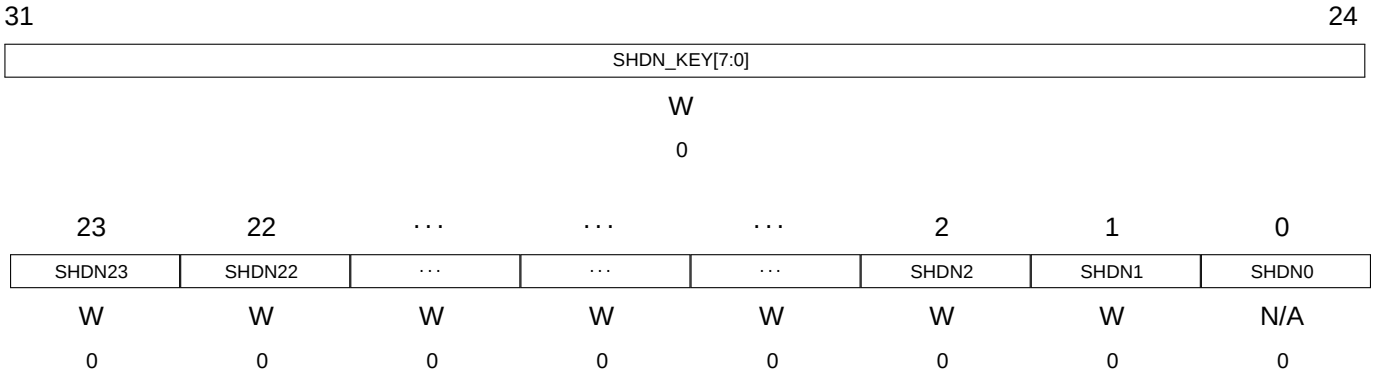
CORE_NUM[31:0] *Core Number*

Stores the number of processor cores.

4.3.4 Shutdown Register

Address: 0x08

Type: shared



SHDN*n* *Shut down Core n*

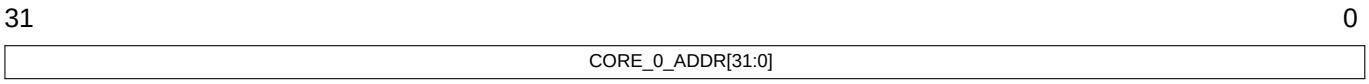
Writing immediately causes Core *n* to shut down. Core 0 cannot be shut down using this register. The operation will succeed only when attempting to shut down cores that are currently active and *SHDN_KEY* field is set properly. The operation will fail when bit corresponding to the Core 0 will be set. 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.

4.3.5 Start Address Registers

Address: 0x10

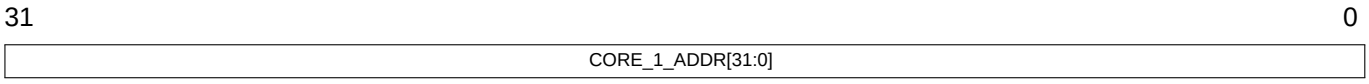


N/A

0

Address: 0x14

Type: shared

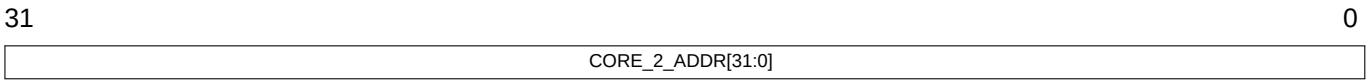


R/W

0

Address: 0x18

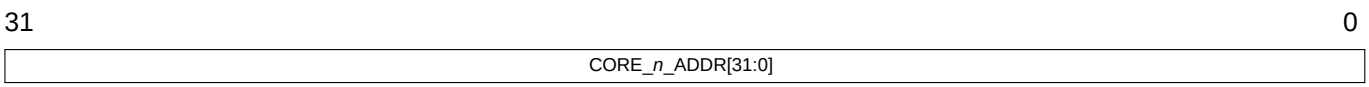
Type: shared



R/W

0

Address: . . .



R/W

0

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

Stores the Core *n* start address.

4.3.6 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.

4.4 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.

4.4.1 Registers List

Address Offset	Register	Name
0x30020004	RSTRSN	Reset Cause Register
0x30020008	PWDRST	Reset Register
0x3002000C	DPRST	Deep Reset Register
0x30020014	INFO	Info Register

4.4.2 Reset Cause Register

Address: 0x04

Type: shared

31	30	···	···	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.

4.4.3 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.

4.4.4 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.

4.4.5 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.

4.5 Interrupt Controller

Interrupt Controller is responsible for generating and handling interrupts (asynchronous events from peripherals) and exceptions (synchronous events caused by executing particular instruction). Figure 4.2 presents the simplified exception and interrupt generation datapath. Incoming interrupts are masked by *Interrupt Mask Register* and then registered. If *Status Register* field *(OIEN,PIEN,CIEN)* = 1 then the interrupt with the highest priority is forwarded to the processor and its number is stored in *IRQ_NUM* field of *Status Register*. If *CIEN* = 1 and field *(OIEN,PIEN,CIEN)* > 1 then the interrupt with priority higher then the one being executed is forwarded to the processor and its number is stored in *IRQ_NUM* field of *Status Register*. Registered interrupts are cleared during execution of *RFE* instruction. User have to clear interrupt flag in appropriate peripheral module till that moment, otherwise the interrupt can be registered again. Exceptions have virtually the highest priority, so they can always preempt any interrupt or exception. On the other hand, interrupt controller will serve interrupt instead of exception if they occure in the same clock cycle.

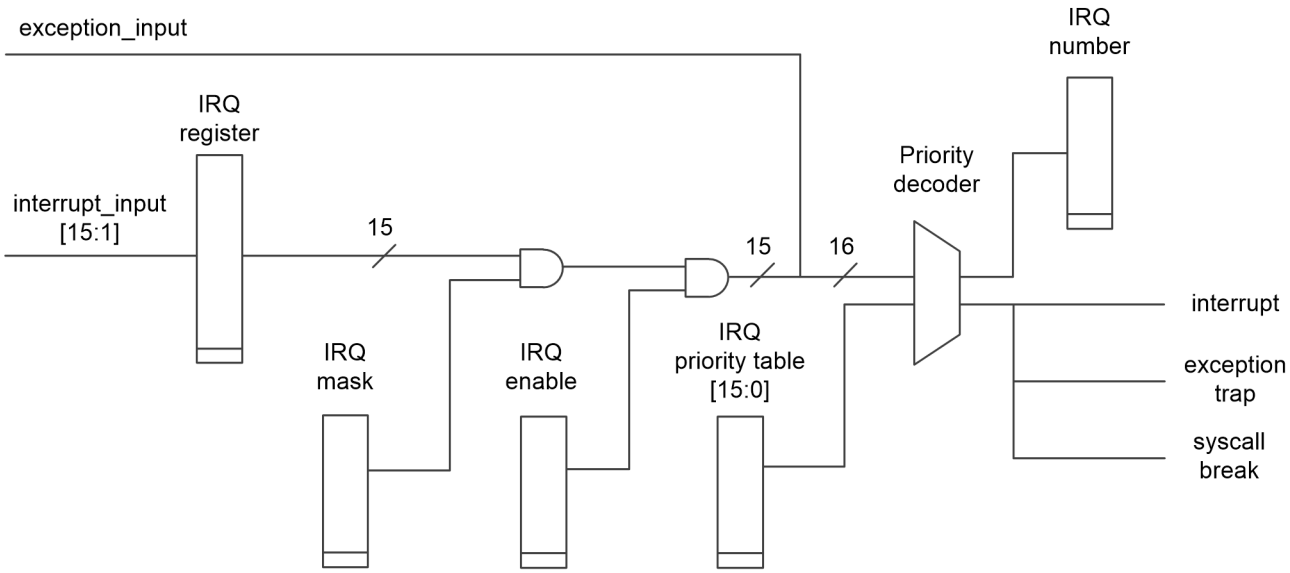


Figure 4.2. Simplified exception and interrupt generation datapath.

There are three types of events forwarded to the processor core and three corresponding exception vectors. Asynchronous events from peripherals (interrupts), synchronous events from the processor core(e.g. overflow, trap) and synchronous and excepted system events caused by SYSCALL, BREAK instruction. Exception vector table and Interrupt Controller registers are described in detail below.

Event	Vector
interrupt	0x00000028
syscall/break	0x00000030
exception/trap	0x00000038

4.5.1 Stack Protection Unit

Stack Protection Unit continuously monitors load and store operations with GPR[29] (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 4.3 presents the simplified stack exception generation datapath.

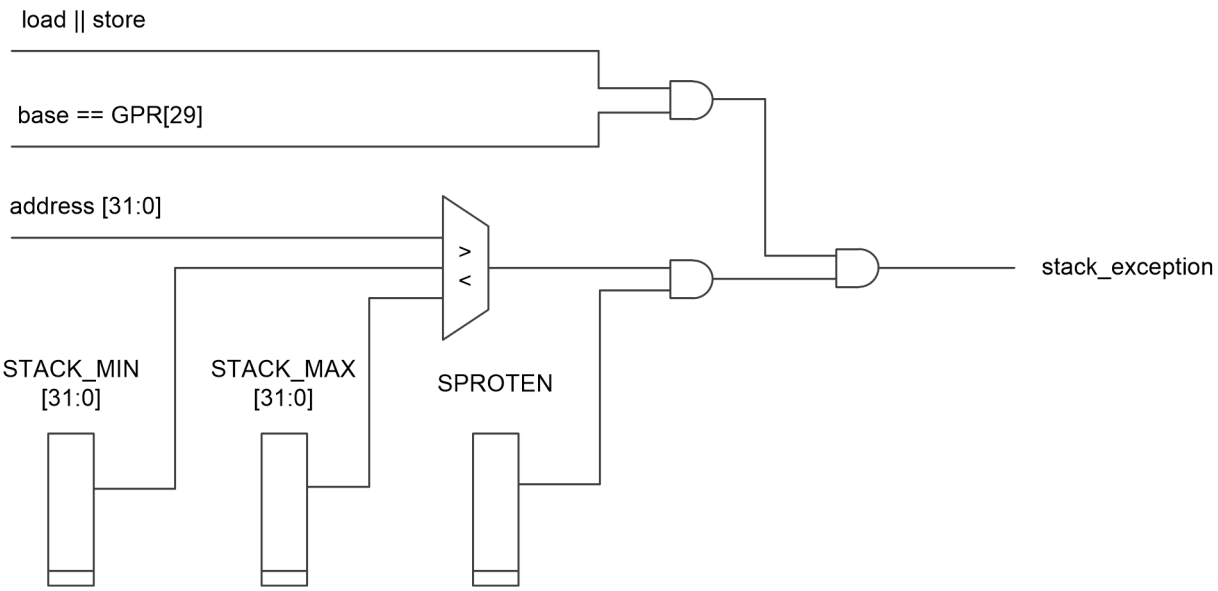


Figure 4.3. Simplified stack exception generation datapath.

4.5.2 Interrupt Nesting

Interrupt Controller allows interrupt preemption to serve higher priority events. During interrupt service routine the current interrupt priority is recovered from *Interrupt Priority* register pointed by *IRQ_NUM* field. If any pending interrupt has higher priority, the current interrupt routine will be preempted and *IRQ_NUM* along with *Interrupt History* register will be updated. The only exception is level zero (highest) priority. It is reserved for processor exceptions and can be preempted by other exception. After executing *RFE* instruction both *IRQ_NUM* and *Interrupt History* will be automatically recovered. *Interrupt History* register is able to cover all interrupt priorities assuming no dynamically changed interrupt priorities are used. Figure 4.4 presents the exemplary preemption flow assuming the interrupts with lower numbers has higher priority.

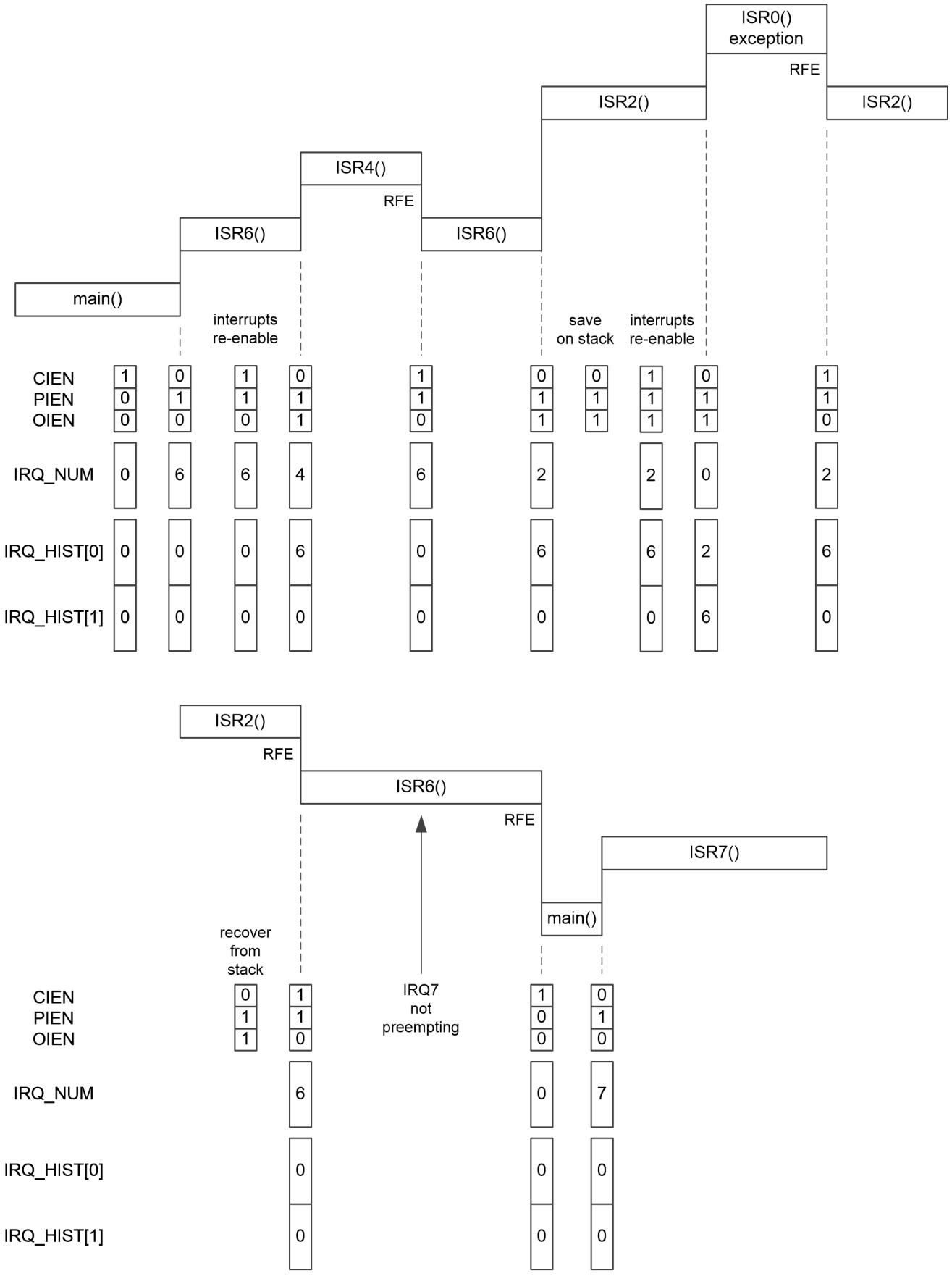


Figure 4.4. Interrupt preemption flow example.

4.5.3 Registers List

Address Offset	Register	Name
0x30030000	STATUS	Status Register
0x30030004	RET_PC	Return Address Register
0x30030008	EXCRSN	Exception Cause Register
0x3003000C	BAD_ADDR	Bad Address Register
0x30030010	LL_ADDR	Load Linked Address
0x30030014	ROM_UNLOCK	ROM Unlock Register
0x30030018	IRQ_HIST	Interrupt History Register
0x3003001C	CPU_INFO_0	Processor Resources Register 0
0x30030020	CPU_INFO_1	Processor Resources Register 1
0x3003002C	ICORE_IRQMAP	Intercore Interrupt Mapping Register
0x30030030	ICORE_IRQTRIG	Intercore Interrupt Trigger Register
0x30030034	ICORE_IRQF	Intercore Interrupt Flags Register
0x30030038	SP_MIN	Minimum Stack Pointer Value Register
0x3003003C	SP_MAX	Maximum Stack Pointer Value Register
0x30030040	IRQ_PRIOR_0	Interrupt Priority Register 0
0x30030044	IRQ_PRIOR_1	Interrupt Priority Register 1
0x30030048	IRQ_PRIOR_2	Interrupt Priority Register 2
. . .	. . .	. . .
0x300300C0	IRQ_MASK	Interrupt Mask Register
0x300300C4	CPU_TILE_ID	Processor Tile Index Register
0x300300C8	CPU_IDCODE	JTAG Format Processor IDCODE Register
0x300300CC	DBG_BAUD	On-Chip Debugger Baud Register

4.5.4 Status Register

Address: 0x00

Type: exclusive

31	30	29	28	27	26	25	24
CORE_ID[7:0]							
R							
CORE_ID							

23	22	21	20	19	18	17	16
				BD		IRQ_NUM[3:2]	
R	R	R	R	R	R	R/W	
0	0	0	0	0	0	0	

15	14	13	12	11	10	9	8
IRQ_NUM[1:0]		OMODE	PMODE	CMODE	OIEN	PIEN	CIEN
R/W		R/W	R/W	R/W	R/W	R/W	R/W
0		0	0	0	0	0	0

7	6	5	4	3	2	1	0
			IR	SPROTEN	LLBIT		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.

LLBIT *Load Linked Bit*

Bit is set if there is an active RWM sequence on the processor.

SPROTEN *Stack Protection Enable*

Bit is used to enable or disable hardware stack protection.

IR *Interrupt Request*

Bit is set if any peripheral requests asynchronous interrupt independently on Interrupt Controller settings. This bit can be set with no unmasked interrupts or interrupts disabled.

CIEN *Current Interrupt Enable*

Current Interrupt Enable bit.

PIEN *Previous Interrupt Enable*

Previous value of *CIEN* bit. Field (OIEN,PIEN,CIEN) is shifted left on exception event and right after execution of *RFE* instruction.

OIEN *Old Interrupt Enable*

Previous value of *PIEN* bit. Field (OIEN,PIEN,CIEN) is shifted left on exception event and right after execution of *RFE* instruction.

CMODE *Current Core Mode*

Bit is set if processor core is in User Mode.

PMODE *Current Core Mode*

Previous value of *CMODE* bit. Field (OMODE,PMODE,CMODE) is shifted left on exception event and right after execution of *RFE* instruction.

OMODE *Current Core Mode*

Previous value of *PMODE* bit. Field (OMODE,PMODE,CMODE) is shifted left on exception event and right after execution of *RFE* instruction.

IRQ_NUM[3:0] *IRQ Number*

Stores the number of currently pending interrupt.

BD *Branch Delay*

Bit is set if current exception took place in branch delay slot.

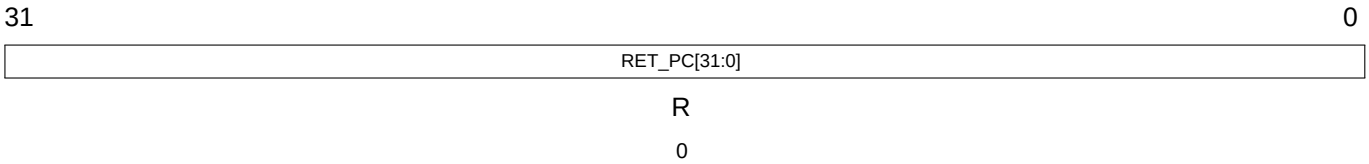
CORE_ID[7:0] *Core ID*

Stores index of processor core.

4.5.5 Return Address Register

Address: 0x04

Type: exclusive



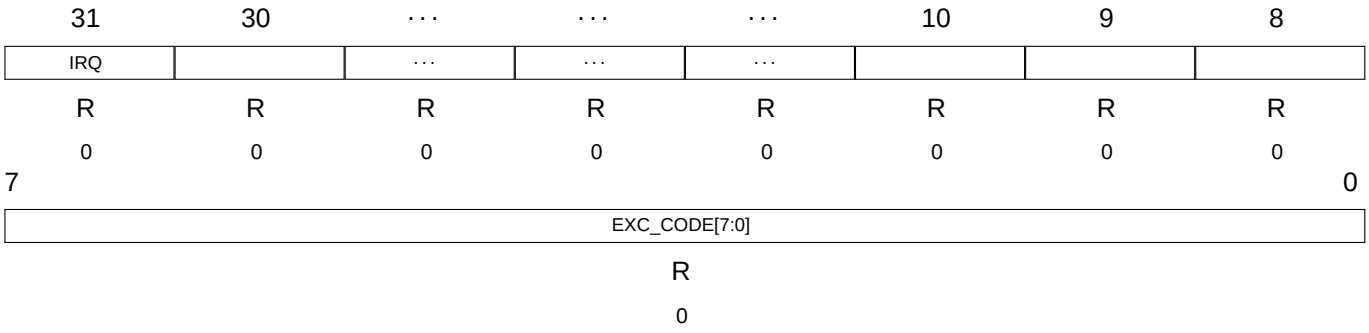
RET_PC[31:0] *Return Address*

Stores the address of instruction that caused exception.

4.5.6 Exception Cause Register

Address: 0x08

Type: exclusive



Exception Code	Exception Type
1	Overflow Exception
2	Load Exception
3	Store Exception
4	Syscall Exception
5	Break Exception
6	Unknown Exception
7	Address Exception
8	Pipeline Error Exception
9	Stack Protection Exception
10	Privileged Data Exception
11	Privileged Instruction Exception
12	Trap Exception
13	Tightly-coupled Peripheral Exception
14	AMBA Peripheral Exception
15	Instruction Bus Exception
16	Data Bus Exception

Overflow Exception

Bit is set on arithmetic overflow exception.

Load Exception

Bit is set if load address is not naturally aligned or is pointing to reserved address space.

Store Exception

Bit is set if store address is not naturally aligned or is pointing to reserved address space.

Syscall Exception

Bit is set on executing *SYSCALL* instruction.

Break Exception

Bit is set on executing *BREAK* instruction.

Unknown Exception

Bit is set on executing unsupported instruction.

Address Exception

Bit is set if jump address is not naturally aligned.

Pipeline Error Exception

Bit is set if an error occurred in the processor core pipeline.

Stack Exception

Bit is set if stack protection exception occurred.

Privileged Data Exception

Bit is set if load or store instruction address is pointing to the privileged data region. This exception can occur while accessing kernel reserved areas in user mode.

Privileged Instruction Exception

Bit is set on executing privileged instruction in user mode.

Trap Exception

Bit is set on exception caused by executing *TRAP* instruction.

Peripheral Exception

Bit is set on attempt to perform not word-size access to tightly-coupled peripherals region or on attempt to perform access outside implemented scratch-pad ram memory range.

APB Peripheral Exception

Bit is set on attempt to access non-existing APB peripheral. Inappropriate access size or peripheral internal reason can also cause this exception.

Instruction Bus Exception

Bit is set if instruction bus error occurs. For example on attempt to perform access outside implemented instruction memory range.

Data Bus Exception

Bit is set if data bus error occurs. For example on attempt to perform access outside implemented data memory

range. To increase system bandwidth, only read exceptions are signalized. When error occurs on data bus during writing, the operation will silently fail.

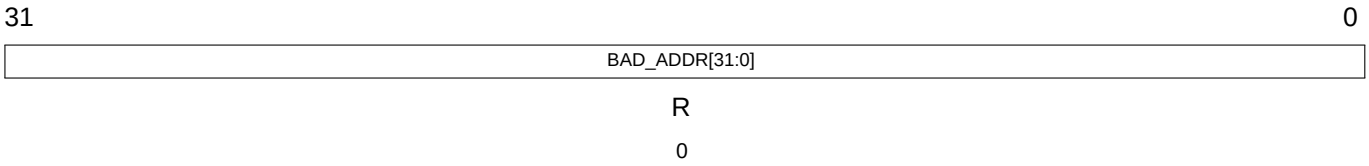
IRQ *Interrupt Exception*

Bit is set if exception is caused by external interrupt.

4.5.7 Bad Address Register

Address: 0x0C

Type: exclusive



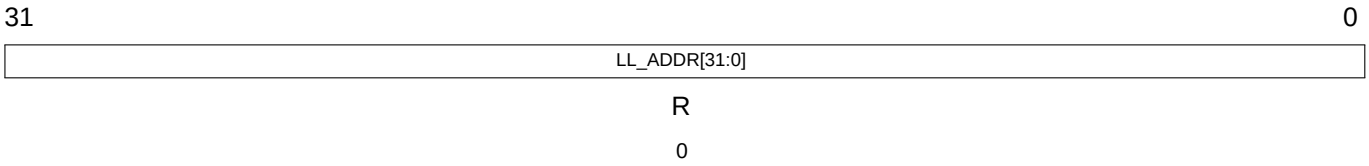
BAD_ADDR[31:0] *Bad Address*

The register stores the violating address in case of load, store or address exception. Otherwise register stores the address of interrupting instruction itself.

4.5.8 Load Linked Address Register

Address: 0x10

Type: exclusive



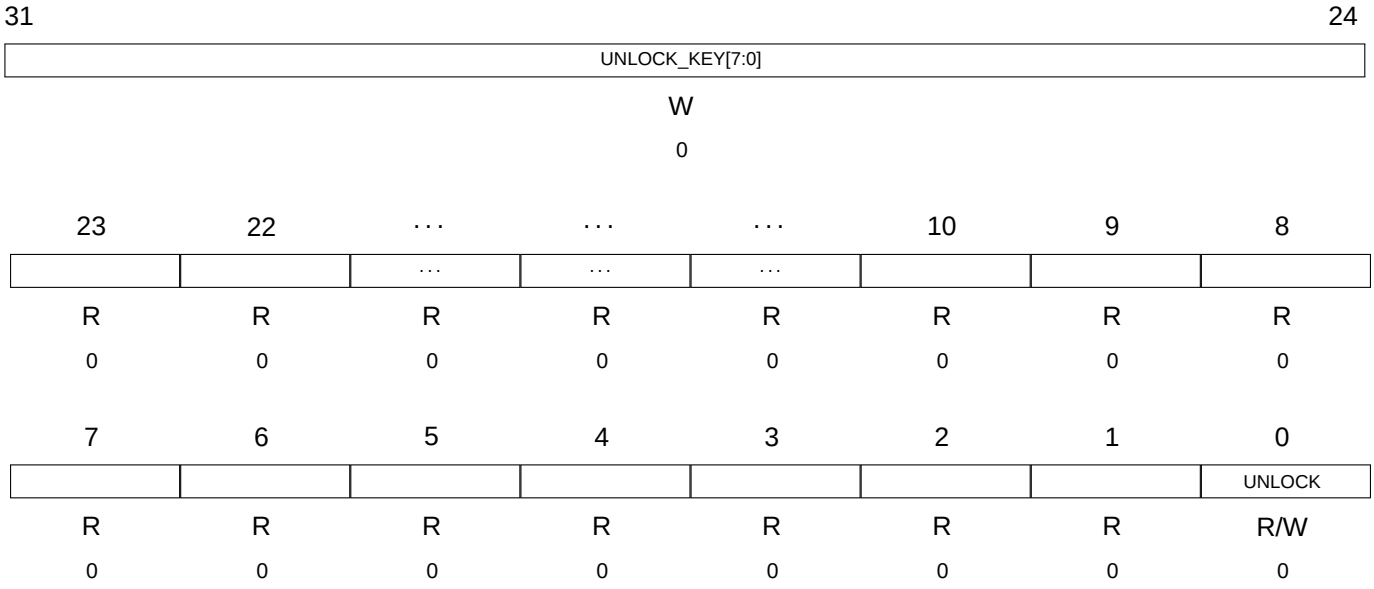
LL_ADDR[31:0] *Load Linked Address*

The address of the last procesor RMW sequence.

4.5.9 ROM Unlock Register

Address: 0x14

Type: shared



UNLOCK *ROM Unlock*

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

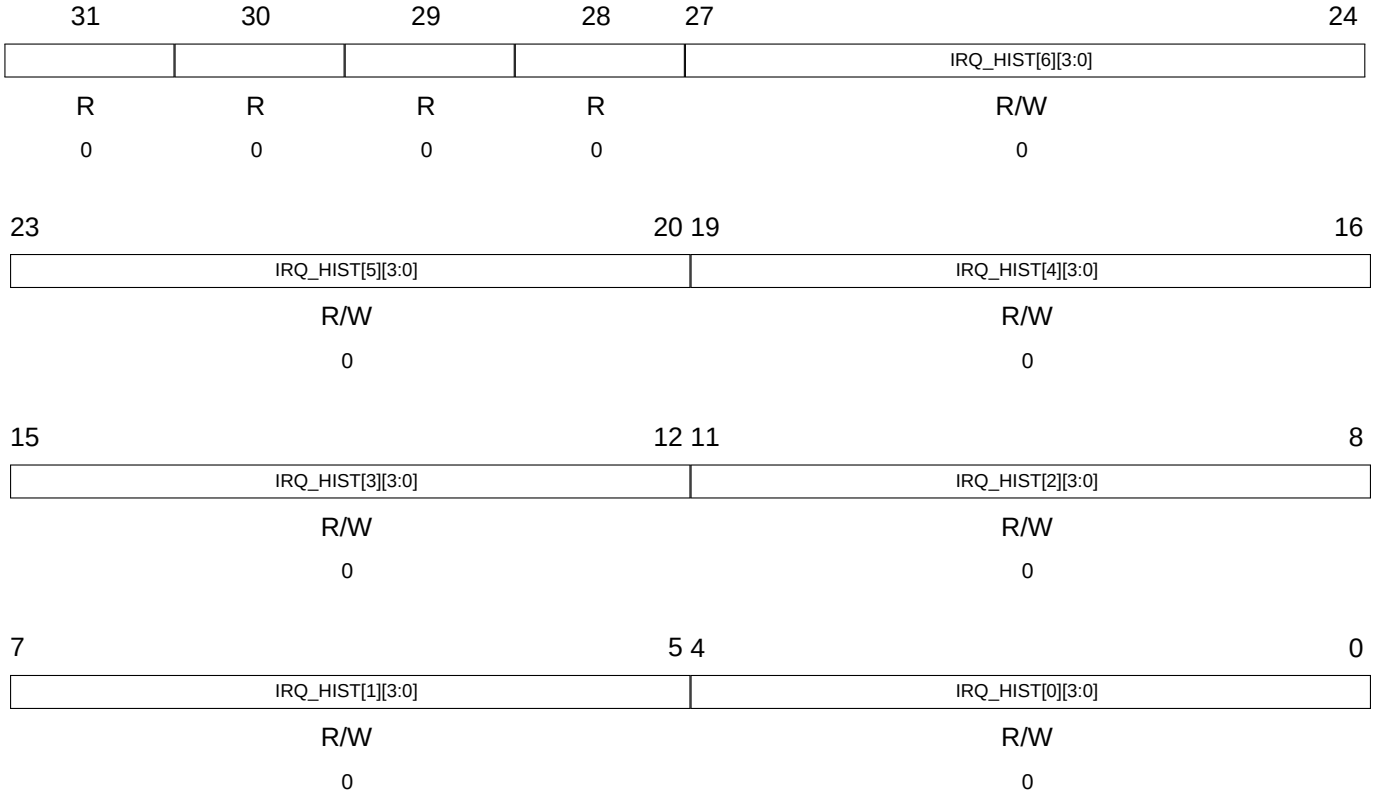
UNLOCK_KEY[7:0] *Unlock Key*

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

4.5.10 Interrupt History Register

Address: 0x18

Type: exclusive



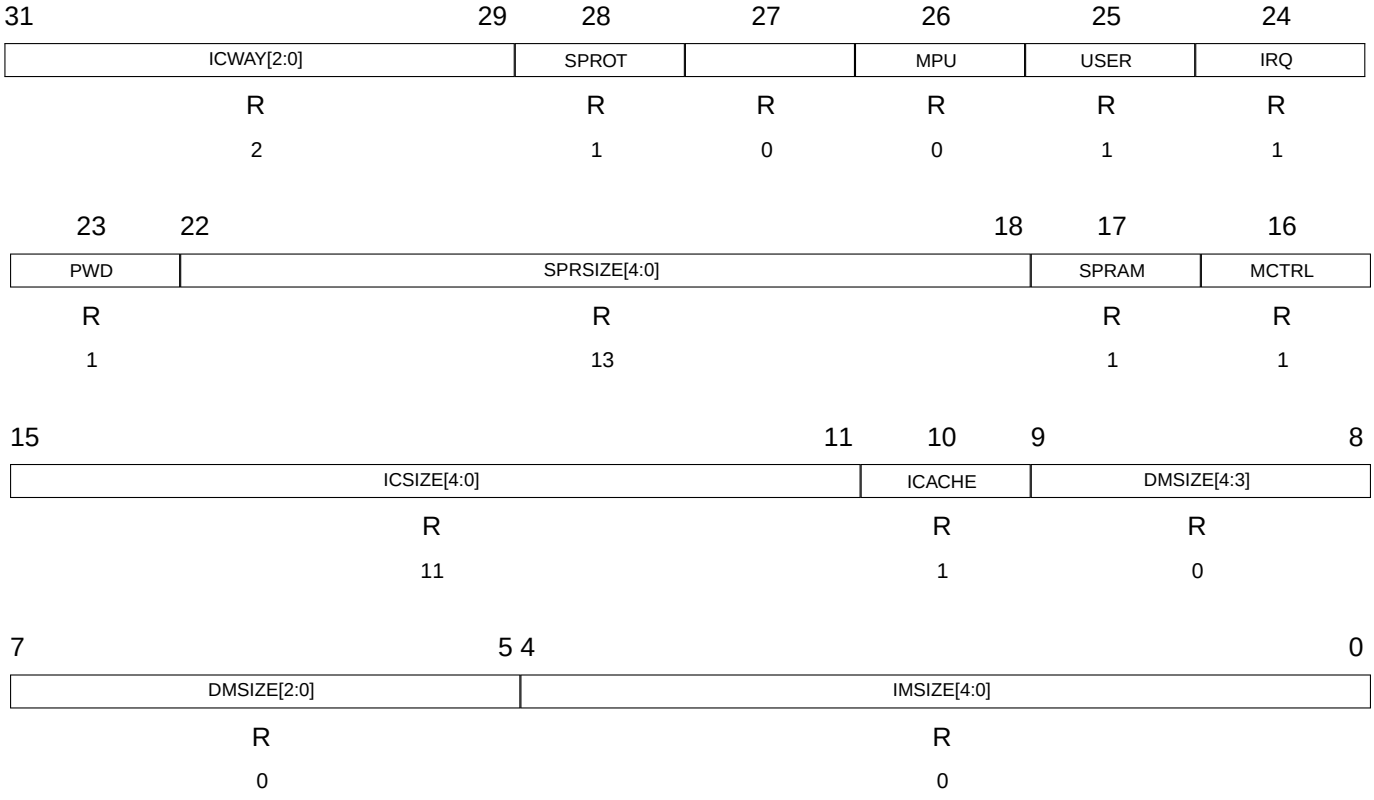
IRQ_HIST[*n*][3:0] *Interrupt History n*

This register stores interrupt nesting history. Eight levels of nested interrupts are recorded (including *Status Register*) minimizing the need to store register content on the stack. When interrupt priorities are not dynamically changed on runtime, the history register covers preemption across all interrupt priorities. Bit fields *IRQ_HIST[6:0]* are shifted left and *IRQ_NUM* is stored in *IRQ_HIST[0]* on exception. Bit fields *IRQ_HIST[6:0]* are shifted right and zero is stored in *IRQ_NUM[6]* while executing *RFE* instruction.

4.5.11 Processor Resources Register 0

Address: 0x1C

Type: exclusive



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

Amount of on-chip instruction memory - $2^{IMSIZE}b$.

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

Amount of on-chip data memory - $2^{DMSIZE}b$.

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.

IRQ *Interrupt Controller*

Bit is set if processor has Interrupt Controller.

USER *User Mode*

Bit is set if processor supports user mode.

MPU *Memory Protection Unit*

Bit is set if processor has Memory Protection Unit.

SPROT *Stack Protection*

Bit is set if processor has stack protection hardware.

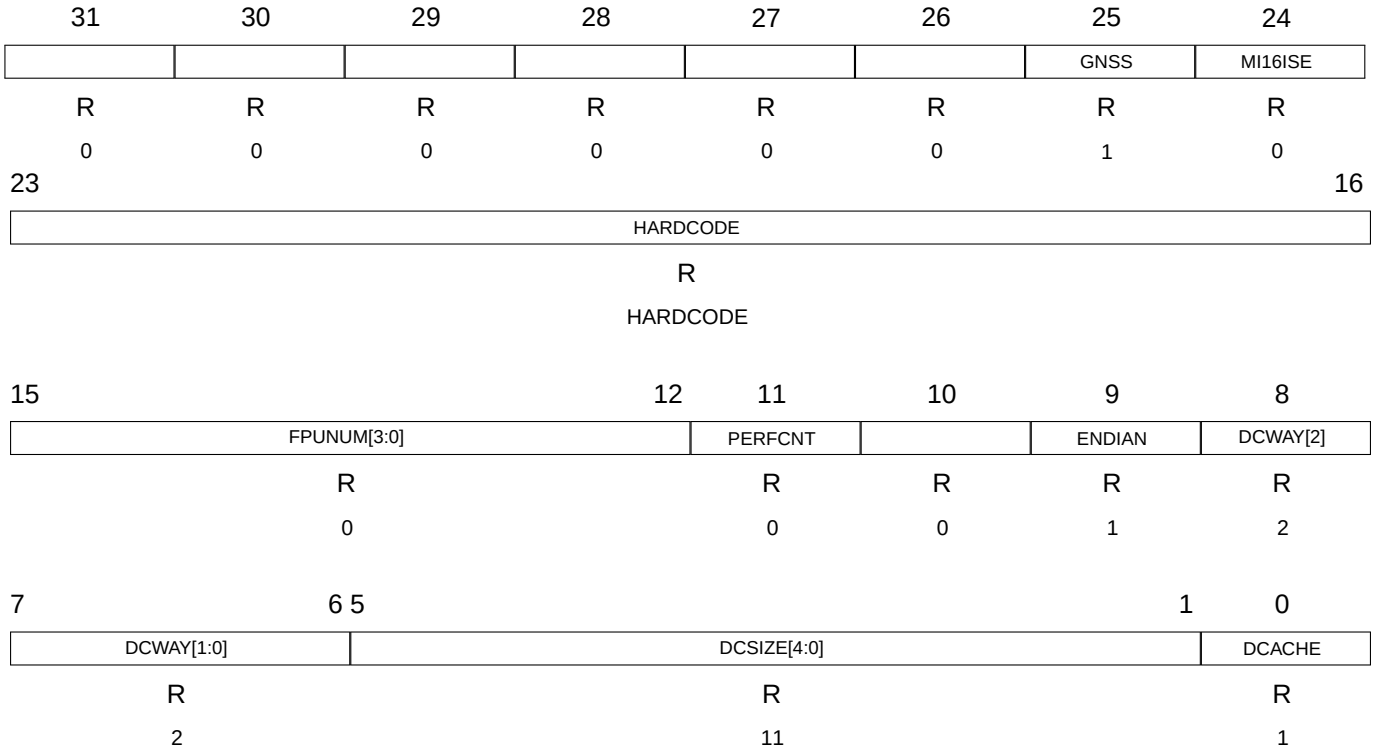
ICWAY[2:0] *Instruction Cache Ways*

Number of instruction cache ways.

4.5.12 Processor Resources Register 1

Address: 0x20

Type: exclusive



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.

PERFCNT *Performance Counter*

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

FPUNUM[3:0] *Number of FPUs*

Number of Floating Point Units.

HARDCODE[7:0] *Hardware Code*

Hardcoded hardware version.

MI16ISE *Compressed ISE*

Compressed 16-bit instruction set extension.

GNSS *GNSS Controller*

Bit is set if processor has GNSS Controller.

4.5.13 Intercore Interrupt Mapping Register

Address: 0x2C

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* *Intercore Interrupt Mapping n*

Stores the number of interrupt that will be recorded on intercore interrupt event.

4.5.14 Intercore Interrupt Trigger Register

Address: 0x30

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 Intercore Interrupt Trigger*

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

4.5.15 Intercore Interrupt Flags Register

Address: 0x34

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 intercore interrupt on this core. Only necessary bits are implemented.

4.5.16 Minimum Stack Pointer Value Register

Address: 0x38

Type: exclusive

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

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.

4.5.17 Maximum Stack Pointer Value Register

Address: 0x3C

Type: exclusive

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

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.

4.5.18 Interrupt Priority Registers

Address: 0x40

Type: shared

31	...	...	...	3	2	0
	...	...	...		IRQ_PRIOR0[2:0]	
R	R	R	R	R	R	
0	0	0	0	0	0	

Address: 0x44

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_PRIOR[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 interrupts. Allowed values starts from 1 (highest priority) to 7 (lowest priority). Value 0 is reserved for exceptions which have the highest priority. IRQ_PRIOR0 stores the priority of exceptions and is read-only.

4.5.19 Interrupt Mask Register

Address: 0xC0

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
IRQM15	IRQM14	IRQM13	IRQM12	IRQM11	IRQM10	IRQM9	IRQM8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

7	6	5	4	3	2	1	0
IRQM7	IRQM6	IRQM5	IRQM4	IRQM3	IRQM2	IRQM1	
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R
0	0	0	0	0	0	0	0

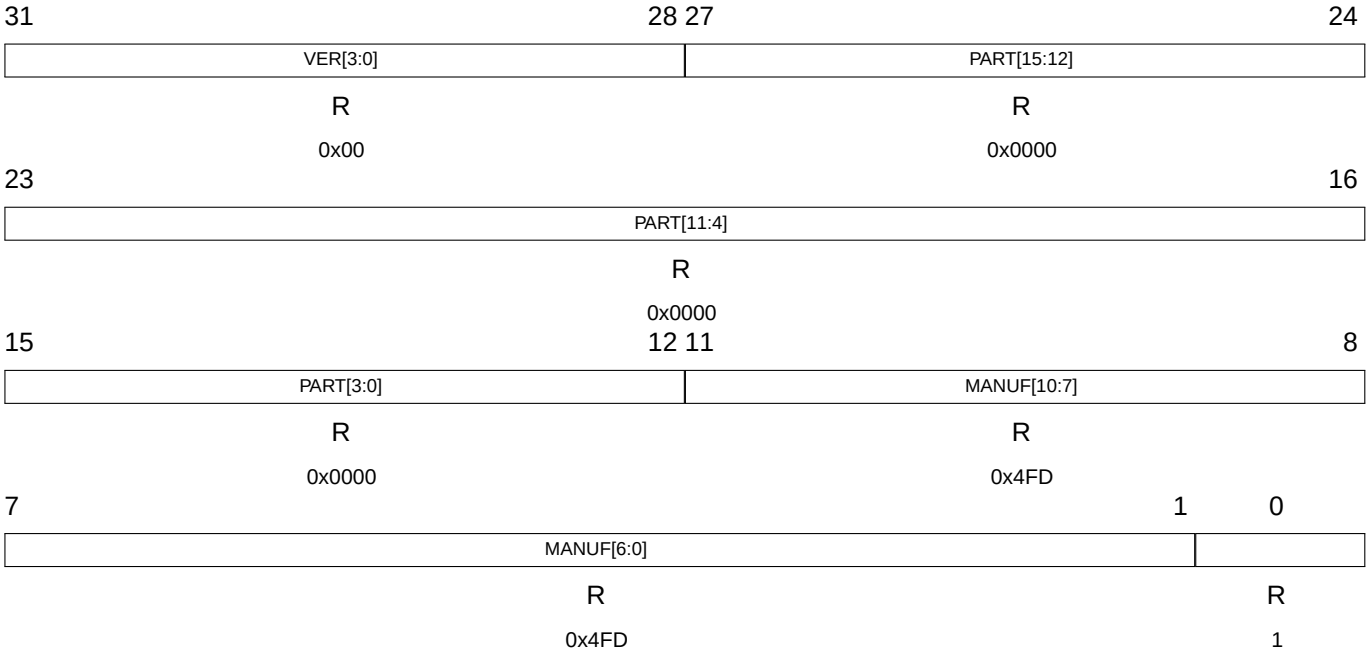
IRQM*n* *Interrupt n Mask*

Setting bit IRQM*n* enabled corresponding interrupt.

4.5.20 Processor IDCODE Register

Address: 0xC8

Type: shared



MANUF[10:0] *Manufacturer ID*

JTAG format ID code Manufacturer ID

PART[15:0] *Part Number*

JTAG format ID code Part Number

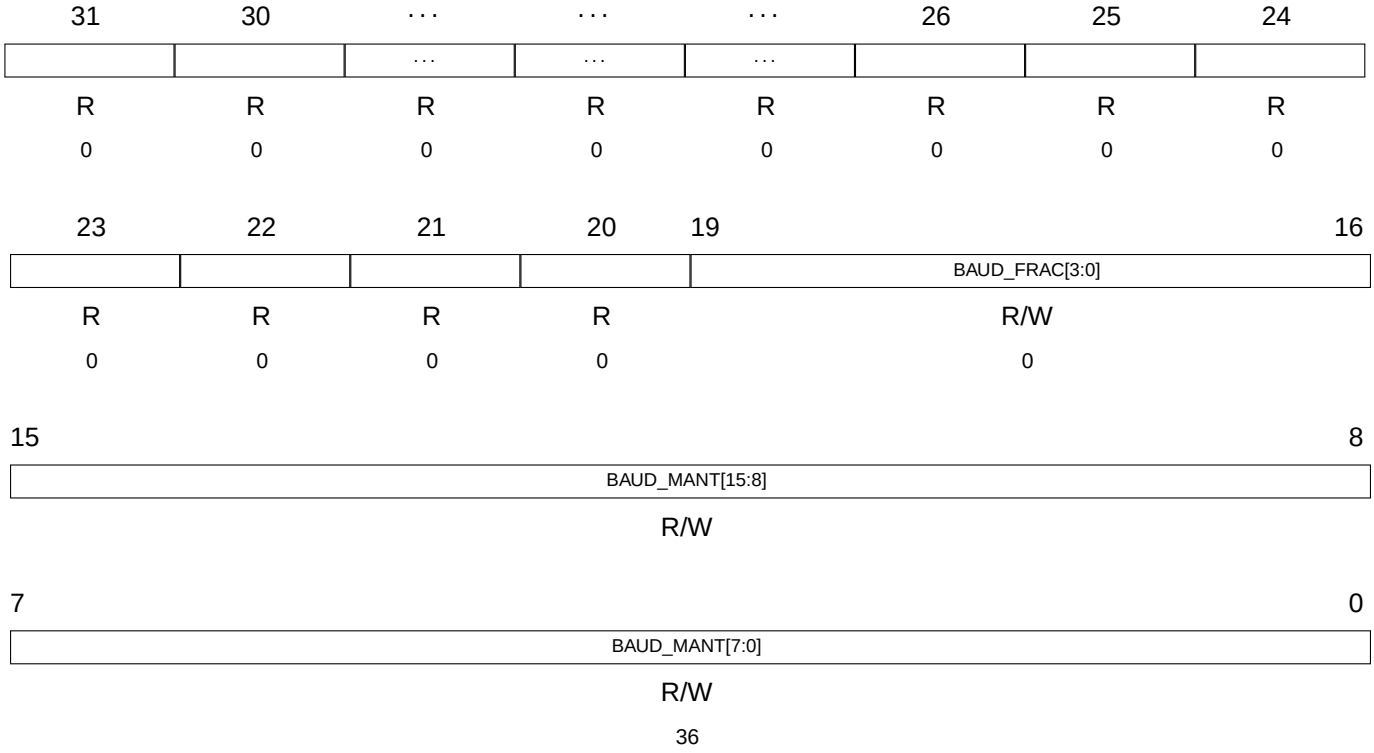
VER[3:0] *Version*

JTAG format ID code Version

4.5.21 On-Chip Debugger Baud Register

Address: 0x24

Type: shared



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

4.6 GNSS Controller

GNSS Controller module is used to aquire data samples from the GNSS analog frontends. The core can autonomously perform acquisition of a configured amount of data and store it in shared memory. For diagnostic purposes, the internal acquisition data fifos are mapped to the GNSS controller address space. They can be accessed only by the main processor core. The GNSS Controller is also used to support operation of the GNSS-ISE™ embedded in the processor core.

4.6.1 Registers List

Address Offset	Register	Name
0x30040000	STATUS	Status Register
0x30040004	COUNT	Count Register
0x30040008	ADDRI	Real Address Register
0x3004000C	ADDRQ	Imaginary Address Register
0x30040010	COUNTI	Remaining Real Count Register
0x30040014	COUNTQ	Remaining Imaginary Count Register
0x30040018	CTRL	GNSS-ISE Control Register
0x3004001C	TICKF	GNSS-ISE Code Tick Interrupt Flags Register
0x30040020	OVRF	GNSS-ISE Code Overrun Interrupt Flags Register
0x30040024	ERRF	GNSS-ISE Code Error Interrupt Flags Register
0x30040028	IRQMAP	Interrupt Mapping Register
0x3004002C	FIFOCNT	FIFO Count Override Register
0x30040030	FIFOWR	FIFO Write Override Register
0x30040034	TMSTMP_LO	Timestamp Low Register
0x30040038	TMSTMP_HI	Timestamp High Register
0x3004003C	ADCVAL	ADC Value Register
0x30046000-0x300463FF		Internal Real FIFO Memory Address Space
0x30046400-0x300467FF		Internal Imaginary FIFO Memory Address Space
0x30048000-0x30048FFF		GNSS-ISE Code Memory Address Space

4.6.2 Status Register

Address: 0x00

Type: shared

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**	**21**	**20**	**19**	**18**	**17**	**16**
		ACQIF	ACQIE		OVFQ	OVFI	ERR
R	R	R/W	R/W	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
		START	BUSY		MODE	RFAFE[1:0]	
R	R	W	R/W	R	R/W	R/W	
0	0	0	0	0	0	0	
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	L2EN	L5EN	L1EN		L2	L5	L1
R	R/W	R/W	R/W	R	R	R	R
0	0	0	0	0	0	0	0

L1 *L1 Band Available*

Bit is set if processor supports L1 band.

L5 *L5 Band Available*

Bit is set if processor supports L5 band.

L2 *L2 Band Available*

Bit is set if processor supports L2 band.

L1EN *L1 Band Enable*

Set this bit to enable L1 ADC data engine.

L5EN *L5 Band Enable*

Set this bit to enable L5 ADC data engine.

L2EN *L2 Band Enable*

Set this bit to enable L2 ADC data engine.

RFAFE[1:0] *RF Analog Frontend Selection*

Selects Analog Frontend for data acquisition:

RFAFE[1:0]	Analog Frontend
00	L1 Frontend
01	L5 Frontend
10	L2 Frontend
11	None

MODE *Acquisition Mode*

Selects acquisition mode:

0 Samples are stored as signed 4-bit data.

1 Samples are stored as signed 8-bit data.

BUSY *Controller Busy*

Bit is set if data acquisition is in progress.

START *Start Acquisition*

Write this bit to start data acquisition.

ERR *Acquisition Error*

Bit is set if controller encountered problem with access to the main memory during data acquisition. Gathered data can be corrupted.

OVFI *I Channel Overflow*

Bit is set if internal FIFO overflow occured on I channel. Gathered data can be corrupted.

OVFQ *Q Channel Overflow*

Bit is set if internal FIFO overflow occured on Q channel. Gathered data can be corrupted.

ACQIE *Acquisition Interrupt Enable*

Enable interrupt on data acquisition finish.

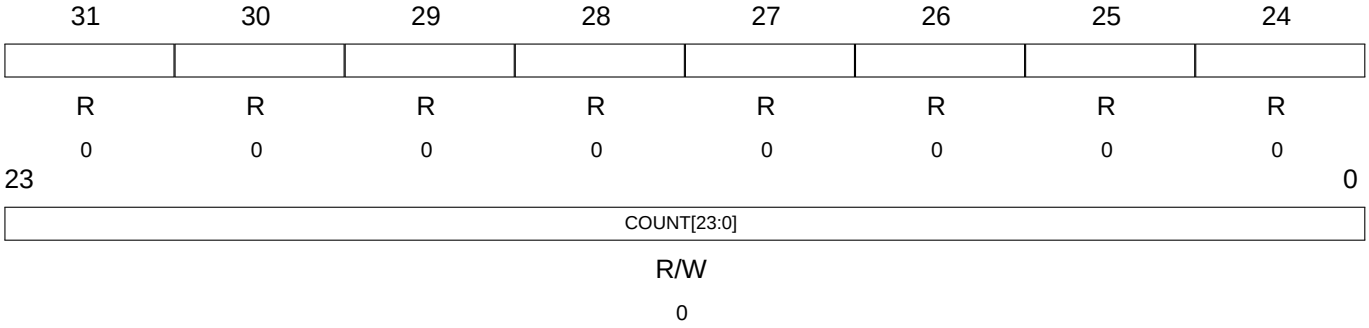
ACQIE *Acquisition Interrupt Flag*

Bit is set module finished data acquisition. Bit has to be cleared manually.

4.6.3 Count Register

Address: 0x04

Type: shared



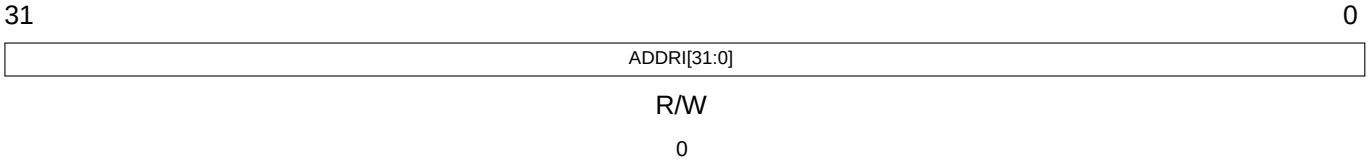
COUNT[23:0] *Count Value*

The number of data pacekts to store during acquisition. The packet is a 32-bit word and containt eigth 4-bit samples or four 8-bit samples depending on MODE bit configuration.

4.6.4 Real Address Register

Address: 0x08

Type: shared



ADDRI[31:0] *Real Address*

The address to store real channel data samples array.

4.6.5 Imaginary Address Register

Address: 0x0C

Type: shared

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

ADDRQ[31:0] *Imaginary Address*

The address to store imaginary channel data samples array.

4.6.6 Remaining Real Count Register

Address: 0x10

Type: shared

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	0
COUNTI[23:0]	
R/W	
0	

COUNTI[23:0] *Remaining Real Count*

The remaining number of data packets to store of real channel.

4.6.7 Remaining Imaginary Count Register

Address: 0x14

Type: shared

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															0
COUNTQ[23:0]															
R/W															
0															

COUNTQ[23:0] *Remaining Imaginary Count*

The remaining number of data packets to store of imaginary channel.

4.6.8 GNSS-ISE Control Register Register

Address: 0x18

Type: exclusive

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
					ERRIE	OVRIE	TICKIE
R	R	R	R	R	R/W	R/W	R/W
0	0	0	0	0	0	0	0

TICKIE *GNSS-ISE Code Tick Interrupt Enable*

Enable interrupt on GNSS-ISE tracking channel primary code tick.

OVRIE *GNSS-ISE Code Overrun Interrupt Enable*

Enable interrupt on GNSS-ISE tracking channel code tick overrun. Interrupt is indicated when new tracking channel code tick is received while the previous tick flag is still set.

ERRIE *GNSS-ISE Code Error Interrupt Enable*

Enable interrupt on GNSS-ISE tracking channel code generator error. Interrupt is issued when from internal reason the code generator was not able to properly generate new code sample.

4.6.9 GNSS-ISE Code Tick Interrupt Flags Register

Address: 0x1C

Type: exclusive

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
TICKF15	TICKF14	TICKF13	TICKF12	TICKF11	TICKF10	TICKF9	TICKF8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
TICKF7	TICKF6	TICKF5	TICKF4	TICKF3	TICKF2	TICKF1	TICKF0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

TICKFn *GNSS-ISE Code Tick Interrupt Flag of Channel n*

Bit is set if GNSS-ISE tracking channel *n* primary code tick was received. Every bit in the register can be cleared after writing one to the corresponding position.

4.6.10 GNSS-ISE Code Overrun Interrupt Flags Register

Address: 0x20

Type: exclusive

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
OVRF15	OVRF14	OVRF13	OVRF12	OVRF11	OVRF10	OVRF9	OVRF8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
OVRF7	OVRF6	OVRF5	OVRF4	OVRF3	OVRF2	OVRF1	OVRF0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OVRFn *GNSS-ISE Code Overrun Interrupt Flag of Channel n*

Bit is set if GNSS-ISE tracking channel *n* new code tick was received while the previous tick flag was still set. Every bit in the register can be cleared after writing one to the corresponding position.

4.6.11 GNSS-ISE Code Error Interrupt Flags Register

Address: 0x24

Type: exclusive

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
ERRF15	ERRF14	ERRF13	ERRF12	ERRF11	ERRF10	ERRF9	ERRF8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
ERRF7	ERRF6	ERRF5	ERRF4	ERRF3	ERRF2	ERRF1	ERRF0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

ERRFn *GNSS-ISE Code Overrun Interrupt Flag of Channel n*

Bit is set if from the internal reason GNSS-ISE tracking channel *n* new code sample was not generated properly. Every bit in the register can be cleared after writing one to the corresponding position.

4.6.12 Interrupt Mapping Register

Address: 0x28

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
0	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	1	0	0	0	0

IRQ*n* *Interrupt Mapping n*

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

4.6.13 FIFO Count Override Register

Address: 0x2C

Type: shared

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7							0
COUNT[7:0]							
W							
0							

COUNT[7:0] *FIFO Count Override Value*

For diagnostics purpose, to override internal FIFO count register.

4.6.14 FIFO Write Override Register

Address: 0x30

Type: shared

31	0
DATA[31:0]	
W	
0	

DATA[31:0] *FIFO Data Override Value*

To performe diagnostic write to the internal FIFO.

4.6.15 Cycle Low Register

Address: 0x34

Type: shared

31	0
TMSTMP_LO[31:0]	
R	
0	

TMSTMP_LO[31:0] *Timestamp Low Data*

Register holds the low part of stored performance counter value. Performance counter snapshot is taken on the beginning of every data acquisition process.

4.6.16 Cycle High Register

Address: 0x38

Type: exclusive

31	0
TMSTMP_HI[31:0]	
R	
0	

TMSTMP_HI[31:0] *Timestamp High Data*

Register holds the high part of stored performance counter value. Performance counter snapshot is taken on the beginning of every data acquisition process.

4.6.17 ADC Value Register

Address: 0x3C

Type: exclusive

31	0
ADVCAL[31:0]	
R	
0	

ADCVAL[31:0] *ADC Data*

For diganostic purposes. To read current ADC value.

4.7 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 4.5 and 4.6 presents the cache address mapping examples.

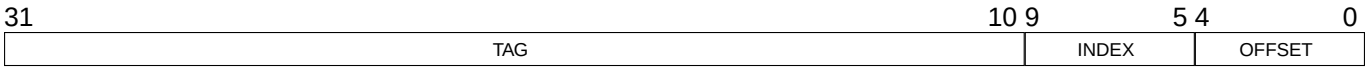


Figure 4.5. 1 KiB/way, 32 bytes/line address mapping example.

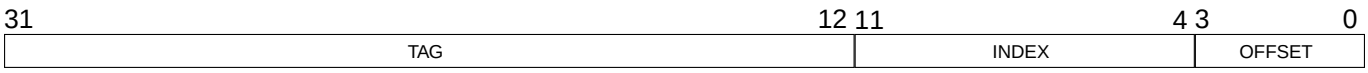


Figure 4.6. 4 KiB/way, 16 bytes/line address mapping example.

Figures 4.7 and 4.8 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.

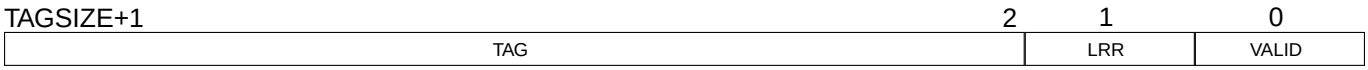


Figure 4.7. Tag memory entry with LRR bit.

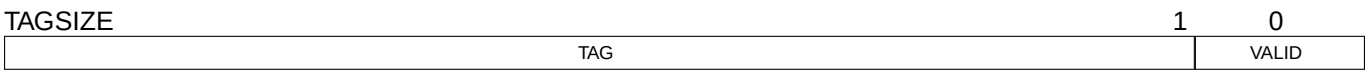


Figure 4.8. Tag memory entry without LRR bit.

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.

4.7.1 Registers List

Address Offset	Register	Name
0x30070000	STATUS	Status Register
0x30070004	FLUSH	Flush Register
0x30070008	INFO	Info Register

4.7.2 Status Register

Address: 0x00

Type: exclusive

31	30	...	...	...	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.

4.7.3 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.

4.7.4 Info Register

Address: 0x08

Type: shared

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	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.

4.8 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 4.9 and 4.10 presents the cache address mapping examples.

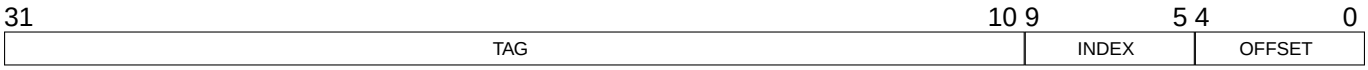


Figure 4.9. 1 KiB/way, 32 bytes/line address mapping example.

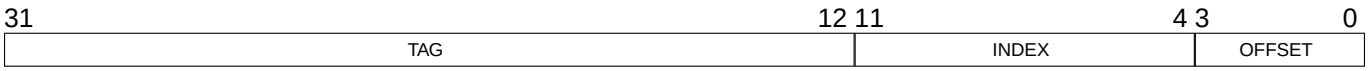


Figure 4.10. 4 KiB/way, 16 bytes/line address mapping example.

Figures 4.11, 4.12 and 4.13 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. To save area, tag memories can be configured to store only necessary address tag bits depending on on-chip memories size. In such case the MSB defines if current set stores ROM data (0) or RAM data (1).

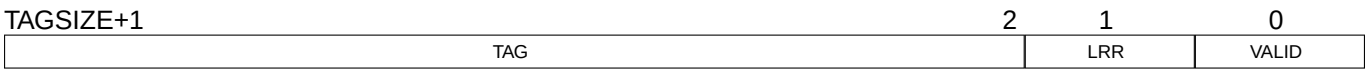


Figure 4.11. Full tag memory entry with LRR bit.

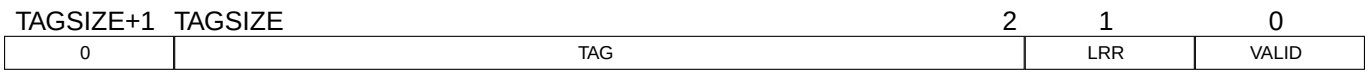


Figure 4.12. Tag memory entry with LRR bit, pointing to ROM region.

Figure 4.13. Tag memory entry without LRR bit, pointing to RAM region.

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.

4.8.1 Registers List

Address Offset	Register	Name
0x30072000	STATUS	Status Register
0x30072004	FLUSH	Flush Register
0x30072008	INFO	Info Register

4.8.2 Status Register

Address: 0x00
Type: exclusive

31	30	···	···	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.

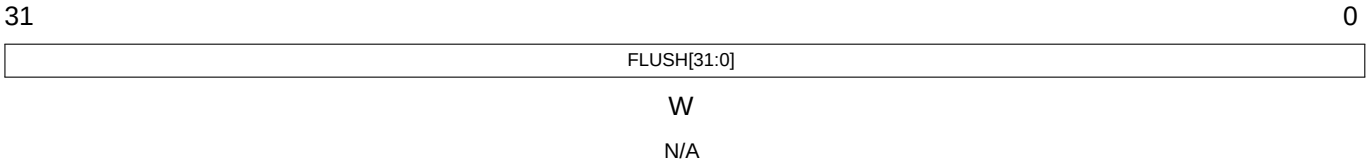
BUSY *Write Buffer Busy*

Bit is set if data cache write buffer is busy.

4.8.3 Flush Register

Address: 0x04

Type: exclusive



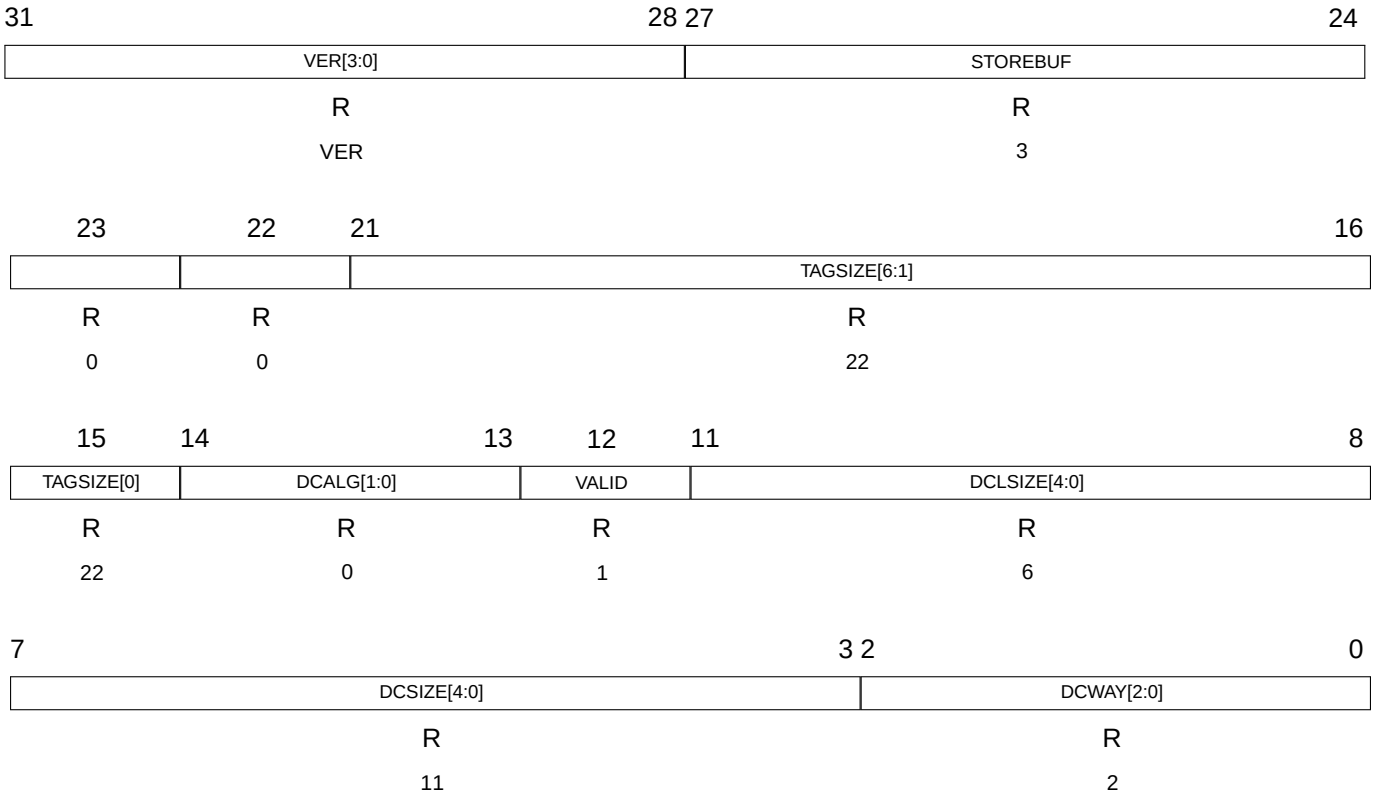
FLUSH[31:0] *Flush Data Tag Memory*

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

4.8.4 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.

5. On-Chip Debugger

CC100-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 serial wire interface or JTAG.

The table below shows the detailed description of Debug Region memory map. The visible content of locations starting from address 0x91000000 depends on current debug context which points to the particular processor core.

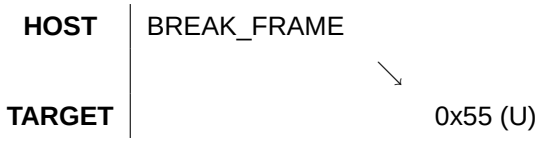
Address	Description
0x90000000 - 0x9000000C	4 x Breakpoint
0x90000010 - 0x9000001C	4 x Watchpoint
0x90000020 - 0x90000020	Burst Counter Register
0x91000000 - 0x9100007C	Integer Register-File
0x91000080 - 0x910000FC	Floating-Point Register-File
0x92000000 - 0x920xxxxx	Instruction Cache Data Way 0
	Instruction Cache Data Way 1
	. . .
	Instruction Cache Data Way *n*
0x92100000 - 0x921xxxxx	Instruction Cache Tag Way 0
	Instruction Cache Tag Way 1
	. . .
	Instruction Cache Tag Way *n*
0x92200000 - 0x922xxxxx	Data Cache Data Way 0
	Data Cache Data Way 1
	. . .
	Data Cache Data Way *n*
0x92300000 - 0x923xxxxx	Data Cache Tag Way 0
	Data Cache Tag Way 1
	. . .
	Data Cache Tag Way *n*

5.1 Serial Wire Debug

Serial Wire Debug unit consists of a 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.

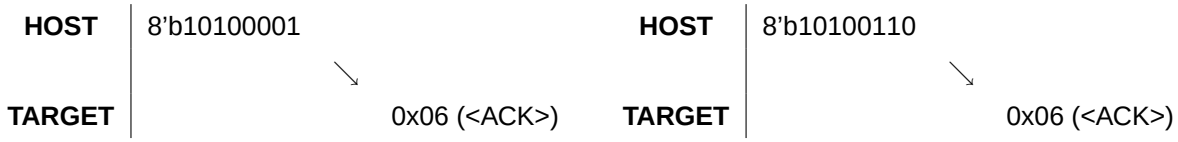
5.1.1 Baud Rate Detection

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.



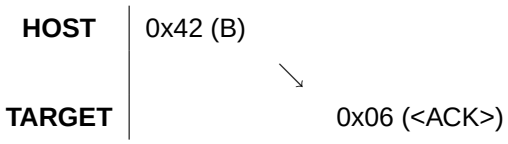
5.1.2 Debug Context

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.

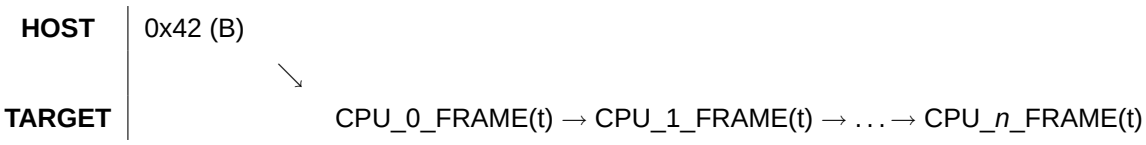


5.1.3 Detecting Debug Mode

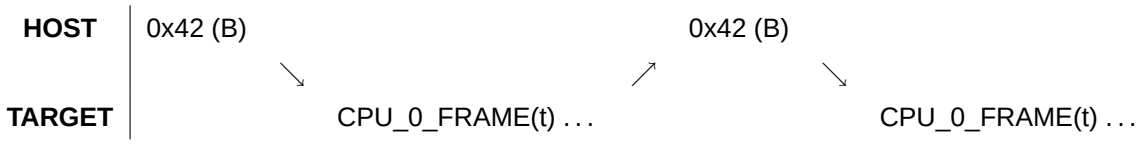
The detection of current processor mode can be done by sending 0x42 (B) char. If processor is not in the debug mode, the target will respond with 0x06 (<ACK>) char.



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.



5.1.4 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.

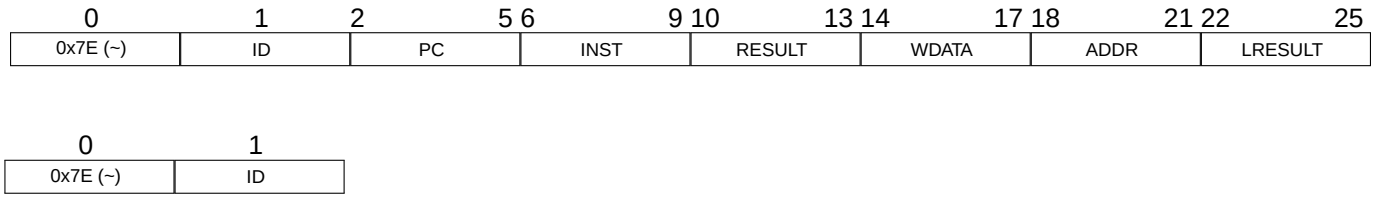


Figure 5.1. Long and short processor core state frame structure.

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

5.1.5 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. Finally, 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:

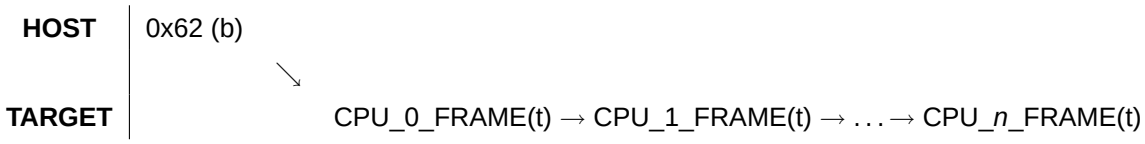
- executing a BREAK instruction with 0x0F code
- hardware breakpoint/watchpoint hit
- executing debugger Break Command

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) → CPU_1_FRAME(t) → . . . → CPU_*n*_FRAME(t)

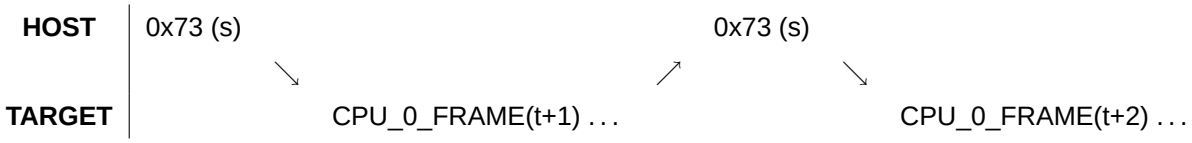
5.1.6 Break Command

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.



5.1.7 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.



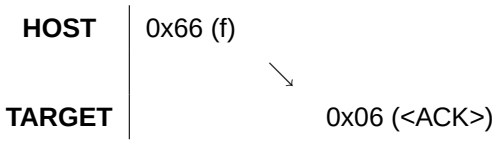
5.1.8 Leaving Debug Mode

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

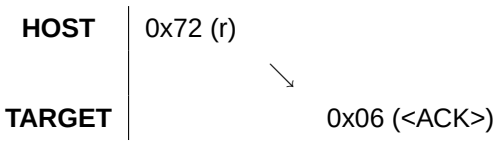
5.1.9 Free Running Command

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.



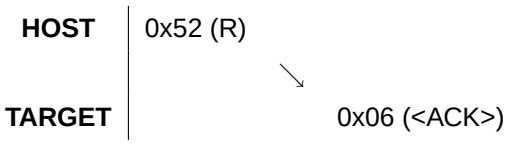
5.1.10 Processor Reset Command

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.



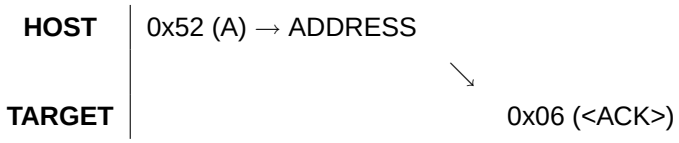
5.1.11 Debugger Reset Command

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.



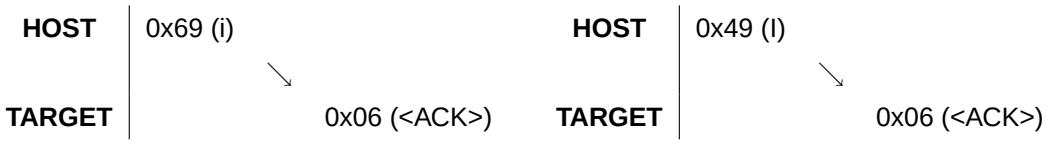
5.1.12 Address Command

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.



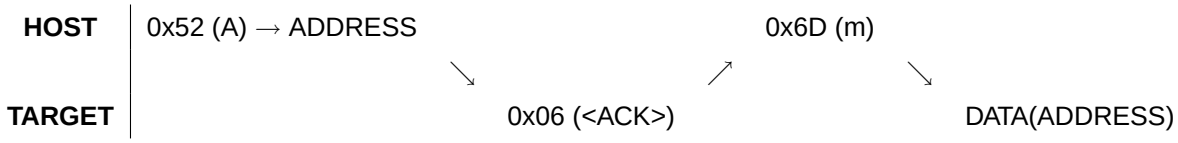
5.1.13 Address Auto-Increment On/Off Command

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.

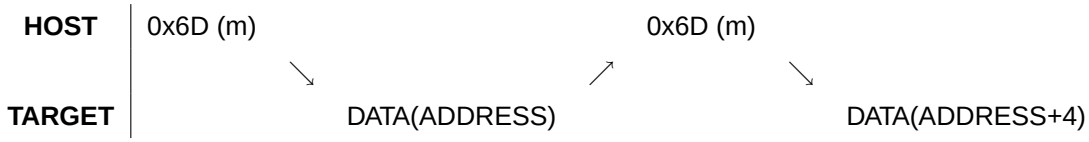


5.1.14 Memory Read Command

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).

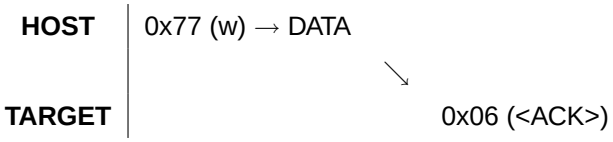


When the autoincrementation is enabled every Read Memory Command execution will dump consecutive memory location content.

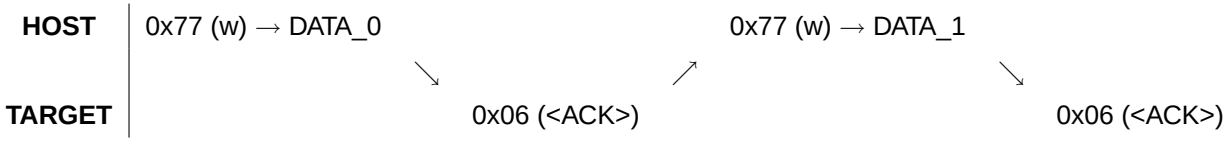


5.1.15 Memory Write Command

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).

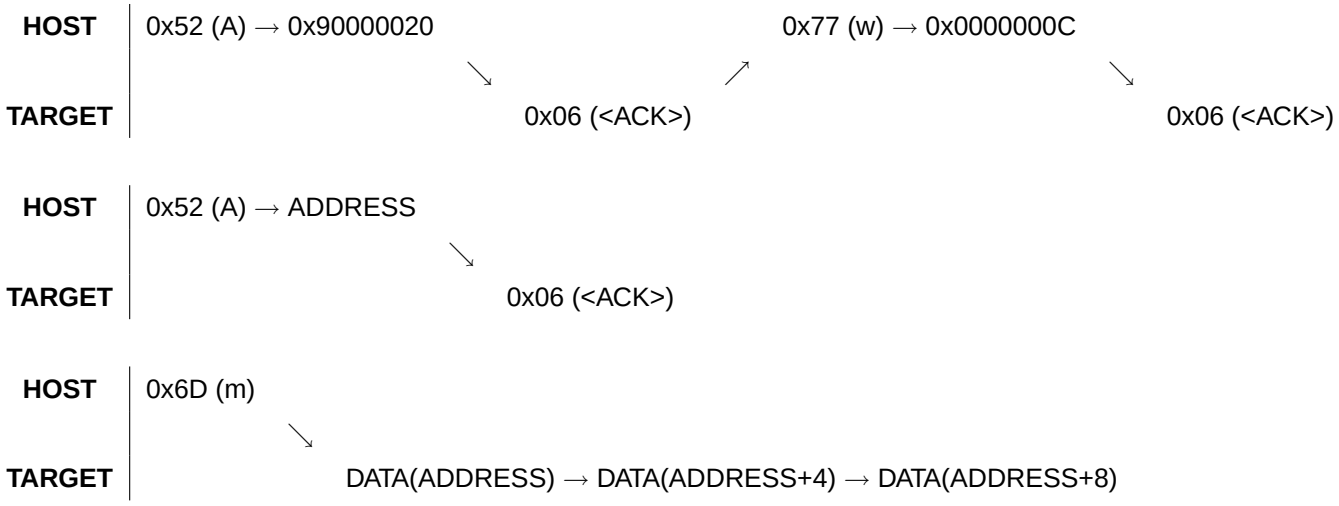


When the autoincrementation is enabled every Memory Write Command execution will store data to the consecutive memory location.

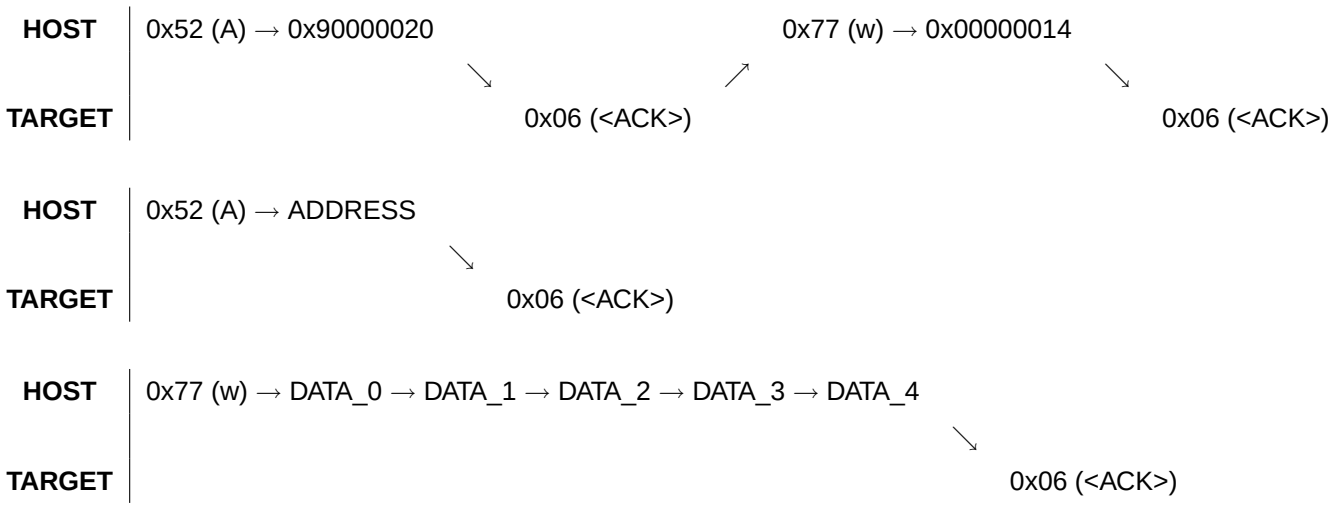


5.1.16 Burst Counter Register

The On-Chip Debugger Burst Counter Register is located at address 0x90000020. 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.


HOST
TARGET
0x52 (A) → 0x90000020
0x06 (<ACK>)
0x77 (w) → 0x00000014
0x06 (<ACK>)
HOST
TARGET
0x52 (A) → ADDRESS
0x06 (<ACK>)
HOST
TARGET
0x77 (w) → DATA_0 → DATA_1 → DATA_2 → DATA_3 → DATA_4
0x06 (<ACK>)

5.2 JTAG Interface

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 following 4-bit instructions:

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 betweeb 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.

5.2.1 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 5.2.

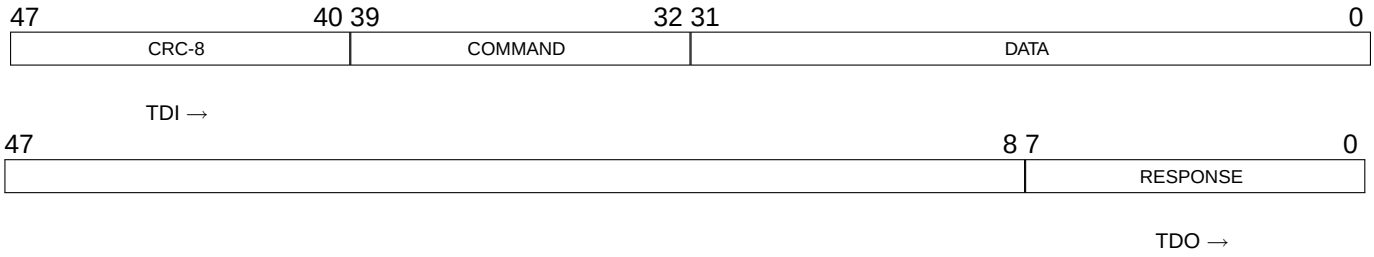


Figure 5.2. Debug Command Register structure.

5.2.2 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 5.3.

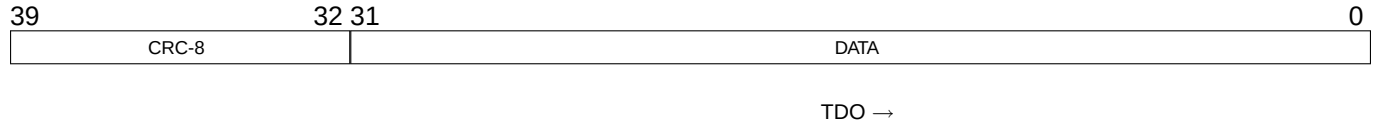


Figure 5.3. Debug Data Register structure.

5.2.3 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:

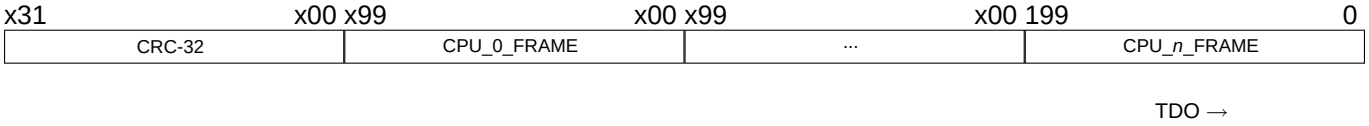


Figure 5.4. Debug Stauts Register structure.

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.

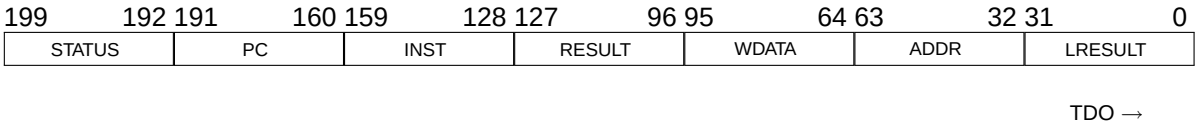


Figure 5.5. JTAG processor core state frame structure.

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

5.2.4 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

5.2.5 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.

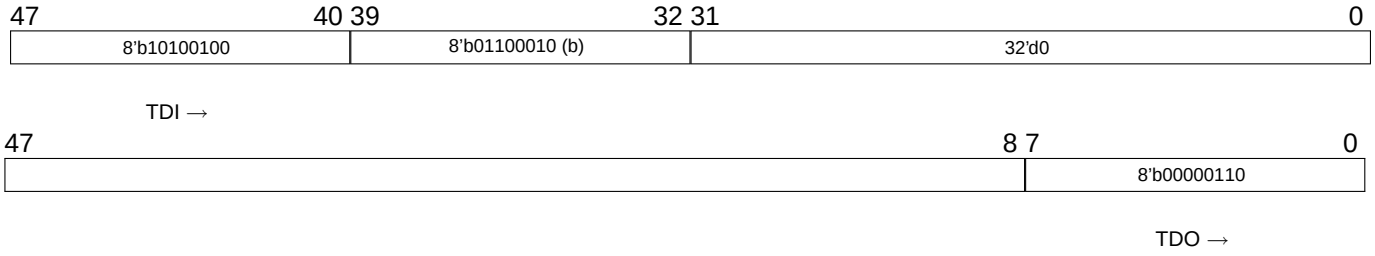


Figure 5.6. Break Command structure.

5.2.6 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.

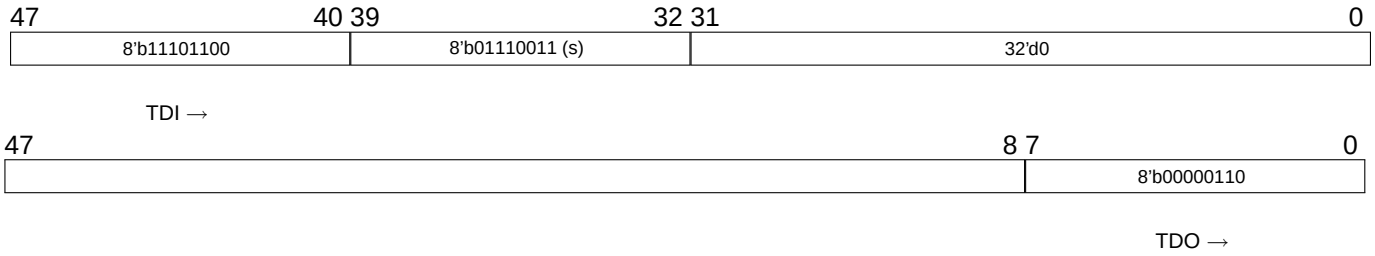


Figure 5.7. Step Command structure.

5.2.7 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

5.2.8 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.

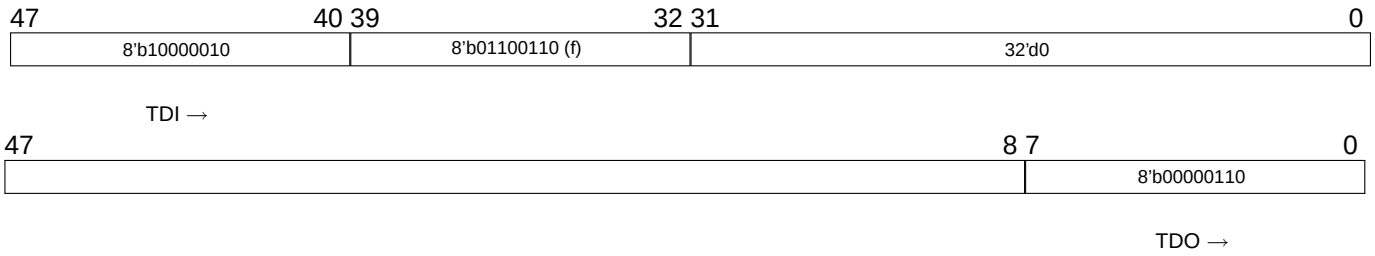


Figure 5.8. Free Running Command structure.

5.2.9 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.

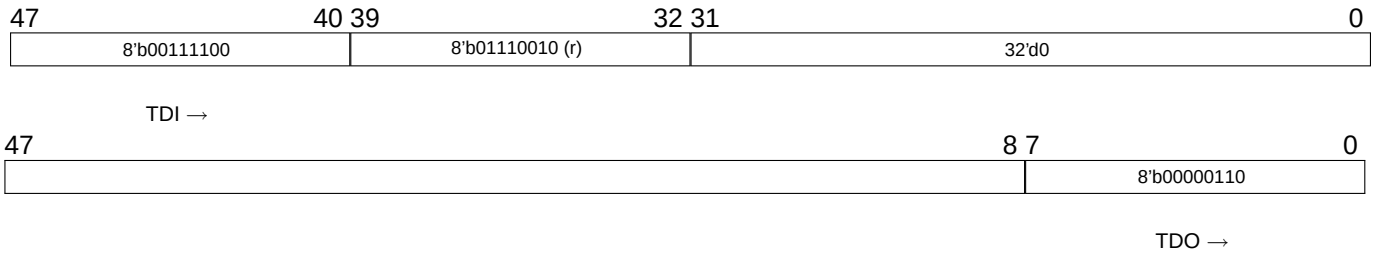


Figure 5.9. Reset Command structure.

5.2.10 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.

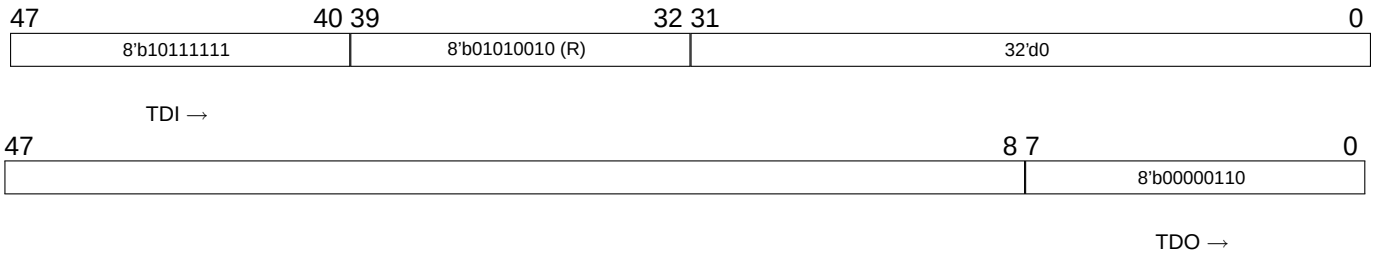


Figure 5.10. Debugger Reset Command structure.

5.2.11 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.

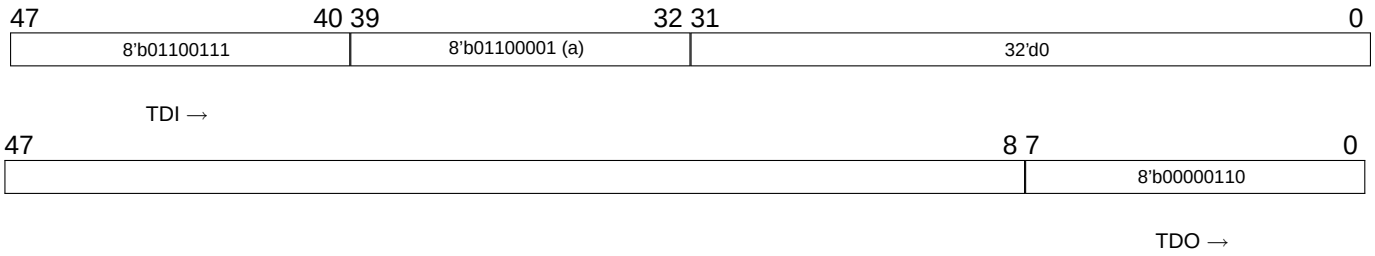


Figure 5.11. Address Command structure.

5.2.12 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.

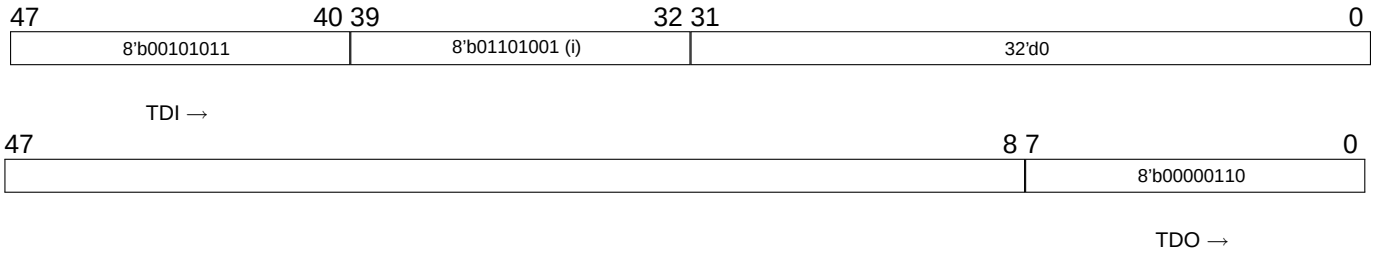


Figure 5.12. Auto-Increment Off Command structure.

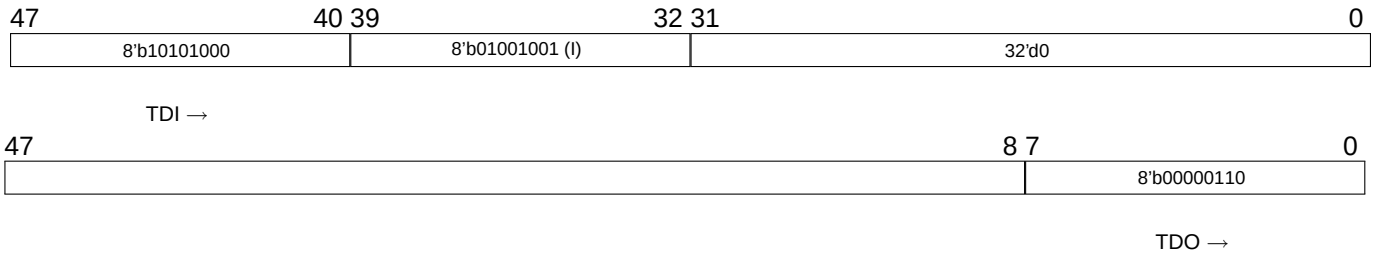


Figure 5.13. Auto-Increment On Command structure.

5.2.13 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).

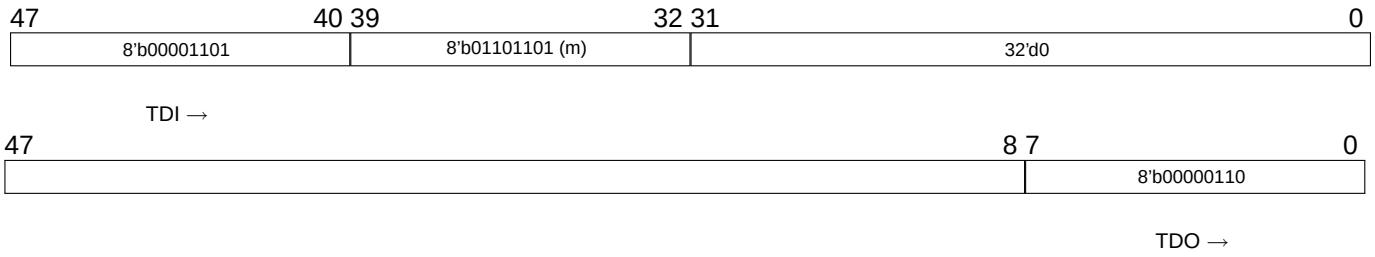


Figure 5.14. Memory Read Command structure.

5.2.14 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.

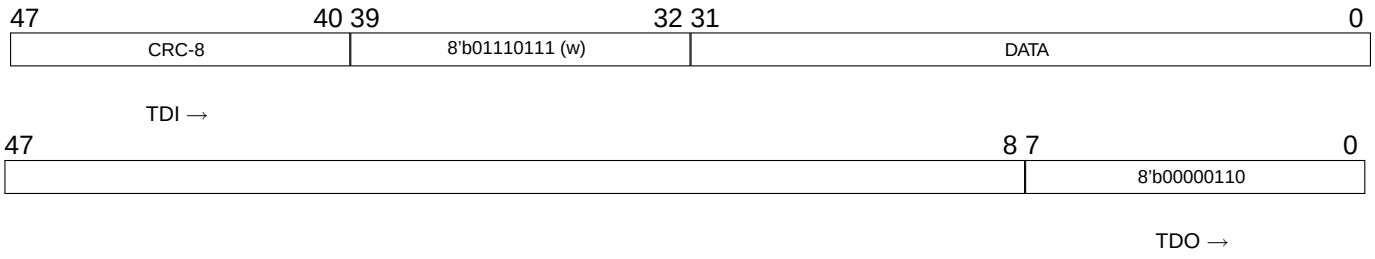


Figure 5.15. Memory Write Command structure.

ChipCraft Sp. z o.o. Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND www.chipcraft-ic.com

 CC100-C-Doc_082018.