XBurst®2 CPU Core

Programming Manual

Release Date: June 2, 2017

北京君正集成电路股份有限公司
Ingenic Semiconductor Co.,Ltd.

XBurst®2 CPU Core

Programming Manual

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10
Xibeiwang East Road, Haidian District, Beijing, China,
Tel: 86-10-56345000
Fax:86-10-56345001
Http: //www.ingenic.com

CONTENTS

1 Overview 5

1.1 Features of XBurst®2 CPU for X2000 5

2 Operating Modes 6

3 CP0 8

3.1 CP0 Register Summary 8

3.1.1 CP0 Registers Grouped by Function 8

3.1.2 CP0 Registers Grouped by Number 10

3.2 CP0 register Formats 12

3.2.1 CP0 Register Field Types 12

3.3 CP0 Register Descriptions 13

3.3.1 CPU Configuration and Status Registers 13

3.3.2 TLB Management Registers 29

3.3.3 Exception Control Registers 38

3.3.4 Timer Registers 43

3.3.5 Cache Management Registers 44

3.3.6 Thread Context and Shadow Control Registers 45

3.3.7 CPU Performance Monitor Registers 46

3.3.8 Debug Registers 48

3.3.9 User Mode Support Registers 54

3.3.10 Kernel Mode Scratch Registers 56

4 Exceptions and Interrupts 58

4.1 Exception Priority 58

4.2 Exception Vector Locations 59

4.3 Exception Handling Process 60

4.3.1 Enter Exception Handler Routine 60

4.3.2 Return from Exception Handler Routine 60

4.4 Exception Categories 61

5 Memory Management Unit 63

5.1 Overview 63

5.2 Virtual Memory Space 64

5.2.1 User Mode 64

5.2.2 Kernel Mode 65

5.3 TLB 67

5.3.1 Instruction Micro TLB 67

5.3.2 Data Micro TLB 67

5.3.3 Variable Page Size TLB (VTLB) 67

5.3.4 Fixed Page Size TLB (FTLB) 70
5.3.5 Filling JTLB Entry 71
5.4 Virtual to Physical Address Translation 72

6 Caches 73

6.1 L1-cache 73
6.1.1 Cache Coherency Attribute 73
6.1.2 Cache Relative CP0 Registers 74
6.1.3 Cache Operation Relative Instructions 74
6.1.4 PREF/PREFX instruction 77
6.1.5 SYNC instruction 78
6.2 L2-cache 79

7 Initialize Core State 80

7.1 Initialized Core State by Hardware 80
7.1.1 Coprocessor 0 State 80
7.1.2 TLB Initialization 80
7.1.3 Cache Initialization 80
7.2 Initialized Core State by Software 80
7.2.1 General Purpose Registers Initialization 80

8 CCU 81

8.1 Overview 81
8.2 Register Description 82
8.2.1 Cores SLeep Control Register 83
8.2.2 Core Sleep Status Register 84
8.2.3 Core Software Reset Register 85
8.2.4 Memory Subsystem Control Register 86
8.2.5 Memory Subsystem Implementation Register 87
8.2.6 CPU Configuration Register 88
8.2.7 Peripheral IRQ Pending Register 89
8.2.8 Peripheral IRQ Mask Register 89
8.2.9 Mailbox IRQ Pending Register 90
8.2.10 Mailbox IRQ Mask Register 90
8.2.11 OST IRQ Pending Register 91
8.2.12 OST IRQ Mask Register 91
8.2.13 Debug Interrupt Pending Register 92
8.2.14 Debug Interrupt Mask Register 92
8.2.15 Reset Entry Register 93
8.2.16 Mailbox Register<N> 93
8.2.17 CCU Spin Lock Register 94
8.2.18 CCU Spin Atomic Register 94
8.2.19 Global Interrupt Mask Register 95

8.2.20 CPU Feature Configuration Register ... 96
8.2.21 Bus Exception Control Register ... 97
8.3 Usage ... 98
8.3.1 The Configuration of CCU ... 98
9 EJTAG Debug Support ... 99
9.1 Overview ... 99
9.2 Detecting Debug Mode ... 100
9.3 Ways of Entering Debug Mode ... 100
9.4 Exiting Debug Mode ... 100
9.5 Hardware Breakpoints ... 100
9.5.1 Instruction Breakpoints ... 101
9.5.2 Data Breakpoints ... 101
9.5.3 Overview of Instruction Breakpoint Registers ... 101
9.5.4 Overview of Data Breakpoint Registers ... 101
9.6 Conditions for Matching Breakpoints ... 102
9.6.1 Conditions for Matching Instruction Breakpoints ... 102
9.6.2 Conditions for Matching Data Breakpoints ... 102
9.6.3 SIMD Load/Store Handling ... 103
9.7 Debug Exceptions from Breakpoints ... 103
9.7.1 Debug Exception by Instruction Breakpoint ... 104
9.7.2 Debug Exception by Data Breakpoint ... 104
9.7.3 Breakpoint Used as Triggerpoint ... 104
9.8 Test Access Port (TAP) ... 104
9.8.1 EJTAG Internal and External Interfaces ... 105
9.8.2 Test Access Port Operation ... 105
9.8.3 Test Access Port (TAP) Instructions ... 109
9.8.4 TAP Processor Accesses ... 111
9.9 EJTAG Registers ... 113
9.9.1 General Purpose Control and Status ... 113
9.9.2 Instruction Breakpoint Registers ... 114
9.9.3 Data Breakpoint Registers ... 117
9.9.4 EJTAG TAP Registers ... 122
9.10 Debug Exception ... 130
9.10.1 Debug Exception Priorities ... 130
9.10.2 Debug Exception Vector Location ... 131
9.10.3 General Debug Exception Processing ... 131
9.10.4 Debug Single Step Exception ... 132
9.10.5 Debug Interrupt Exception ... 133
9.10.6 Debug Instruction Break Exception ... 133
9.10.7 Debug Breakpoint Exception ... 133
9.10.8 Debug Data Break on Load/Store Exception ... 134
9.11 Debug Mode Exceptions ... 134

9.11.1 Exceptions Taken in Debug Mode 134
9.11.2 Debug Mode Exception Processing 135
9.12 MIPS EJTAG Compliant Mode 136
9.13 Accelerated EJTAG Mode 136
9.13.1 ACC Mode Flag 136
9.13.2 EJTAG Control Register in ACC mode (ECR_A) 136
9.13.3 Processor Access Address Register in ACC mode (ADDRESS_A) 138
9.13.4 Processor Access Data Register in ACC mode (DATA_A) 138
9.13.5 Debug Mode Address Space in Compliant Mode (AM = 0) 139
9.13.6 Debug Mode Address Space in ACC Mode (AM = 1) 139
9.13.7 Supported JTAG Instructions in ACC Mode 140
Revision History 141

1 Overview

XBurst®2 CPU is a high performance and low power implementation of MIPS32 ISA r5 + Configurable SIMD ISA (MIPS MSA128 + XBurst® MXA128 for X2000) instruction set architecture.

1.1 Features of XBurst®2 CPU for X2000

- 32-bit MIPS32 ISA R5 plus MIPS SIMD instruction set architecture:MSA128
- Ingenic SIMD instruction set architecture: MXA128
- dual-issue, superscalar, super pipeline with Simultaneous Multi-Threading (SMT)
  - Two hardware threads per physical core
  - Quad instructions fetch per cycle
  - Dual-issue instructions per hardware thread per cycle
  - Smart branch prediction scheme with Instruction trace buffer (ITB), Branch Target Buffer (BTB), Pattern History Table (PHT), Return Address Stack (RAS) and Jump Target Buffer (RTB)
- L1 cache size up to 32KB for Instruction cache and Data cache, respectively, 8-way set associative
- high-performance dual-issue floating-point (single and double) and 128-bit SIMD Unit
  - 32 x 128-bit registers, 128-bit loads/stores to/from SIMD unit
  - data types: 8/16/32 bits integer, Q15/Q31 fixed point and 32/64 bits floating point
  - IEEE-754 2008 compliance
- Programmable Memory Management Unit (MMU)
  - 1st level mini-TLB (MTLB): 8-entry instruction MTLB, 16-entry data MTLB
  - 2nd level joint TLBs: 32-entry VTLB, 256-entry 4-way set associative FTLB
- Integrated Cache Crossbar (CCX) and L2 cache
  - Configurable size up to 1MB
  - Support multi-core configurations up to 4 physical cores (4 cores with total 8 threads)
  - Hardware prefetcher for streaming performance
- Advanced power management
  - Hardware automatic clock gating for idle subsystems
- XBurst2 CPU implements SMT technology - a physical core contains 2 simultaneous threads. For convenience of description, a thread can be regarded as a logic core. Therefore, except explicitly claim, a core means a logic core in later chapters.

2 Operating Modes

XBurst®2 CPU core supports three operating modes:

- Debug Mode
- Kernel Mode
- User Mode

Table 2-1 the conditions of operating mode

Debug.DM	Status.UM	Status.EXL	Status.ERL	Operating Mode
1	X	X	X	Debug Mode
0	0	X	X	Kernel Mode
0	X	1	X	
0	X	X	1	
0	1	0	0	User Mode

X denotes don't care.

3 CP0

The XBurst2 System Control Coprocessor (CP0) provides the register interface to the XBurst2 core and supports memory management, address translation, exception handling, and other privileged operations. Each CP0 register has a unique number that identifies it, referred to as its register number. A separate select number is used to differentiate additional registers within the register number. For example, as shown in the table below, there are eight configuration registers with register number 16. If the select number is omitted, it is zero.

This chapter contains the following sections:

• Section 3.1 “CP0 Register Summary”

• Section 3.2 “CP0 Register Formats”

• Section 3.3 “CP0 Register Descriptions”

3.1 CP0 Register Summary

The following two subsections show the CP0 register set, Table 3-1 CP0 Register Grouped by Function and Table 3-2 CP0 Registers Grouped by Number.

3.1.1 CP0 Registers Grouped by Function

The CP0 registers set are divided into several register groups listed below.

Table 3-1 CP0 Register Grouped by Function

Category	Register Name	Register Number	Register Select	
CPU Configuration and Status	Config	16	0	section 3.3.1.1
	Config1	16	1	section 3.3.1.2
	Config2	16	2	section 3.3.1.3
	Config3	16	3	section 3.3.1.4
	Config4	16	4	section 3.3.1.5
	Config5	16	5	section 3.3.1.6
	PRId	15	0	section 3.3.1.7
	EBase	15	1	section 3.3.1.8
	Status	12	0	section 3.3.1.9
	IntCtl	12	1	section 3.3.1.10
TLB Management	Index	0	0	section 3.3.2.1
	Random	1	0	section 3.3.2.2
	EntryLo0	2	0	section 3.3.2.3
	EntryLo1	3	0	section 3.3.2.3
	EntryHi	10	0	section 3.3.2.4
	Context	4	0	section 3.3.2.5
	PageMask	5	0	section 3.3.2.6
	PageGrain	5	1	section 3.3.2.7
	Wired	6	0	section 3.3.2.8

	BadVAddr	8	0	section 3.3.2.9
Exception Control	Cause	13	0	section 3.3.3.1
	EPC	14	0	section 3.3.3.2
	ErrorEPC	30	0	section 3.3.3.3
Timer Registers	Count	9	0	section 3.3.4.1
	Compare	11	0	section 3.3.4.2
Cache Management	TagLo	28	0	section 3.3.5.1
	DataLo	28	1	section 3.3.5.2
Shadow Registers	SRStl	12	2	section 3.3.6.1
	SRSMap	12	3	section 3.3.6.2
Performance Monitoring	PerfCntCtl0	25	0	section 3.3.7.1
	PerfCntCnt0	25	1	section 3.3.7.2
	PerfCntCtl1	25	2	section 3.3.7.1
	PerfCntCnt1	25	3	section 3.3.7.2
Debug	Debug	23	0	section 3.3.8.1
	Debug2	23	6	section 3.3.8.2
	DEPC	24	0	section 3.3.8.3
	DESAVE	31	0	section 3.3.8.4
	WatchLo	18	0	section 3.3.8.5
	WatchHi	19	0	section 3.3.8.6
User Mode Support	HWREna	7	0	section 3.3.9.1
	UserLocal	4	2	section 3.3.9.2
	LLAddr	17	0	section 3.3.9.3
Kernel Mode Support	KScratch1	31	2	section 3.3.10.1
	KScratch2	31	3	section 3.3.10.2
	KScratch3	31	4	section 3.3.10.3
	KScratch4	31	5	section 3.3.10.4
	KScratch5	31	6	section 3.3.10.5
	KScratch6	31	7	section 3.3.10.6

3.1.2 CP0 Registers Grouped by Number

The following table provides a numerical list of the processor CP0 register.

Table 3-2 CP0 Registers Grouped by Number

Register			Function	Location
Num	Sel	Name		
0	0	Index	Index into the TLB array	section 3.3.2.1
1	0	Random	Randomly generated index into the TLB array	section 3.3.2.2
2	0	EntryLo0	Low-order portion of the TLB entry for even-numbered virtual pages.	section 3.3.2.3
3	0	EntryLo1	Low-order portion of the TLB entry for odd-numbered virtual pages.	section 3.3.2.3
4	0	Context	Pointer to page table entry in memory.	section 3.3.2.5
4	2	UserLocal	It is written and interpreted by software	section 3.3.9.2
5	0	PageMask	PageMask controls the variable page sizes in TLB entries.	section 3.3.2.6
5	1	PageGrain	PageGrain controls the granularity of the page sizes in TLB entries.	section 3.3.2.7
6	0	Wired	Controls the number of fixed ("wired") TLB entries.	section 3.3.2.8
7	0	HWREna	Enable access to selected hardware registers in non-privileged mode via the RDHWR instruction.	section 3.3.9.1
8	0	BadVaddr	The virtual address triggering the most recent address-related exception.	section 3.3.2.9
9	0	Count	Interval counter.	section 3.3.4.1
9	6	SpinLock	Spinlock register.	
10	0	EntryHi	High-order portion of the TLB entry.	section 3.3.2.4
11	0	Compare	Compare value for interval count.	section 3.3.4.2
12	0	Status	Processor status and control.	section 3.3.1.9
12	1	IntCtl	Setup for interrupt vector and interrupt priority features.	section 3.3.1.10
12	2	SRSCtl	Shadow register set control	section 3.3.6.1
12	3	SRSMap	Shadow register map	section 3.3.6.2
13	0	Cause	Cause of last exception	section 3.3.3.1
14	0	EPC	Program counter of resuming after servicing the most recent normal exception.	section 3.3.3.2
15	0	RPID	Processor identification and revision	section 3.3.1.7
15	1	EBase	Exception handler's base address.	section 3.3.1.8
16	0	Config	Configuration register.	section 3.3.1.1
16	1	Config1	Configuration for MMU, catches etc.	section 3.3.1.2
16	2	Config2	Configuration for MMU ,caches etc.	section 3.3.1.3
16	3	Config3	Interrupt and ASE capabilities.	section 3.3.1.4

16	4	Config4	Indicates presence of Config5 register	section 3.3.1.5
16	5	Config5	Provides information on EVA and cache error exception vector	section 3.3.1.6
17	0	LLAddr	The physical address of the load operation for the most recent Load Linked (LL) instruction.	section 3.3.9.3
18	0	WatchLo	Watchpoint address. (low order)	section 3.3.8.5
19	0	WatchHi	Watchpoint address (high order) and mask.	section 3.3.8.6
23	0	Debug	EJTAG Debug register.	section 3.3.8.1
23	6	Debug2	EJTAG Debug 2 register.	section 3.3.8.2
24	0	DEPC	Program counter of resuming after servicing the most recent debug/debug-mode exception.	section 3.3.8.3
25	0	PerfCntCtl0	Performance counter 0 control	section 3.3.7.1
25	1	PerfCntCnt0	Performance counter 0 count	section 3.3.7.2
25	2	PerfCntCtl1	Performance counter 1 control	section 3.3.7.1
25	3	PerfCntCnt1	Performance counter 1 count	section 3.3.7.2
28	0	TagLo	Cache tag read/write interface for I-Cache and D-Cache.	section 3.3.5.1
28	1	DataLo	Low-order data read/write interface for I-Cache and D-Cache	section 3.3.5.2
30	0	ErrorEPC	Program counter of resuming after servicing the most recent Error exception (like reset).	section 3.3.3.3
31	0	DESAVE	Debug handler scratchpad register.	section 3.3.8.4
31	2	KScratch1	Kernel scratch pad register 1.	section 3.3.10.1
31	3	KScratch2	Kernel scratch pad register 2.	section 3.3.10.2
31	4	KScratch3	Kernel scratch pad register 3.	section 3.3.10.3
31	5	KScratch4	Kernel scratch pad register 4.	section 3.3.10.4
31	6	KScratch5	Kernel scratch pad register 5.	section 3.3.10.5
31	7	KScrathc6	Kernel scratch pad register 5.	section 3.3.10.6

3.2 CP0 register Formats

This section contains descriptions of each CP0 register. The registers are listed in numerical order, first by register number, then by select field number.

3.2.1 CP0 Register Field Types

For each register described below, field descriptions include the read/write properties of the field, and the reset state of the field. The read/write properties are described in Table 3-3.

Table 3-3 CP0 Register Field R/W Access Types

Notation	Hardware Interpretation	Software Interpretation
R/W	A field in which all bits are readable and writeable by software and potentially by hardware. Hardware updates of this field are visible by software reads. Software updates of this field are visible by hardware reads. If the reset state of this field is "Undefined", either software or hardware must initialize the value before the first read will return a predictable value. This should not be confused with the formal definition of UNDEFINED behavior.	
R	A field that is either static or is updated only by hardware. If the reset state of this field is either "0" or "1", hardware initializes this field.	A field to which the value written by software is ignored. Software may write any value to this field without affecting hardware behavior. Software reading the field will return the last value updated by hardware. If the reset state of this field is "Undefined," software reading of this field results in an UNPREDICTABLE value except after a hardware update done under the conditions specified in the description of the field.
W	A field that can be written by software but cannot be read by software. Software reading of this field will return an UNDEFINED value.	
W0	Hardware can write "0" or "1" to this field	Software writes always causes the field to be cleared to zero.
W1C	Hardware can write "0" or "1" to this field.	Software should write "1" to this field to clear it.
Reserved	A field that hardware does not update, and assumes a zero value.	A field to which the write operation by software is always ingored. And software reading of this field will return zero.

3.3 CP0 Register Descriptions

The following subsections describe the CP0 registers listed in above.

3.3.1 CPU Configuration and Status Registers

This section contains the following CPU Configuration and Status registers.

- Device Configuration - Config(CP0 Register16, Select 0)
- Device Configuration 1-Config 1(CP0 Register16, Select1)
- Device Configuration 2-Config 2(CP0 Register16, Select2)
- Device Configuration 3-Config 3(CP0 Register16, Select3)
- Device Configuration 4-Config 4(CP0 Register16, Select4)
- Device Configuration 5-Config 5(CP0 Register16, Select5)
- Processor ID-PRId(CP0 Register 15, Select 0)
- Exception Base Address-EBase(CP0 Register 15, Select 1)
- Status (CP0 Register 12, Select 0)
- Interrupt Control - IntCtl (CP0 Register 12, Select 1)

3.3.1.1 Device Configuration - Config(CP0 Register16, Select 0)

The *Config* register specifies various configuration and capabilities information. Most of the fields in the *Config* register are initialized by hardware reset, or are constant value.

Config Register

31	30 28	27 25	24 16	15	14 13	12 10	9 7	6 4	3	2 0
M	K23	KU	0	BE	AT	AR	MT	0	VI	K0

Name	Bits	Description	R/W	Reset
M	31	This bit is hardwired to "1" to indicate the presence of the *Config1* register.	R	1
K23	30:28	These fields are unused because XBurst CPU does not support fixed mapping MMU.	R	0
KU	27:25		R	0
0	24:16	returns zero on read and must be written as zero	R	0

BE	15	Indicates the endian mode. XBurst CPU supports little endian only.
		0: littel endian;
		1: big endian;
			R	0

AT	14:13	Architecture type implemented by the processor. Hardwired to 2’b00 which indicates that the architecture type is MIPS32. This field only denotes address and register width. The exact implemented instruction sets are denoted by the ISA	R	0

		register field of *Config3.*		
AR	12:10	Architecture revision level. This bit always reads 1 to reflect Release 5 of MIPS32 architecture.	R	1
MT	9:7	MMU type. This field is hardwired to 3’b100 to indicate a VTLB+FTLB MMU.	R	4
0	6:4	Must be written as zero; returns zero on read	R	0
VI	3	Virtual instruction cache. This field is hardwired to 1'b0 to indicate the instruction cache of XBurst2 CPU is not virtual.	R	0
K0	2:0	Kseg0 cache attributes. SeeTable 6-2 Cache Coherency Attributes for detail	R/W	2

3.3.1.2 Device Configuration 1-Config 1(CP0 Register16, Select1)

The *Config1* register is an adjunct to the *Config* register and encodes additional capabilities information. All fields in the *Config1* register are read-only.

The Icache and Dcache configuration parameters include encoding for the number of sets per way, the line size, and the associativity. The total cache size for a cache is therefore:

Cache Size = Associativity * Line Size * Set Per Way

Config1 Register

31	30 25	24 22	21 19	18 16	15 13	12 10	9 7	6	5	4	3	2	1	0
M	MMU size	IS	IL	IA	DS	DL	DA	C2	MD	PC	WR	CA	EP	FP

Name	Bits	Description	R/W	Reset
M	31	This bit is hardwired to ‘1’ to indicate the presence of the Config2 register .	R	1
MMU size	30:25	The size of the VTLB array (the array has MMUSize + 1 entries). Refer to the *Config4* register for more information. 0x1F indicates 32 VTLB entries for this implementation.	R	0x1F
IS	24:22	L1 instruction cache number of sets per way. This field is encoded as follows: 0 : Reserved 1 : 128 sets per way 2~7 : Reseveed.	R	1
IL	21:19	L1 instruction cache line size. This field is encoded as follows: 0~3 : Reserved 4 : 32-byte line size 5~7 : Reserved.	R	4
IA	18:16	L1 Instruction cache associativity. This field is encodes as follows: 0~6 : Reserved 7 : 8-way	R	7
DS	15:13	L1 Data cache number of sets per way. This field is encodes as follow: 0 : Reserved 1 : 128 set per way 2~7 : Reseveed.	R	1
DL	12:10	L1 data cache line size. This field is encoded as follows: 0~3 : Reserved 4 : 32-byte line size 5~7 : Reserved.	R	4
DA	9:7	L1 data cache associativity.This field is encoded as follows: 0~6 : Reserved 7 : 8-way	R	7
C2	6	Coprocessor 2 implemented:	R	1

		0: Coprocessor 2 not implemented. 1: Coprocessor 2 implemented.		
MD	5	MDMX Application Specific Extension (ASE). 0x0: indicates that the MDMX ASE is not implemented	R	0
PC	4	Performance Counter implemented. Performance counter always is implemented. Hence this bit is always logic '1'. Refer to the *PerfCtl0-1* and *PerfCnt0*-1 registers for more information.	R	1
WR	3	Watch registers implemented. Refer to *WathcLo/WatchHi* registers for more information.	R	1
CA	2	MIPS16e present. This bit always reads as 0 to indicate no support of MIPS16e ISA	R	0
EP	1	EJTAG implemented. This bit always reads as 1 to indicate the EJTAG unit is implemented.	R	1
FP	0	FPU implemented. If an FPU is implemented, further capabilities of the FPU can be read from the capability bits in the FIR register belonging to CP1 registers.	R	1

3.3.1.3 Device Configuration 2-Config 2(CP0 Register16, Select2)

The *Config2* register encodes level 2 cache configurations.(level 3 cache is not implemented).

Config2 Register

31	30 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
M	TU	TS	TL	TA	SU	SS	SL	SA

Name	Bits	Description	R/W	Reset
M	31	This bit is hardwired to '1' to indicate the presence of the Config3 register.	R	1
TU	30:28	L3 cache is not implemented for XBurst2 CPU, hence the bit fields of TU, TS,TL,TA are not used and are all tied to 0.	R	0
TS	27:24		R	0
TL	23:20		R	0
TA	19:16		R	0
SU	15:13	Version of L2. 0 : V0.0 1 : V1.0 2 : V2.0 3-8 : reserved	R	2
SS	11:8	L2 cache sets per way.This field is encoded as follows: 0, 1, 6, 7 : Reserved 2 : 256 set per way 3 : 512 set per way 4 : 1024 set per way 5 : 2048 set per way	R	4
SL	7:4	L2 cache line size. This field is encoded as follows: 0~4, 6, 7 : Reserved 5 : 64-byte line size	R	5
SA	3:0	L2 cache associativity.This field is encoded as follows: 0~6 : Reserved 7 : 8-way 8~14 : Reserved 15 : 16-way	R	15

3.3.1.4 Device Configuration 3-Config 3(CP0 Register16, Select3)

Config 3 provides information about the presence of optional extensions to the base MIPS32 architecture in addition to those specified in Config 2. All fields in the Config3 register are read-only.

Config3 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	0	CMGCR	MSAP	BP	BI	SC	PW	VZ	IPLW		MMAR			MCU	ISAOnExc	ISA		ULRI	RXI	DSP2P	DSPP	CTXTC	ITL	LPA	VEIC	VInt	SP	CDMM	MT	SM	TL

Name	Bits	Description	R/W	Reset
M	31	Denotes the presence of *Config4* register.	R	1
0	30	Must be written as zero; returns zero on read	R	0
CMGCR	29	Coherency Manager memory-mapped Global Configuration Register Space. 0 denotes it is not implemented.	R	0

MSAP	28	MSA Present.
		0: MSA is not implemented;
		1: MSA is implemented;
			R	1

BP	27	*BadInstrP* register implemented. Always read as zero, indicating that it is not implemented.	R	0
BI	26	*BadInstr* register implemented. Always read as zero, indicating that it is not implemented.	R	0
SC	25	Segment Control implemented. The bit indicates whether SegCtl0~SegCtl1 are present.	R	0
PW	24	Hardware Page Table Walk implemented. Read as zero , indicating that it is not implemented.	R	0
VZ	23	Virtualization implemented; Read as zero, indicating that it is not implemented.	R	0
IPLW	22:21	Config3$_{MCU}$ is zero, indicating that MCU ASE is not implemented so this field is not used.	R	0
MMAR	20:18	Config3$_{ISA}$ is zero, indicating that microMIPS32 is not implemented so this field is not used.	R	0
MCU	17	MCU ASE implemented. Always read as zero, indicating that it is not implemented.	R	0
ISAOn-Exc	16	Reflects the ISA to be used after vectoring to an exception. 0: MIPS32 is used on entrance to an exception vector; 1: microMIPS is used on entrance to an exception vector; Always read as zero as microMIPS is never implemented.	R	0
ISA	15:14	Indicates Instruction Set Availability. Always read as zero, indicating that MIPS32 is implemented	R	0
ULRI	13	UserLocal register implemented. 0: UserLocal register is not implemented;	R	1

		1: UserLocal register is implemented;		

RXI	12	Indicates whether the RIE and XIE bits exist within the PageGrain register.
		0: RIE and XIE are not implemented;
		1: RIE and XIE are implemented;
			R	1

DSP2P	11	MIPS DSP ASE Revision 2 implemented. Always read as zero, indicating that it is not implemented.	R	0
DSPP	10	MIPS DSP ASE implemented. Always read as zero, indicating that it is not implemented.	R	0
CTXTC	9	*ContextConfig* register is implemented. Always read as zero, indicating that it is not implemented.	R	0
ITL	8	MIPS IFlow Trace mechanism implemented. Always read as zero, indicating that it is not implemented.	R	0
LPA	7	Large physical address for MIPS64. This bit returns zero on read for MIPS32.	R	0
VEIC	6	Supporting an external interrupt controller is implemented Read as zero, indicating that it is not implemented.	R	0
VInt	5	Vectored interrupts implemented. 0: Vectored interrupts are not implemented; 1: Vectored interrupts are implemented;	R	1
SP	4	Small (1KByte) page support is implemented. Always reads as zero, indicating that XBurst2 CPU does not support 1KByte page.	R	0
CDMM	2	Common Device Memory Map (CDMM) implemented. Always read as zero, indicating that it is not implemented.	R	0
MT	2	MIPS MT ASE implemented. Always read as zero, indicating that it is not implemented.	R	0
SM	1	SmartMIPS ASE implemented. Always read as zero, indicating that it is not implemented.	R	0
TL	0	Trace Logic implemented. Read as 0 to indicate Trace Logic is not implemented.	R	0

3.3.1.5 Device Configuration 4-Config 4(CP0 Register16, Select4)

The *Config4* register encodes additional capabilities such as TLBINV instruction support and the number of kernel scratch registers.

Config4 Register

31	30 29	28	27 26 25 24	23 22 21 20 19 18 17 16	15 14	13	12 11 10 9 8	7 6 5 4	3 2 1 0
M	IE	AE	VTLB SizeExt	KscrExist	MMU ExDef	0	FTLB PageSize	FTLB Ways	FTLB Sets

<table>
<tr><th>Name</th><th>Bits</th><th>Description</th><th>R/W</th><th>Reset</th></tr>
<tr><td>M</td><td>31</td><td>Denotes the presence of <i>Config5</i> register.</td><td>R</td><td>1</td></tr>
<tr><td>IE</td><td>30:29</td><td>TLB invalidate instruction configuration.
For XBurst2 CPU, both TLBINV and TLBINVF are supported. TLBINV* instructions operate on entire MMU.</td><td>R</td><td>0x3</td></tr>
<tr><td>AE</td><td>28</td><td>Read as zero, indicating that EntryHI$_{ASID}$ is not extended.</td><td>R</td><td>0</td></tr>
<tr><td>VTLB SizeExt</td><td>27:24</td><td>If Config4MMUExt=3 then this field is concatenated to the left of the most significant bit of the Config1MMUSize field to indicate the size of the VTLB.</td><td>R</td><td>0</td></tr>
<tr><td>KScrExist</td><td>23:16</td><td>Indicates how many scratch registers are available to kernel-mode software within COP0 register 31. Each bit represents a select for Coprocessor0 Register 31. Bit 16 represents Select 0(<i>DESAVE</i> register), Bit 23 represents Select 7. If the bit is set, the associated scratch register is implemented and available for kernel-mode software.</td><td>R</td><td>0xFC</td></tr>
<tr><td>MMU ExDef</td><td>15:14</td><td>MMU Extension Definition. Always read as 3, indicating that boht FTLB and VTLB are supported:
Config4[3:0] indicates FTLB sets.
Config4[7:4] indicates FTLB ways.
Config4[12:8] indicates FTLB page size.
Config4[27:24] used as VTLBSizeExt.</td><td>R</td><td>0x3</td></tr>
<tr><td>0</td><td>13</td><td>Must be written as zero, returns zeros on read.</td><td>R</td><td>0</td></tr>
<tr><td>FTLB PageSize</td><td>12:8</td><td>Indicate the page size of the FTLB array entries. The whole FTLB entries must be flushed before this register field value being changed by software.
<table>
<tr><th>Encoding</th><th>Page Size</th></tr>
<tr><td>00000</td><td>Reserved</td></tr>
<tr><td>00001</td><td>4KB</td></tr>
<tr><td>00010</td><td>16KB</td></tr>
<tr><td>00011-11111</td><td>Reserved</td></tr>
</table></td><td>R/W</td><td>0x01</td></tr>
<tr><td>FTLB</td><td>7:4</td><td>Indicate the set associativity of the FTLB array.</td><td>R</td><td>0x2</td></tr>
</table>

<table>
<tr><th></th><th></th><th></th><th></th><th></th></tr>
<tr><td>Ways</td><td></td><td><table><tr><th>Encoding</th><th>Associativity</th></tr><tr><td>0000-0001</td><td>Reserved</td></tr><tr><td>0010</td><td>4</td></tr><tr><td>0011-1111</td><td>Reserved</td></tr></table></td><td></td><td></td></tr>
<tr><td>FTLB Sets</td><td>3:0</td><td>Indicates the number of sets per way within the FTLB array.<table><tr><th>Encoding</th><th>Set per Way</th></tr><tr><td>0000-0101</td><td>Reserved</td></tr><tr><td>0110</td><td>64</td></tr><tr><td>0111-1111</td><td>Reserved</td></tr></table></td><td>R</td><td>0x6</td></tr>
</table>

3.3.1.6 Device Configuration 5-Config 5(CP0 Register16, Select5)

The *Config5* register encodes additional capabilities for the address mode programming and cache error exceptions.

Config5 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	K	CV	EVA	MSAEn	0																										NFExist

Name	Bits	Description	R/W	Reset
M	31	*Config6* register is not implemented.	R	0
K	30	Segmentation Control is not implemented.	R	0
CV	29	Segmentation Control is not implemented.	R	0
EVA	28	Enhanced Virtual Addressing instructions not implemented	R	0
MSAEn	27	MIPS SIMD architecture (MSA) enable. This bit is encoded as follows: 0: MSA instructions and registers are disabled. Executing an MSA instruction causes a MSA disabled exeption. 1: MSA instructions and registers are enabled.	R/W	0
0	26:1	Must be written as zero; returns zero on read	R	0
NFExist	0	The nested fault feature does not exist.	R	0

3.3.1.7 Processor ID-PRId(CP0 Register 15, Select 0)

The Processor Identification (*PRId*) register is a 32 bit read only register that contains information identifying manufacturer, manufacturing options, processor identification and revision level of the processor.

PRId Register

31 24	23 16	15 8	7 0
0	Company ID	Processor ID	Revision

Name	Bits	Description	R/W	Reset
0	31:24	Must be written as zero; returns zero on read	R	0
Company ID	23:16	Company ID. Identifies the company that designed or manufactured the processor.	R	0x13
Processor ID	15:8	Processor ID. Identifies the uA type of processor. This field allows software to distinguish between the various types of processors. 1) Generation(15~13): 0 - XBurst1; 1 - XBurst2; others - reserved 2) uA(12~8): micro architecture version. 0 – X2000	R	0x20

Revision	7:0	The revision number of mass production.
		1) Process(7~4):
		encode of IC process technology.
		0 - SMIC40;
		1 - TSMC40;
		2 - GF28;
		3 - TSMC28;
		4 - SAMSUNM28;
		5 - TSMC22;
		6 - GF22;
		others - reserved
		2) version(3~0):
		internal version of different implementation.
		Please refer to SOC document.
			R	preset

3.3.1.8 Exception Base Address-EBase(CP0 Register 15, Select 1)

The *EBase* register is a read/write register containing the base address of the exception vectors used when *StatusBEV* equals 0, and a read-only CPU number value that may be used by software to distinguish different cores in a multi-processor system.

The write-gate bit is implemented. This allows exception vectors to be placed anywhere in the address space. To ensure backward compatibility, the write-gate bit must be set before bit 31-30 can be changed.

The size of the ExcBase field depends on the state of the WG bit. At reset, the WG bit is cleared by default. In this case, the ExcBase field is comprised of bits 29:12. Bits 31:30 of the EBase Register are not writeable and are forced to a value of 2'b10 by hardware so that the exception handler will be executed from the *kseg0/kseg1* segments.

The addition of the base address and the exception offset is done inhibiting a carry between bit 29 and bit 30 of the final exception address. The combination of these two restrictions forces the final exception address to be in the kseg0 or kseg1 unmapped virtual address segments.

If the value of the Ebase base register is to be changed, this must be done with *Status.BEV* equal to 1. The operation result is UNDEFINED if the Ebase is written with a different value when *Status.BEV* is 0.

Ebase Register

31 30 29 12	11	10	9 0
Exception Base	WG	0	CPUNum

Name	Bits	Description	R/W	Reset
Exception Base	31:12	This field specifies the base address of the exception vectors when StatusBEV is zero. Note that bits 31~30 can merely be overwritten when WG is set.	R/W	0x80_000
WG	11	Write Gate for bits 31..30. 0: Bits 31~30 are unchanged when write to EBase 1: Bits 31~30 can be modified by writing to EBase	R/W	0
0	10	Must be written as zero, returns zero on read.	R	0
CPUNum	9:0	This field specifies the serial number of the CPU core in a multi-processor system and can be used by software to distinguish a particular core from the others. This field can also be read via RDHWR register 0.	R	Preset

3.3.1.9 Status (CP0 Register 12, Select 0)

The Status register is a read/write register that contains the operating mode, interrupt enabling and the diagnostic states of the processor.

Status Register

31 28	27	26	25	24	23	22	21	20	19	18	17 16	15 8	7 5	4	3	2	1	0
CU3-CU0	RP	FR	RE	MX	0	BEV	TS	SR	NMI	ASE	0	IM[7:0]	0	UM	R0	ERL	EXL	IE

Name	Bits	Description	R/W	Reset
CU3	31	Coprocessor 3 Usable. Hardwired to zero.	R	0
CU2	30	Coprocessor 2 Usable. 0: Access not allowed 1: Access allowed CU2 is used for MXA (Ingenic's dedicated SIMD128 ISA)	R/W	0
CU1	29	Coprocessor 1 Usable. 0: Access not allowed. 1: Access allowed. CU1 is used for a floating-point unit.	R/W	0
CU0	28	Coprocessor 0 Usable. 0: Access not allowed. 1: Access allowed. Coprocessor 0 is always usable when the core is running in Kernel Mode or Debug Mode, independent of the state of the CU0 bit.	R/W	0
RP	27	Enables reduced power mode implemented. Always read as zero, indicating that it is not implemented.	R	0
FR	26	Floating-point register mode for 64-bit point unit. 0: only 32-bit datatype can be hold in one floating-point register, 64-bit datatypes are stored in even-odd pairs of registers. 1: Both 32-bit and 64-bit datatypes can be hold	R	1
RE	25	Reverse Endian. Hardwired to zero as this feature is not implemented.	R	0
MX	24	MIPS DSP extension. Hardwired to zero as this feature is not implemented.	R	0
0	23	Must be written as zero; returns zero on read.	R	0
BEV	22	Control the location of exception vectors. 0: Normal 1: Bootstrap See Exception Vector Locations for detail	R/W	1

TS	21	Indicates that the TLB has detected a match on multiple entries. This detection occurs only on a write to the VTLB. When such a detection occurs, the core initiates a machine check exception and sets this bit. If the condition can be corrected, this bit should be cleared by software before resuming normal operation.	R/W	0
SR	20	Soft Reset. Hardwired to zero as this feature is not implemented.	R	0
NMI	19	NMI is not implemented, write is ignored and read as zero.	R	0
ASE	18	ASE is not implemented, write is ignored and read as zero.	R	0
0	17:16	Must be written as zero; returns zero on read	R	0
IM[7:0]	15:10	Interrupt Mask. Controls the enabling of each of the hardware interrupts. 0: Interrupt request disabled 1: Interrupt request enabled Since *Config3.VEIC* is hardwired to '0', indicating that external interrupt controller mode is not supported, so IM[7:0] is only used for interrupt mask.	R/W	0x0
IM1...IM0	9:8	Interrupt Mask: Controls the enabling of each of the software interrupts. 0: Interrupt request disabled. 1: Interrupt request enabled.	R/W	0x0
0	7:5	Must be written as zero; returns zero on read	R	0
UM	4	The encoding for this bit is : 0: Base mode is Kernel Mode. 1: Base mode is User Mode.	R/W	0
R0	3	Supervisor mode is not implemented, write is ignored and read as zero.	R	0
ERL	2	Error Level, can be set by hardware when a reset exception is taken or by executing MTC0 $12, 0. 0: normal level; 1: error level. When ERL is set: ● The core is running in kernel mode ● Hardware and software interrupts are disabled ● The ERET instruction will use the return address held in Error EPC instead of EPC. ● Segment kuseg is treated as an unmapped and uncached region.	R/W	1
EXL	1	Exception Level, can be set by hardware when any exception other than reset exception is taken or by executing MTC0 $12, 0.	R/W	0

		0: normal level; 1: exception level; When EXL is set: ● The core is running in kernel mode ● Hardware and software interrupts are disabled ● TLB Refill exceptions use the general exception vector instead of the TLB Refill vector. ● EPC Cause$_{BD}$ and SRSCtl will not be updated if another exception is taken		
IE	0	Global Interrupt Enable: 0: interrupts disabled 1: interrupts enabled Note that IE being cleared just prohibit acknowledging interrupts by executing IRQ handler, it can't prevent some hardware behaviors like captured interrupt signals waking the core from SLEEP state.	R/W	0

3.3.1.10 Interrupt Control - IntCtl (CP0 Register 12, Select 1)

The *IntCtl* register controls the expanded interrupt capabilities if vectored interrupt and/or external interrupt controller.

IntCtl Register

31 29	28 26	25 23	22 10	9 5	4 0
IPTI	IPPCI	IPFDC	0	VS	0

<table>
<tr><th>Name</th><th>Bits</th><th>Description</th><th>R/W</th><th>Reset</th></tr>
<tr><td>IPTI</td><td>31:29</td><td>This field specifies the IP number to which the Timer Interrupt request is merged.
Write is ignored and read as 0x7.</td><td>R</td><td>0x7</td></tr>
<tr><td>IPPCI</td><td>28:26</td><td>This field specifies the IP number to which the Performance Counter Interrupt request is merged.
Write is ignored and read as 0x6.</td><td>R</td><td>0x6</td></tr>
<tr><td>IPFDC</td><td>25:23</td><td>Write is ignored and read as zero because Fast Debug Channel Interrupt request is not implemented.</td><td>R</td><td>0x0</td></tr>
<tr><td>0</td><td>22:10</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
<tr><td>VS</td><td>9:5</td><td>Vector spacing. If vectored interrupts are implemented (as denoted by <i>Config3.VEIC or Config3.VINT),</i> this field specifies the spacing between vectored interrupts.
<table>
<tr><th>VS Field Encoding</th><th>Spacing Between Vectors(hex)</th><th>Spacing Between Vectors(decimal)</th></tr>
<tr><td>0x00</td><td>0x000</td><td>0</td></tr>
<tr><td>0x01</td><td>0x020</td><td>32</td></tr>
<tr><td>0x02</td><td>0x040</td><td>64</td></tr>
<tr><td>0x04</td><td>0x080</td><td>128</td></tr>
<tr><td>0x08</td><td>0x100</td><td>256</td></tr>
<tr><td>0x10</td><td>0x200</td><td>512</td></tr>
</table></td><td>R/W</td><td>0</td></tr>
<tr><td>0</td><td>4:0</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
</table>

Encoding of IPTI, IPPCI Fields

Encoding	IP bit	Hardware Interrupt Source
2	2	HW0 (INTC IRQ)
3	3	HW1 (SMP MAILBOX IRQ)
4	4	HW2 (OST IRQ)
5	5	HW3 (Reserved)
6	6	HW4(IPPCI)
7	7	HW5 (IPTI)

3.3.2 TLB Management Registers

This section contains the following TLB management registers.

- Index Register (CP0 Register 0, Select 0)
- Random Register (CP0 Register 1, Select 0)
- EntryLo0, EntryLo1 Register (CP0 Register 2, 3, Select 0)
- EntryHi Register (CP0 Register 10, Select 0)
- Context Register (CP0 Register 4, Select 0)
- PageMask Register (CP0 Register 5, Select 0)
- Page Granularity-PageGrain (CP0 Register 5, Select 1)
- Wired Register (CP0 Register 6, Select 0)
- BadVAddr Register (CP0 Register 8, Select 0)

3.3.2.1 Index Register (CP0 Register 0, Select 0)

The index register contains the index used to access the TLB for TLBP, TLBR, and TLBWI instructions. The width of the index field is 9 which is equal to Ceiling(log2(VTLBSize+FTLBSize)).

Index Register

<table>
<tr><td>31</td><td>30</td><td>8</td><td>0</td></tr>
<tr><td>P</td><td>Reserved</td><td colspan="2">Index</td></tr>
</table>

<table>
<tr><th>Name</th><th>Bits</th><th>Description</th><th>R/W</th><th>Reset</th></tr>
<tr><td>P</td><td>31</td><td>Probe Failure. Hardware writes this bit after executing TLBP instruction to indicate whether a TLB match occurred:
<table>
<tr><th>Encoding</th><th>Meaning</th></tr>
<tr><td>0</td><td>A match occurred, and the Index field contains the index of the matching entry</td></tr>
<tr><td>1</td><td>No match occurred and the Index field is UNPREDICTABLE</td></tr>
</table></td><td>R</td><td>0</td></tr>
<tr><td>Reserved</td><td>30:9</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
<tr><td>Index</td><td>8:0</td><td>TLB index.
Software writes this field to provide the index of the TLB entry referenced by the TLBR and TLBWI instructions.
Hardware writes this field with the index of the matching TLB entry after executing of the TLBP instruction. If the TLBP fails, the contents of this field are UNPREDICTABLE.</td><td>R/W</td><td>0</td></tr>
</table>

3.3.2.2 Random Register (CP0 Register 1, Select 0)

The *Random* register is a read-only register，it is used to index the VTLB during a TLBWR instruction. The value of the register varies between an upper and lower bound as follow:

- A lower bound is set by the number of VTLB entries reserved for exclusive use by the operating system (the contents of the *Wired* register). The entry indexed by the *Wired* register is the first entry available to be written by a TLBWR.
- An upper bond is set by the total number of VTLB entries minus 1.

Within the required constraints of the upper and lower bounds, the manner in which the processor selects values for the *Random* register is implementation-dependent.

The processor initializes the *Random* register to the upper bound when reset and when the *Wired* register is updated.

Random Register

31 5	4 0
Reserved	Random

Name	Bits	Description	R/W	Reset
Reserved	31:5	Must be written as zero; returns zero on read	R	0
Random	4:0	TLB Random Index	R	0x1f

3.3.2.3 EntryLo0, EntryLo1 Register (CP0 Register 2, 3, Select 0)

The pair of *EntryLo* registers acts as the interface between the TLB and the TLBR, TLBWI, and TLBWR instructions. The contents of the *EntryLo0* and *EntryLo1* registers are undefined after an address error, TLB invalid, TLB modified, or TLB refill exceptions.

EntryLo0, EntryLo1 Register

31	30	29 26	25 6	5 3	2	1	0
RI	XI	0	PFN	C	D	V	G

Name	Bits	Description	R/W	Reset
RI	31	Read Inhibit. If this bit is set, an attempt to read data from the page will trigger a TLB Invalid exception, even if the *V*(Valid) bit is set.	R/W	0
XI	30	Execute Inhibit. If this bit is set, an attempt to fetch from the page will trigger a TLB Invalid exception, even if the *V* (Valid) bit is set.	R/W	0
0	29:26	Must be written as zero; returns zero on read	R	0
PFN	25:6	Page Frame Number. The PFN field corresponds to bits 31..12 of the physical address.	R/W	0
C	5:3	Cache attribute of the page. See Table 6-2 Cache Coherency Attributes for detail.	R/W	0
D	2	Dirty attribute of the page. The "Dirty" flag indicates that the page has been written, and/or is writable. If D has been set, stores to the page are permitted. However, if D has been cleared, stores to the page cause a TLB Modified exception.	R/W	0
V	1	Valid attribute of the page. If this bit is a zero, accesses to the page cause a TLB *Invalid* exception. This bit can make just one of a pair of pages be valid	R/W	0
G	0	Global attribute of the page. The "Global" bit. On a TLB entry update, the logical AND result of the G bits in both the *EntryLo0* and *EntryLo1* registers becomes the G bit to be filled in Entry0/Entry1 TLB entry. If the TLB entry G bit is a one, then the ASID comparisons are ignored during TLB matches.	R/W	0

3.3.2.4 EntryHi Register (CP0 Register 10, Select 0)

The *EntryHi* register contains the virtual address match information used for TLB read and write operations.

A TLB exception (TLB Refill, TLB Invalid, or TLB Modified) causes bits $VA_{31:13}$ of the virtual address to be written into the VPN2 field of the *EntryHi* register. The ASID field is written by software with the current address space identifier value and is used during the TLB comparison process to determine TLB match.

The VPN2 field of the *EntryHi* register is not defined after an address error exception occurs.

EntryHi Register

31 … 13	12 11	10	9 8	7 6 5 4 3 2 1 0
VPN2	VPN2X	EHINV	ASIDX	ASID

Name	Bits	Description	R/W	Reset
VPN2	31:13	$VA_{31:13}$ of the virtual address. This field is written by hardware on a TLB exception or on a TLB read, and can be written by software before a TLB write.	R/W	0
VPN2X	12：11	XBurst®2 CPU does not support 1KB pages and returns zero on read.	R	0
EHINV	10	TLB HW Invalidate. If this bit is set, the TLBWI instruction will invalidate the VPN2 field of the selected TLB entry. A TLBR instruction will update this field with the VPN2 invalid bit of the read TLB entry.	R/W	0
ASIDX	9:8	Write is ignored and read as zero.	R	0
ASID	7:0	Address space identifier. This field is written by hardware on a TLBR operation, and can be written by software to set the current process' ASID.	R/W	0

3.3.2.5 Context Register (CP0 Register 4, Select 0)

The *Context* register is a read/write register containing a pointer to an entry in the page table entry (PTE) array. PTE array is an operating system data structure that stores virtual-to-physical address translations. During a TLB miss, the operating system loads the TLB with the missing translation inforamtion from the PTE array. The *Contex* register duplicates some of the information provided in the *BadVAddr* register.

A TLB exception (TLB Refill, TLB Invalid, or TLB Modified) causes bits VA31:13 of the virtual address to be written into the *BadVPN2* field of the *Context* register. The *PTEBase* field is written only by software and used by the operating system.

The *BadVPN2* field of the *Context* register is not defined after an address error exception occurs.

Context register

31 23	22 4	3 0
PTEBase	BadVPN2	Reserved

Name	Bits	Description	R/W	Reset
PTEBase	31:23	This field is normally written with a value that allows the operating system to use the *Context* Register as a pointer into the current PTE array in memory.	R/W	0
BadVPN2	22:4	This field is written only by hardware on a TLB exception. It contains bits VA [31:13] of the virtual address that cause the exception.	R	0
Reserved	3:0	Must be written as zero; returns zero on read	R	0

3.3.2.6 PageMask Register (CP0 Register 5, Select 0)

The *PageMask* register is a read/write register used for reading from and writing to the TLB. It holds a comparison mask that sets the variable page size for each TLB entry.

PageMask Register

31 27	26 13	12 0
Reserved	Mask	Reserved

Name	Bits	Description	R/W	Reset
Reserved	31:27	Must be written as zero; returns zero on read	R	0
Mask	26:13	Mask bits for varying page size. 00_0000_0000_0000: 4KB 00_0000_0000_0011: 16KB 00_0000_0000_1111: 64KB 00_0000_0011_1111: 256KB 00_0000_1111_1111: 1MB 00_0011_1111_1111: 4MB 00_1111_1111_1111: 16MB 11_1111_1111_1111: 64MB	R/W	0x0
Reserved	12:0	Must be written as zero; returns zero on read	R	0

3.3.2.7 Page Granularity-PageGrain (CP0 Register 5, Select 1)

The *PageGrain* register is a read/write register used for enabling 1KB page support and further implementing the XI/RI mechanism.

PageGrain Register

31	30	29	28	27	26 13	12 8	7 0
RIE	XIE	0	ESP	IEC	0	ASE	0

Name	Bits	Description	R/W	Reset
RIE	31	Read Inhibit Enable. Read always as 1, indicates RI bit of the *EntryLo0* and *EntryLo1* registers is enabled.	R	1
XIE	30	Execute Inhibit Enable. Read always as 1, indicates XI bit of the *EntryLo0* and *EntryLo1* registers is enabled.	R	1
Reserved	29	Must be written as zero; returns zero on read	R	0
ESP	28	Always read as 0 since 1KB page is not supported.	R	0
IEC	27	Enable unique exception code for the Read-Inhibit and Execute-Inhibit exceptions. 0: Read-Inhibit and Execute-Inhibit exceptions both use the TLBL exception code. 1: Read-Inhibit exceptions use the TLBRI exception code. Execute-Inhibit exceptions use the TLBXI exception code.	R	0
Reserved	26:13	Must be written as zero; returns zero on read	R	0
ASE	12:8	Ignored on write; return zero on read.	R	0
Reserved	7:0	Must be written as zero; returns zero on read	R	0

3.3.2.8 Wired Register (CP0 Register 6, Select 0)

The *Wired* register is a read/write register that specifies the lower bound to which the *Random* register can reach. In another word, after non-zero value is set to the *Wired* register, VTLB entries from entry(0) to entry(Wired) then can not be overwritten by a TLBWR instruction.

The *Wired* register is reset to zero. Writing the *Wired* register will cause the *Random* register to reset to its upper bound.

The operation of the processor is UNDEFINED if a value greater than or equal to the number of VTLB entries is written to the *Wired* register.

Wired Register

31 5	4 0
Reserved	Wired

Name	Bits	Description	R/W	Reset
Reserved	31:5	Must be written as zero; returns zero on read	R	0
Wired	4:0	TLB wired boundary	R/W	0x0

3.3.2.9 BadVAddr Register (CP0 Register 8, Select 0)

The *BadVAddr* register is a read only register that captures the most recent virtual address causing one of the following exceptions:

- Address error (AdEL or AdES)
- TLB Refill
- TLB Invalid
- TLB Modified

The *BadVAddr* register does not capture address information for cache or bus errors.

BadVAddr Register

31	0
BadVAddr	

Name	Bits	Description	R/W	Reset
BadVAddr	31:0	Failed virtual address in Address Error or TLB exceptions.	R	0x0

3.3.3 Exception Control Registers

This section contains the following exception control registers.

- Cause Register (CP0 Register 13, Select 0)
- Exception Program Counter (CP0 Register 14, Select 0)
- ErrorEPC Register (CP0 Register 30, Select 0)

3.3.3.1 Cause Register (CP0 Register 13, Select 0)

The *Cause* register primarily describes the cause of the most recent exception. In addition, it contains other fieldes with dedicated purposes like controlling interrupt's pattern of acknowledgement.

Cause Register

31	30	29 28	27	26	25 24	23	22	21	20 18	17 16	15 10	9 8	7	6 2	1 0
BD	TI	CE	DC	PCI	ASE	IV	WP	FDCI	0	ASE	IP[7:2]	IP[1:0]	0	Exc Code	0

Name	Bits	Description	R/W	Reset
BD	31	Indicates whether the last exception taken occured in a branch delay slot. 0: Not in delay slot 1: In delay slot The core can update BD only if Status$_{EXL}$ was zero when an exception occurred.	R	0
TI	30	Indicates whether a timer interrupt is pending. 0: No timer interrupt is pending. 1: Timer interrupt is pending.	R	0
CE	29:28	Coprocessor unit number referenced when a Coprocessor unusable exception is taken. 00: Coprocessor 0 (CP0) 01: Coprocessor 1 (FPU) 10: Coprocessor 2 (MXA) 11: Coprocessor 3 (Reserved)	R	0
DC	27	Disable *Count* register. 0:Enable counting of *Count* register 1:Disable counting of *Count* register	R/W	1
PCI	26	Denotes whether a Performance Counter interrupt is pending. 0: No performance counter interrupt is pending. 1: Performance counter interrupt is pending.	R	0
ASE	25:24	These bits are reserved because MCU ASE is not implemented, read as zero and write is ignored.	R	0
IV	23	Indicates whether an interrupt exception uses the general	R/W	0

<table>
<tr><td></td><td></td><td>exception vector or a special interrupt vector.
0:Use the general exception vector(0x180)
1:Use the special interrupt vector(0x200)</td><td></td><td></td></tr>
<tr><td>WP</td><td>22</td><td>Indicates that the watch exception was deferred because Status_{EXL} or Status_{ERL} were a one at the time the watch exception was detected. This bit both indicates that the watch exception was deferred, and causes the exception to be initiated once Status_{EXL} and Status_{ERL} are both zero. As such, software must clear this bit as part of the watch exception handler to prevent a watch exception loop. Software should not write a 1 to this bit when its value is a 0, thereby causing a 0-to-1 transition.</td><td>R/W</td><td>0</td></tr>
<tr><td>FDCI</td><td>21</td><td>Fast Debug Channel Interrupt.
Always read as 0 as FDC is not implemented.</td><td>R</td><td>0</td></tr>
<tr><td>0</td><td>20:18</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
<tr><td>ASE</td><td>17:16</td><td>These bits are reserved because MCU ASE is not implemented, write is ignored and read as zero.</td><td>R</td><td>0</td></tr>
<tr><td>IP[7:2]</td><td>15:10</td><td>Indicates pending interrupts.
<table>
<tr><th>Bit</th><th>Name</th><th>Meaning</th></tr>
<tr><td>15</td><td>IP7</td><td>Hardware interrupt 5</td></tr>
<tr><td>14</td><td>IP6</td><td>Hardware interrupt 4</td></tr>
<tr><td>13</td><td>IP5</td><td>Hardware interrupt 3</td></tr>
<tr><td>12</td><td>IP4</td><td>Hardware interrupt 2</td></tr>
<tr><td>11</td><td>IP3</td><td>Hardware interrupt 1</td></tr>
<tr><td>10</td><td>IP2</td><td>Hardware interrupt 0</td></tr>
</table>
IP[7:2] are connected with IRQ sources as follows:
● IP7, Timer Interrupt request
● IP6, Performance Counter Interrupt request
● IP5, reserved
● IP4, OST Interrupt request
● IP3, Mailbox Interrupt request
● IP2, INTC Interrupt request</td><td>R</td><td>0</td></tr>
<tr><td>IP[1:0]</td><td>9:8</td><td>Controls the request for software interrupt.
<table>
<tr><th>Bit</th><th>Name</th><th>Meaning</th></tr>
<tr><td>9</td><td>IP1</td><td>Request of SW interrupt 1</td></tr>
<tr><td>8</td><td>IP0</td><td>Request of SW interrupt 0</td></tr>
</table></td><td>R/W</td><td>0</td></tr>
<tr><td>0</td><td>7</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
<tr><td>Exc Code</td><td>6:2</td><td>Exception code. See Table 3.4</td><td>R</td><td>0</td></tr>
<tr><td>0</td><td>1:0</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
</table>

Table 3-4 Cause Register ExcCode Field Descriptions

Exception	Mnemonic	Description

Code Value		
0	Int	Interrupt.
1	Mod	TLB modification exception.
2	TLBL	TLB exception. (load or instruction fetch)
3	TLBS	TLB exception. (store)
4	AdEL	Address error exception. (load or instruction fetch)
5	AdES	Address error exception. (store)
6	N/A	--
7	N/A	--
8	Sys	Syscall exception.
9	Bp	Breakpoint exception.
10	RI	Reserved instruction exception.
11	CPU	Coprocessor unusable exception.
12	Ov	Integer overflow exception.
13	Tr	Trap exception.
14	MSAFPE	MSA Floating-Point exception
15	FPE	Floating point exception.
16-20	N/A	--
~~19~~	~~TLBRI~~	~~TLB Read-Inhibit exception~~
~~20~~	~~TLBXI~~	~~TLB Execution-Inhibit exception~~
21	MSADis	MSA Disabled exception
22	N/A	--
23	WATCH	Reference to WatchHi/WatchLo address.
24	Mcheck	Machine Check.
25-31	-	Reserved.

3.3.3.2 Exception Program Counter (CP0 Register 14, Select 0)

The Exception Program Counter (*EPC*) is a read/write register that contains the address at which a process resumes after an exception has been serviced. All bits of the *EPC* register are significant and writable.

Unless the *EXL* bit in the *Status* register is already a 1, the core writes the *EPC* register when an exception occurs.

For synchronous (precise) exceptions, the *EPC* contains one of the following:

- The program counter of the instruction that is the direct cause of the exception.
- The program counter of the preceding branch or jump instruction that is the adjacent one of the exception-causing instruction in the branch delay slot, and therefore the Branch Delay bit in the *Cause* register is set.

Note that the core does not update the *EPC* register when the EXL bit in the *Status* register has been set to one for a new exception. Moreover, the register can be modified via the MTC0 instruction.

Exception Program Counter

31	0
EPC	

Name	Bits	Description	R/W	Reset
EPC	31:0	Exception Program Counter.	R/W	0

3.3.3.3 ErrorEPC Register (CP0 Register 30, Select 0)

The *ErrorEPC* register is a read-write register, similar to the *EPC* register, at which program resumes after a Reset exception. All bits of the *ErrorEPC* register are significant and writable.

ErrorEPC Register

31	0
ErrorEPC	

Name	Bits	Description	R/W	Reset
ErrorEPC	31:0	Error Exception Program Counter.	R/W	undef

3.3.4 Timer Registers

This section contains the following timer registers.

- Count (CP0 Register 9, Select 0)
- Compare (CP0 Register 11, Select 0)

3.3.4.1 Count (CP0 Register 9, Select 0)

The *Count* register acts as a timer, incrementing at a constant rate, that is, when enabled by clearing the *DC* bit in the *Cause* register, the counter increments every clock cycle. The *Count* field starts counting from whatever value being pre-loaded into it, and it wraps back to zero when reaching maximum value 0xFFFF_FFFF.

Count Register

31 … 0
Count

Name	Bits	Description	R/W	Reset
Count	31:0	Interval counter	R/W	0

3.3.4.2 Compare (CP0 Register 11, Select 0)

The *Compare* register acts in conjunction with the *Count* register to implement a timer and timer interrupt function. When the value of the *Count* register equals the value of the *Compare* register, a timer interrupt arises. This IRQ request is then conntected with hardware interrupt signal pin number 5 in order to set interrupt bit *IP(7)* in the *Cause* register.

For diagnostic purposes, the *Compare* register is a read/write register. In normal use, however, the *Compare* register is write-only. As a side effect, writing to this register clears the timer intrerrupt.

Compare Register

31 … 0
Compare

Name	Bits	Description	R/W	Reset
Compare	31:0	Interval count compare value	R/W	0

3.3.5 Cache Management Registers

This section contains the following registers.

- TagLo Register (CP0 Register 28, Select 0)
- DataLo Register (CP0 number 28, Select 1)

3.3.5.1 TagLo Register (CP0 Register 28, Select 0)

The *TagLo* register acts as the interface to the L1-cache tag array. The Index Store Tag and Index Load Tag operations of the CACHE instruction for I-Cache and D-cache use the *TagLo* register as the source and destination of tag information, respectively.

However, software must be able to write zeros into the *TagLo* register and then use the Index Store Tag cache operation to initialize the cache tags of I-Cache and D-cache to a valid state at powerup.

TagLo Register

31 12	11 1	0
PTagLo	0	V

Name	Bits	Description	R/W	Reset
PTagLo	31:12	Physical address of the indexed cache line.	R/W	0
0	11:1	Must be written as zero; returns zero on read	R	0
V	0	Valid bit of the cache line.	R/W	0

3.3.5.2 DataLo Register (CP0 number 28, Select 1)

The *DataLo* register acts as the interface to the L1-cache data array. The Index Load Tag operation of the CACHE instruction reads the corresponding data values into the *DataLo* register.

DataLo Register

31 0
DataLo

Name	Bits	Description	R/W	Reset
DataLo	31:0	Low-order data read from cache.	R/W	0

3.3.6 Thread Context and Shadow Control Registers

The shadow Register Set Control (SRSCtl) register and Shadow Register Set Map(SRSMap) register are defined to allow software to read this register to determine that shadow registers are not implemented.

- SRSCtl Register (CP0 Register 12, Select 2)
- SRSMap Register (CP0 Register 12, Select 3)

3.3.6.1 SRSCtl Register (CP0 Register 12, Select 2)

The *SRSCtl* register controls the operation of GPR shadow sets in the processor.

SRSCtl Register

31 30	29 26	25 22	21 18	17 16	15 12	11 10	9 6	5 4	3 0
0	HSS	0	EICSS	0	ESS	0	PSS	0	CSS

Name	Bits	Description	R/W	Reset
Reserved	31:30	Must be written as zero; returns zero on read	R	0
HSS	29:26	Highest Shadow Set. Read as zero since only the normal GPRs are implemented.	R	0
Reserved	25:22	Must be written as zero; returns zero on read	R	0
EICSS	21:18	EIC interrupt mode shadow set. Because Config3$_{VEIC}$=0, this field must be written as zero, and returns zero on read.	R	0
Reserved	17:16	Must be written as zero; returns zero on read	R	0
ESS	15:12	Exception Shadow Set. Must be written as zero, and returns zero on read.	R	0
Reserved	11:10	Must be written as zero; returns zero on read	R	0
PSS	9:6	Previous Shadow Set. Must be written as zero, and returns zero on read.	R	0
Reserved	5:4	Must be written as zero; returns zero on read	R	0
CSS	3:0	Current Shadow Set. Must be written as zero, and returns zero on read.	R	0

3.3.6.2 SRSMap Register (CP0 Register 12, Select 3)

The *SRSMap* register is a 32-bit read-only register.

SRSMap Register

31 30 29 26 25 22 21 18 17 16 15 12 11 10 9 6 5 4 3 0
SRSMap

Name	Bits	Description	R/W	Reset
SRSMap	31:0		R	0

3.3.7 CPU Performance Monitor Registers

The performance counters provide the capability to count events or cycles for use in performance analysis. Each performance counter consists of a pair of registers: a 32-bit control register and a 32-bit counter register.

Performance counters can be configured to count events or cycles under a specified set of conditions that are determined by the control register for the performance counter. The counter register increments once for each enabled event. When the most significant bit of the counter register is a one (the counter overflows), the performance counter optionally arise an interrupt request. Pending interrupts from all performance counters beging ORed together become the PCI bit in the *Cause* register. Counting continues after a counter register has overflowed despite of an PCI interrupt being pended or taken.

Each performance counter is mapped into even-odd select values of the *PerfCnt* register: Even selects accessing the control register and odd selects accessing the counter register. Table below shows two performance counters implemented and how they map into the select values of the *PerfCnt* register.

Table 3-5 Example Performance Counter Usage of the PerfCnt CP0 Register

Performance Counter	PerfCnt Register Select Value	PerfCnt Register Usage
0	PerfCnt, Select 0	Control Register 0
	PerfCnt, Select 1	Counter Register0
1	PerfCnt, Select 2	Control Register 1
	PerfCnt, Select 3	Counter Register1

3.3.7.1 Performance Counter Control Register (CP0 Register 25, Select 0 or 2)

Performance Counter Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	0																				Event						IE	U	S	K	EXL

Name	Bits	Description	R/W	Reset
M	31	If this bit is a one, indicating that another pair of Performance Counter Control and Counter registers is implemented with a MTC0 or MFC0 select field value of ‘n+2’ and ‘n+3’.	R	1
0	30:11	Must be written as zero; returns zero on read.	R	0
Event	10:5	Selects the event to be counted by the corresponding Counter Register. See following chapter “Performance Counter Events and Codes” for detail.	R/W	0
IE	4	Performance Counter Interrupt Enable. 0: Performance counter interrupt disabled. 1: Performance counter interrupt enabled.	R/W	0

U	3	Enables event counting in User Mode. 0: Disable event counting in User Mode. 1: Enable event counting in User Mode.	R/W	0
S	2	Must be written as zero; returns zero on read.	R	0
K	1	Enables event counting in Kernel Mode. NOTE: this bit enables event counting only when both EXL and ERL in the *Status* register are zero. 0: Disable event counting in Kernel Mode. 1: Enable event counting in Kernel Mode.	R/W	0
EXL	0	Enables event counting when the EXL bit in the *Status* registers is one and the ERL bit in the *Status* register is zero. 0: Disable event counting while EXL=1,ERL=0 1: Enable event counting while EXL=1,ERL=0 Counting is never enabled when the ERL bit in the *Status* register or the DM bit in the *Debug* register is one.	R/W	0

3.3.7.2 Performance Counter Counter Register (CP0 Register 25, Select 1 or 3)

Performance Counter Counter0 Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Event Count																															

Name	Bits	Description	R/W	Reset
Event Count	31:0	Increments once for each event that is enabled by the corresponding Control Register. When the most significant bit is one, a pending interrupt request is ORed with those from other performance counters and indicated by the PCI bit in the *Cause* register	R/W	0

3.3.7.3 Performance Counter Events and Codes

Event Number	Counter0	Counter1
0	Core clock cycles	
1	Instructions graduated	
2	Branch instructions graduated	False branch predictions
3	Cycles that pipe stalled caused by IU	Cycles that pipe stalled caused by LSU
4	Occurred Data Cache miss latency by load	Occurred Data Cache miss event by load
5	Occurred Data Cache miss latency by store	Occurred Data Cache miss event by store
6	reserved	Reserved
7	Occurred Instruction Cache miss latency	Occurred Instruction Cache miss event
Others	Reserved	Reserved

3.3.8 Debug Registers

This section contains the following dedicated debug purpose hardware registers.

- Debug Register (CP0 Register 23, Select 0)
- Debug2 Register (CP0 Register 23, Select 6)
- Debug Exception Program Counter Register (CP0 Register 24, Select 0)
- Debug Save Register (CP0 Register 31, Select 0)
- WatchLo0 Register (CP0 Register 18, Select 0)
- WatchHi0 Register (CP0 Register 19, Select 0)

3.3.8.1 Debug Register (CP0 Register 23, Select 0)

The *Debug* register is used to control the debug exception and provide information about the cause of the debug exception and when re-entering at the debug exception vector due to a normal exception in debug mode. The R (read only) field information bits are updated by hardware every time the debug exception is taken or when a normal exception is taken when already in debug mode.

Some of the bit fields are only updated on debug exceptions and/or exceptions in debug mode, as shown below:

– DSS, DBp, DDBL, DDBS, DIB, DINT are updated on both debug exceptions and on exceptions in debug modes.

– DExcCode is updated on exceptions in debug mode, and is undefined after a debug exception.

– Halt and Doze are updated on a debug exception, and is undefined after an exception in debug mode.

– DBD is updated on both debug and on exceptions in debug modes.

Debug Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DBD	DM	NoDCR	LSNM	Doze	Halt	CountDM	IBusEP	MCheckP	CacheEP	DBusEP	IEXI	DDBSImp	DDBLImp	EJTAG Ver			DExcCode					NoSSt	SSt	OffLine	DIBImpr	DINT	DIB	DDBS	DDBL	DBp	DSS

Name	Bits	Description	R/W	Reset
DBD	31	Indicates whether the most recent debug exception or exception in Debug Mode occurred in a branch or jump delay slot. 0: not in delay slot; 1: in delay slot	R	0
DM	30	Indicates that the core is operating in debug mode: 0: not running in debug mode 1: running in debug mode	R	0
NoDCR	29	Indicates whether the dseg segment is present: Read as zero, indicating that dseg segment is present.	R	0

LSNM	28	Controls access of load/store between dseg and remaining memory when the dseg segment is present: 0: Load/store in dseg range goes to dseg. 1: Load/store in dseg range goes to main memory.	R/W	0
Doze	27	Read as zero, indicating that the feature is not implemented.	R	0
Halt	26	Read as zero, indicating that the feature is not implemented.	R	0
CountDM	25	The *Count* register behavior in Debug Mode. 1: Count register is normally running in Debug Mode.	R	1
IBusEP	24	In Debug Mode, Bus error exception not applies to a Debug Mode Bus Error exception, this bit is read-only(R) and read as zero.	R	0
MCheckP	23	In Debug Mode, a Machine Check exception not applies to a Debug Mode Machine Check exception, and this bit is read-only(R) and reads as zero.	R	0
CacheEP	22	In Debug Mode, a Cache Error exception not applies to a Debug Mode Cache Error exception. This bit is read-only(R) and read as zero.	R	0
DBusEP	21	In Debug Mode, a Bus Error exception not applies to a Debug Mode Cache Error exception. This bit is read-only(R) and read as zero.	R	0
IEXI	20	An imprecise Error Exception Inhibit (IEXI) is not implemented. This bit is read-only(R) and reads as zero.	R	0
DDBSImpr	19	A Debug Data Break Store Imprecise exception is not implemented, this bit read as zero.	R	0
DDBLImpr	18	A Debug Data Break Load Imprecise exception is not implemented, this bit read as zero.	R	0
EJTAGVer	17:15	JTAG version. 0: Version 2.0	R	0
DExcCode	14:10	Indicates the cause of the latest exception in debug mode. The field is encoded as the ExcCode field in the Cause register for those exceptions that can occur in Debug Mode, with addition of code 30 with the mnemonic CacheErr for cache errors and the use of code 9 with mnemonic Bp for the SDBBP instruction.	R	0
NoSSt	9	Read always as zero, indicating that Single step feature is available.	R	0
SSt	8	Controls whether single-step feature is enabled: 0: No debug single-step exception enabled. 1: Debug single step exception enabled.	R/W	0
OffLine	7	MIPS MT processors is not implemented, this bit is	R	0

		read-only(R) and reads as zero.		
DIBImpr	6	A Debug Instruction Break Imprecise exception is not implemented, this bit reads as zero.	R	0
DINT	5	Indicates that a debug interrupt exception occured. 0: No debug interrupt exception occured 1: Debug interrupt exception occurred	R	0
DIB	4	Indicates that a debug instruction break exception occured. 0: No debug instruction exception occured 1: Debug instruction exception occurred	R	0
DDBS	3	Indicates that a debug data break exception occured on a store. 0: No debug data exception on a store occured 1: Debug data exception on a store occurred	R	0
DDBL	2	Indicates that a debug data break exception occured on a load. 0: No debug data exception on a load occured 1: Debug data exception on a load occurred	R	0
DBp	1	Indicates that a debug software breakpoint exception occured. 0: No debug software breakpoint exception occured 1: Debug software breakpoint exception occurred	R	0
DSS	0	Indicates that a debug single-step exception occured. 0: No debug single-step exception occured 1: Debug single-step exception occured	R	0

3.3.8.2 Debug2 Register (CP0 Register 23, Select 6)

Debug2 Register

31 … 4	3	2	1	0
Reserved	Prm	DQ	TuP	PaCo

Name	Bits	Description	R/W	Reset
Reserved	31:4	Must be written as zeros return zeros on reads.	R	0
Prm	3	Read as zero as the feature is not implemented	R	0
DQ	2	Read as zero as the feature is not implemented	R	0
Tup	1	Read as zero as the feature is not implemented	R	0
PaCO	0	Read as zero as the feature is not implemented	R	0

3.3.8.3 Debug Exception Program Counter Register (CP0 Register 24, Select 0)

The Debug Exception Program Counter (*DEPC*) register is a read/write register that contains the address at which a process resumes after a debug exception has been serviced. All bits of the *DEPC* register are significant and writable.

When a debug exception occurs, the core writes the DEPC register with the value of

- the program counter of the instruction that was the direct cause of the exception, or
- the program counter of the preceding branch or jump instruction that is the adjacent one of the exception-causing instruction in the branch delay slot, and therefore the *Debug Branch Delay* bit in the *Debug* register is set.

Software may write the DEPC register to change the resuming address.

Debug Exception Program Counter Register

31	0
DEPC	

Name	Bits	Description	R/W	Reset
DEPC	31:0	Debug exception point.	R/W	Undef

3.3.8.4 Debug Save Register (CP0 Register 31, Select 0)

The *DESAVE* register functions as a simple scratchpad register. For example, in the dmseg segment this register allows the safe debugging of exception handlers and other types of code where the existence of a valid stack for context saving cannot be assumed.

DESAVE Register

31	0
DESAVE	

Name	Bits	Description	R/W	Reset
DESAVE	31:0	Debug exception save contents.	R/W	Undef

3.3.8.5 WatchLo0 Register (CP0 Register 18, Select 0)

The *WatchLo* and *WatchHi* registers together provide the interface of a watchpoint debug facility which initiates a watch exception if an instruction or data access matches the address specified in the registers. Watch exceptions are taken only if the EXL and ERL bits are zero in the *Status* register. If either bit is a one, the WP bit is set in the *Cause* register, and the watch exception is deferred until both the EXL and ERL bits are zero.

Only a pair of *WatchLo* and *WatchHi* registers is implemented for XBurst CPU core. Software may determine if at least one pair of *WatchLo* and *WatchHi* registers are implemented via the *WR* bit of the Config1 register.

The *WatchLo* register specifies the base virtual address and the type of reference (instruction fetch, load, store) to match. The pair of Watch registers supports all reference types. Software can determine which enables are supported by the Watch register pair by setting all enables bits and reading them back to see which ones were actually set.

Note that a data access watchpoint is never triggered by a Prefetch, CACHE, or SYNCI instruction whose address matches the Watch register pair address-match conditions.

WatchLo Register

31	3	2	1	0
VAddr		I	R	W

Name	Bits	Description	R/W	Reset
VAddr	31:3	This field specifies the virtual address to match. Note that this is a double word address, since bits [2:0] are used to control the type of match.	R/W	0
I	2	If this bit is set, watch exception is enabled for instruction fetches that match the address and are actually issued by the core (speculative fetch never causes Watch exceptions).	R/W	0
R	1	If this bit is set, watch exception is enabled for loads that match the address.	R/W	0
W	0	If this bit is set, watch exception is enabled for stores that match the address.	R/W	0

3.3.8.6 WatchHi0 Register (CP0 Register 19, Select 0)

The *WatchHi* register contains information that qualifies the virtual address specified in the *WatchLo* register: an ASID, a Global (G) bit, and an optional address mask. If the G bit is 1, any virtual address reference that matches the specified address will cause a watch exception. If the G bit is a 0, only those virtual address references for which the ASID value in the*WatchHi* register matches the ASID value in the *EntryHi* register cause a watch exception. The optional mask field provides address masking to qualify the address specified in *WatchLo*.

WatchHi Register

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>M</td><td>G</td><td colspan="4">Reserved</td><td colspan="2">EAS</td><td colspan="8">ASID</td><td colspan="4">Reserved</td><td colspan="9">MASK</td><td>I</td><td>R</td><td>W</td></tr>
</table>

Name	Bits	Description	R/W	Reset
M	31	Another pair of *WatchHi/WatchLo* registers is implemented. Must be written as zero; returns zero on read	R	0
G	30	If this bit is set, any address that matches that specified in the *WatchLo* register causes a watch exception. If this bit is zero, the ASID field of the *WatchHi* register must match the ASID field of the *EntryHi* register to cause a watch exception.	R/W	0
Reserved	29:26	Must be written as zero; returns zero on read	R	0
EAS	25:24	Must be written as zero; returns zero on read.	R	0
ASID	23:16	ASID value which is required to match that in the *EntryHi* register if the G bit is zero in the *WatchHi* register.	R/W	0
Reserved	15:12	Must be written as zero; returns zero on read	R	0
Mask	11:3	Any bit in this field that is a one inhibits the corresponding address bit from participating in the address match. Software may determine how many mask bits are implemented by writing ones to this field and then reading back the result.	R/W	0
I	2	This bit can only be set by hardware when an instruction fetch matches the values in the watch register pair. This bit can only be cleared by software writing a 1 to the bit.	W1C	0
R	1	This bit can only be set by hardware when a load matches the values in the watch register pair. This bit can only be cleared by software writing a 1 to the bit.	W1C	0
W	0	This bit can only be set by hardware when an store condition matches the values in the watch register pair. This bit can only be cleared by software writing a 1 to the bit.	WIC	0

3.3.9 User Mode Support Registers

This section contains the following registers.

- Hardware Enable-HWREna (CP0 Register 7, Select 0)
- UserLocal (CP0 Register 4, Select 2)
- Load Linked Address (CP0 Register 17, Select 0)

3.3.9.1 Hardware Enable-HWREna (CP0 Register 7, Select 0)

The *HWREna* register contains a bit mask that determines which hardware register are accessible via the RDHWR instruction when that instruction is executed in a mode in which coprocessor 0 is not enable.

HWREna Register

31 30	29	28 4	3	2	1	0
Impl	UL	0	CCRes	CC	SYNCI	CPUNum

<table>
<tr><th>Name</th><th>Bits</th><th>Description</th><th>R/W</th><th>Reset</th></tr>
<tr><td>Impl</td><td>31:30</td><td>These bits enable access to the implementation-dependent hardware register 31 and 30.
Read as zero since those registers are not implemented.</td><td>R</td><td>0</td></tr>
<tr><td>UL</td><td>29</td><td>UserLocal register present. Setting 1 permits programs in user mode obtaining the value of the <i>UserLocal</i> register by executing RDHWR $29.</td><td>R/W</td><td>0</td></tr>
<tr><td>0</td><td>28:4</td><td>Must be written as zero; returns zero on read</td><td>R</td><td>0</td></tr>
<tr><td>CCRes</td><td>3</td><td>Resolution of the <i>Count</i> register present. Setting 1 permits programs in user mode obtaining the value of the CCRes by executing RDHWR $3. The value of CCRes denotes the number of cycles between updates of the <i>Count</i> register.
<table>
<tr><th>CCRes</th><th>Value Meaning</th></tr>
<tr><td>1</td><td>Count register increments every cycle</td></tr>
<tr><td>2</td><td>Count register increments every second cycle</td></tr>
<tr><td>3</td><td>Count register increments every third cycle</td></tr>
</table></td><td>R/W</td><td>0</td></tr>
<tr><td>CC</td><td>2</td><td>Count register present. Setting 1 permits programs in user mode to obtain the value of the Count register by executing RDHWR $2</td><td>R/W</td><td>0</td></tr>
<tr><td>SYNCI Step</td><td>1</td><td>L1 cache line size present.
Setting 1 permits programs in user mode to obatin the L1-cache line size. That line size determines the step between successive uses of the SYNCI instruction.</td><td>R/W</td><td>0</td></tr>
<tr><td>CPUNum</td><td>0</td><td>CPUNum present.
Setting 1 permits programs in user mode to obtain the CPU ID number of the core.</td><td>R/W</td><td>0</td></tr>
</table>

3.3.9.2 UserLocal (CP0 Register 4, Select 2)

The *UserLocal* register is a read-write 32-bit register that is not interpreted by the hardware and conditionally readable by software. This register is suitable for a kernel-maintained ID whose value can be read by user-level code with RDHWR $29, as long as HWRENA.UL is set. The presence of the *UserLocal* register is indicated by *Config3.ULRI* = 1.

UserLocal Register

31	0
UserLocal	

Name	Bits	Description	R/W	Reset
UserLocal	31:0	Software information that is not interpreted by hardware.	R/W	0x0

3.3.9.3 Load Linked Address (CP0 Register 17, Select 0)

The *LLAddr* register contains the the physical address corresponding to the virtual address of the load operation caused by the Load Linked (LL) instruction. This register is implementation dependent and for diagnostic purposes only and serves on function during normal operation.

Load Linked Address Register

31	2	1	0
LLAddr			

Name	Bits	Description	R/W	Reset
LLAddr	31:0	This field encodes the physical address read by the most recent Load Linked instruction. The format of this register is implementation dependent, and an implementation may implement as many of the bits or format the address in any way that it find convenient.	R	undef

3.3.10 Kernel Mode Scratch Registers

This section contains the following registers.

- Kernel Scratch Register 1 - KScratch1 (CP0 Register 31, Select 2)
- Kernel Scratch Register 2 - KScratch2 (CP0 Register 31, Select 3)
- Kernel Scratch Register 3 - KScratch3 (CP0 Register 31, Select 4)
- Kernel Scratch Register 4 - KScratch4 (CP0 Register 31, Select 5)
- Kernel Scratch Register 5 - KScratch5 (CP0 Register 31, Select 6)
- Kernel Scratch Register 6 - KScratch6 (CP0 Register 31, Select 7)

3.3.10.1 Kernel Scratch Register 1 - KScratch1 (CP0 Register 31, Select 2)

KScratch1 is a read-write 32-bit register that is used by the kernel for temporary storage of information.

The presence of the *KScratch1* register is indicated by $\mathit{Config4}_{\mathit{KScrExist[2]}}$= 1.

KScratch1 Register

31	0
KScratch1	

Name	Bits	Description	R/W	Reset
KScratch1	31:0	Used by the kernel for temporary storage of information.	R/W	undef

3.3.10.2 Kernel Scratch Register 2 - KScratch2 (CP0 Register 31, Select 3)

KScratch2 is a read-write 32-bit register that is used by the kernel for temporary storage of information.

The presence of the *KScratch2* register is indicated by $\mathit{Config4}_{\mathit{KScrExist[3]}}$= 1.

KScratch2 Register

31	0
KScratch2	

Name	Bits	Description	R/W	Reset
KScratch2	31:0	Used by the kernel for temporary storage of information.	R/W	undef

3.3.10.3 Kernel Scratch Register 3 - KScratch3 (CP0 Register 31, Select 4)

KScratch3 is a read-write 32-bit register that is used by the kernel for temporary storage of information.

The presence of the *KScratch3* register is indicated by $\mathit{Config4}_{\mathit{KScrExist[4]}}$= 1.

KScratch3 Register

31	0
KScratch3	

Name	Bits	Description	R/W	Reset
KScratch3	31:0	Used by the kernel for temporary storage of information.	R/W	undef

3.3.10.4 Kernel Scratch Register 4 - KScratch4 (CP0 Register 31, Select 5)

KScratch4 is a read-write 32-bit register that is used by the kernel for temporary storage of information. The presence of the *KScratch4* register is indicated by *Config4*$_{\mathit{KScrExist[5]}}$= 1.

KScratch4 Register

31	0
KScratch4	

Name	Bits	Description	R/W	Reset
KScratch4	31:0	Used by the kernel for temporary storage of information.	R/W	undef

3.3.10.5 Kernel Scratch Register 5 - KScratch5 (CP0 Register 31, Select 6)

KScratch5 is a read-write 32-bit register that is used by the kernel for temporary storage of information. The presence of the *KScratch5* register is indicated by *Config4*$_{\mathit{KScrExist[6]}}$= 1.

KScratch5 Register

31	0
KScratch5	

Name	Bits	Description	R/W	Reset
KScratch5	31:0	Used by the kernel for temporary storage of information.	R/W	undef

3.3.10.6 Kernel Scratch Register 6 - KScratch6 (CP0 Register 31, Select 7)

KScratch6 is a read-write 32-bit register that is used by the kernel for temporary storage of information. The presence of the *KScratch6* register is indicated by *Config4*$_{\mathit{KScrExist[7]}}$= 1.

KScratch6 Register

31	0
KScratch6	

Name	Bits	Description	R/W	Reset
KScratch6	31:0	Used by the kernel for temporary storage of information.	R/W	undef

4 Exceptions and Interrupts

4.1 Exception Priority

When several exceptions occur simultaneously, the exception with the highest priority is taken. Table 4-1 Priority of Exceptions lists all exceptions including the brief generating conditions and the relative priority of each from highest to lowest.

Table 4-1 Priority of Exceptions

Exception	Description	Priority
Reset	Assertion of reset signal.	0
DSS	EJTAG Debug Single Step.	1
DINT	EJTAG Debug Interrupt.	2
Machine Check	TLB write that conflicts with existing entry.	3
Interrupt	Assertion of unmasked hardware or software interrupt signal.	4
Deferred Watch	Postponed pending watch exception due to EXL/ERL being one	5
DIB	EJTAG Debug hardware instruction break matched.	6
WATCH	Watch on instruction fetch	7
AdEL	Fetch address alignment error or fetch protected address space	8
TLBL	Fetch TLB miss or fetch hit page with V=0 or XI=1.	9
DBp	Execution of SDBBP	10
Sys	Execution of SYSCALL	11
Bp	Execution of BREAK	
CpU	Execution of CpX instruction while relative Status.CUx is disabled.	
RI	Execution of a Reserved Instruction.	
FPE	Floating Point exception.	
MSAFPE	MSA Floating Point exception	
MSADis	MSA Disabled exception	
Ov	Execution of an arithmetic instruction that overflowed.	
Tr	Execution of a trap instruction when tap condition is true.	
DDBL/DDBS	Debug Data Break on Load/Store address match only or Debug Data Break on Store address + data value match	12
WATCH	Watch on data access	13
AdEL/AdES	data access alignment error data access to protected address space	14
TLBL/TLBS	Load/Store TLB miss, or Load/Store hit page with V=0 or RI=1	15
TLB Modify	Store hit page with D=0	16
DDBL/DDBS	Debug Break on Load address + data match	17

4.2 Exception Vector Locations

Table 4-2 Exception Vector

Exception type	**Exception handler entry**				
reset	0xbfc00000				
EJTAG Debug	Prob Trap=0	0xbfc00480			
	Prob Trap=1	0xff200200			
	Status.BEV	Status.EXL	Cause.IV	Base	Offset
TLB Refill	0	0	x	EBase[31:12],12'b0	0x000
	0	1	x		0x180
	1	0	x	0xbfc00200	0x000
	1	1	x		0x180
Interrupt	0	x	0	EBase[31:12],12'b0	0x180
	0	x	1		0x200+
	1	x	0	0xbfc00200	0x180
	1	x	1		0x200
Others	0	x	x	EBase[31:12],12'b0	0x180
	1	x	x	0xbfc00200	

4.3 Exception Handling Process

4.3.1 Enter Exception Handler Routine

It takes several CPU clock cycles to switch from the current context to the exception handler routine.

XBurst®2 CPU core will perform following tasks after an exception is granted.

- Calculate correlative handler entry and cause code in terms of granted exception type and current context.
- Set correct return address value to ErrorPC for granted reset exception, or to EPC for granted generic exception, or to DEPC for granted debug or debug mode exception.
- Set necessary status bits such as Status.EXL and so on.
- After above all has been done, switching core context has been accomplished, then change PC to jump to the expected handler entry for service of granted exception.

4.3.2 Return from Exception Handler Routine

Return from exception routine is performed by executing ERET instruction for non-debug exceptions or DERET instruction for debug/debug mode exceptions.

4.4 Exception Categories

The exception categories being supported by XBurst® CPU core are described below:

Exception	Description
Reset	A Reset Exception occurs when the reset signal is asserted to the core whether it's due to a Cold Reset or a Software Reset behavior.
DSS	When single-step mode is enabled, a Debug Single Step Exception occurs when the core has taken a single execution step in Non-Debug Mode.
DINT	The Debug Interrupt Exception is an asynchronous debug exception that is taken as soon as possible. Debug interrupt request is ignored when the core is already in Debug Mode. The following conditions cause a DINT exception: ● DINT request from EJTAG.
Machine Check	A Machine Check Exception occurs when the core detects one of following conditions being matched: ● Multiple matching entries in the TLB. ● A page with EntryHi.EHINV=0 is written into FTLB and PageMask is not set to a pagesize that is supported by the FTLB. ● A page with EntryHi.EHINV=0 is written into FTLB but the VPN2 field is not consistent with the TLB set selected by the Index register.
Interrupt	The following conditions cause an Interrupt Exception: ● Hardware interrupts come from OST, INTC, etc. ● Software interrupts caused by write one(s) into Cause.IP[1:0].
Deferred Watch	A Deferred Watch Exception occurs when there is a transition of (Status.EXL | Status.ERL) == 1 to (Status.EXL | Status.ERL) == 0 meanwhile WP bit has been set.
DIB	A Debug Instruction Break Exception occurs when an instruction hardware breakpoint matches an executed instruction in Non-debug mode.
Watch	A Watch Exception occurs when an instruction or data reference matches the address information stored in the WatchHi and WatchLo registers. A Watch Exception is taken immediately if the EXL and ERL bits of the Status register are both zero otherwise it's deferred.
Address Error	The following conditions cause an Address Error Exception: ● An instruction is fetched from an non-word address boundary. ● A load/store word instruction is executed with non-word aligned address. ● A load/store halfword instruction is executed with non-halfword aligned address. ● A reference is made to a kernel address space from User Mode.
TLB Refill	A TLB Refill Exception occurs when no TLB entry matches a reference to a mapped address space for an instruction fetch or data access.
Execution Inhibit	An Execute-Inhibit Exception occurs when the virtual address of an instruction fetch matches a TLB entry whose XI bit is set. In the case, the ExcCode of Cause should be set TLBL in this implementation.
Read Inhibit	An Read-Inhibit Exception occurs when the virtual address of a load reference matches a TLB entry whose RI bit is set. In the case, the ExcCode of Cause should

	be set TLBL in this implementation.
TLB Invalid	A TLB Invalid Exception occurs when a instruction fetch or data access matches a TLB entry with valid bit off.
TLB Modified	A TLB Modified Exception occurs on a store reference to a mapped address when the matching TLB entry is valid, but the entry's D bit is zero, indicating that the page is not writable yet.
Sys	A system call exception occurs when a SYSCALL instruction is executed.
Bp	A Breakpoint Exception occurs when a BREAK instruction is executed.
CpU	A Coprocessor Unusable Exception occurs only when a corresponding instruction is executed but the corresponding CUx in Status is zero.
RI	A Reserved Instruction Exception occurs only when a reserved instruction was executed. If both a Coprocessor Unusable Exception and a Reserved Instruction Exception occur on the same instruction, the Coprocessor Unusable Exception takes priority.
FPE	A Floating Point Exception is initiated by the floating point coprocessor signaling an exception.
MSAFPE	MSA Floating Point Exception.
MSADis	MSA Disabled Exception.
Ov	An Integer Overflow Exception occurs when an integer ADD/SUB instruction results in a 2's complement overflow.
Tr	A Trap Exception occurs when a trap instruction results in a TRUE condition.
DDBL/ DDBS	A Debug Data Break Load/Store Exception occurs when a preset hardware data breakpoint matches the address of an executed load/store instruction.

5 Memory Management Unit

5.1 Overview

XBurst®2 CPU core has an on-chip memory management unit (MMU) that implements address translation. The MMU features a resident translation look-aside buffer (TLB) that caches address mapping information for address translation tables located in memory. It enables high-speed translation of virtual addresses into physical addresses. Address translation uses the paging system and supports 8 kinds of page size. The access right to virtual address space can be set for privileged and user modes to provide memory protection.

5.2 Virtual Memory Space

XBurst®2 CPU core uses 32-bit virtual addresses to access a 4-Gbyte virtual address space that is divided into several segments according to different operating mode. Address segments are shown in following figure.

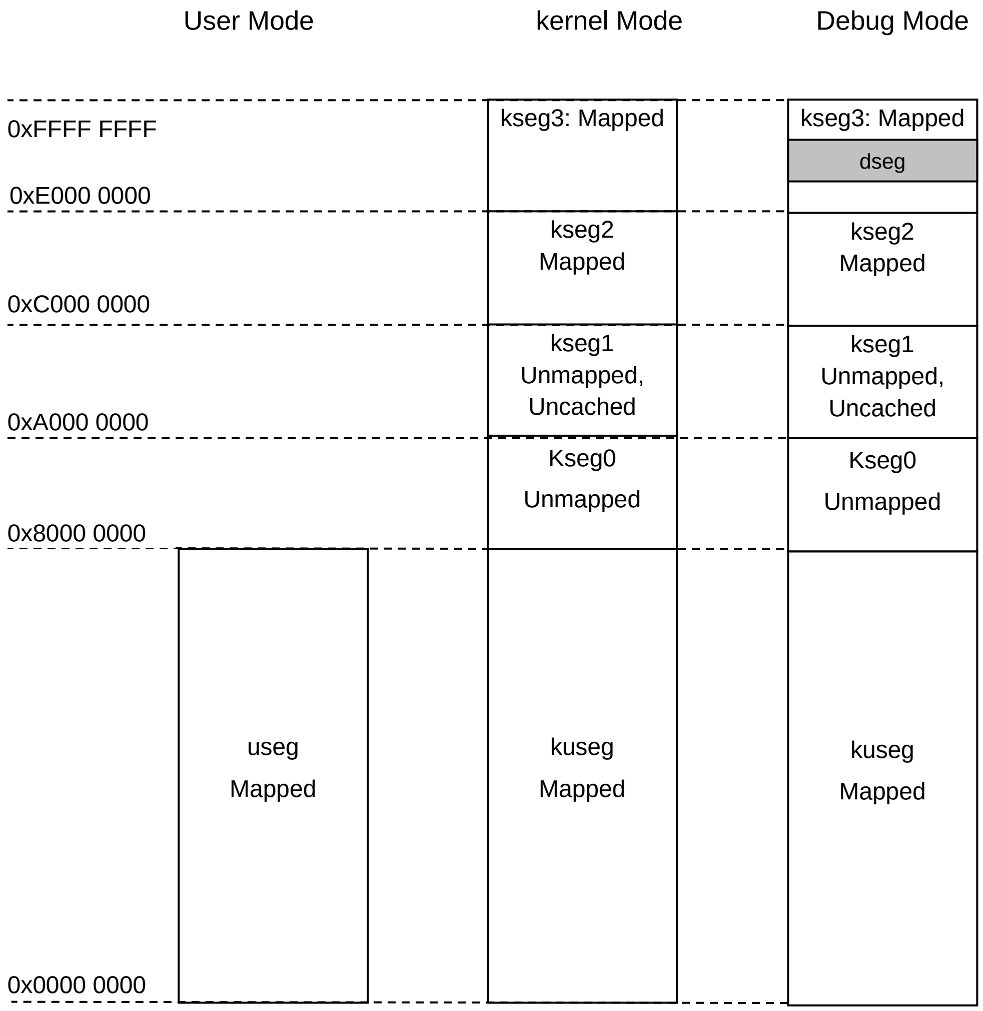


Figure 5.1 Virtual Memory Map

5.2.1 User Mode

The core operates in user mode when the DM bit in the Debug register is 0 and the Status register contains the following bit values:

- UM = 1
- EXL = 0
- ERL = 0

5.2.1.1 useg

The user segment called useg that starts at address 0x0000_0000 and ends at address 0x7FFF_FFFF. The accessing address of useg need be combined with the ASID field of EntryHi register to form a unique virtual address for address mapping by TLB. In user mode, accessing non-useg address will cause an address error exception.

5.2.2 Kernel Mode

The core operates in kernel mode when the DM bit in the *Debug* register is 0 and the *Status* register contains the following values:

- ERL = 1 or EXL=1, in spite of value of UM
- UM = 0, in spite of value of EXL and ERL

When a non-debug exception is granted, EXL or ERL will be set to 1. At the end of an non-debug exception handler routine, an Exception Return (ERET) instruction will be executed to restore the context before exception accepted. In detail, if ERL=1, the ERET instruction will let PC jump to the code position pointed by the *EerrorEPC* register meanwhile clear ERL to 0; while if EXL=1 and ERL=0, the ERET instruction will let PC jump to the code position pointed by the *EPC* register meanwhile clear EXL to 0.

In kernel mode, virtual address space is divided into serveral regions differentiated by the high-order bits of the virtual address.

5.2.2.1 kuseg

kuseg address range is 0x0000_0000 - 0x7FFF_FFFF (2G-Byte). Accessing kuseg need be combined with the ASID field of EntryHi register to form a unique virtual address for address mapping by TLB.

Moreover, when ERL=1, the kuseg region becomes an unmapped and uncached address space. And in that setting, the kuseg virtual address maps directly to the same physical address in spite of the ASID field of EntryHi register.

5.2.2.2 kseg0

In kernel mode, when the most-significant three bits of a virtual address are 100_2, such address belongs to kseg0 with size of 2^{29}-byte (512M-Byte) located at virtual address region 0x8000_0000 - 0x9FFF_FFFF. References to kseg0 are unmapped; the designate physical address for kseg0 is defined by subtracting 0x8000_0000 from the virtual address of kseg0. The K0 field of the *Config* register controls kseg0's cacheability.

5.2.2.3 kseg1

In kernel mode, when the most-significant three bits of a 32-bit virtual address are 101_2, such address belongs to kseg1 with size of 2^{29}-byte (512M-Byte) located at virtual address region 0xA000_0000 - 0xBFFF_FFFF. References to kseg1 are unmapped; the designate physical address for kseg1 is defined by subtracting 0xA000_0000 from the virtual address of kseg1. References to this region are always uncached.

5.2.2.4 kseg2

In kernel mode, when the most-significant three bits of a 32-bit virtual address are 110_2, such address belongs to kseg2 with size of 2^{29}-byte (512M-Byte) located at virtual address region 0xC000_0000 - 0xDFFF_FFFF. References to kseg2 need be combined with the ASID field of *EntryHi* register to form a unique virtual address for address mapping by TLB.

5.2.2.5 kseg3

In kernel mode, when the most-significant three bits of a 32-bit virtual address are 111_2, such address belongs to kseg3 with size of 2^{29}-byte (512M-Byte) located at virtual address region 0xE000_0000 - 0xFFFF_FFFF. References to kseg3 need be combined with the ASID field of EntryHi register to form a unique virtual address for address mapping by TLB

5.3 TLB

The following subsections discuss the TLB memory management scheme used in XBurst®2 CPU core. The TLB consists of two levels: a level-2 joint-TLB (JTLB) and two level-1 micro TLB (MTLB) for instruction fetch and data access seperately:

- Total 8-entry (2 sets * 4-way associative) Instruction micro TLB (IMTLB)
- Total 16-entry (4 sets * 4-way associative) Data micro TLB (DMTLB)
- 32 dual-entry Variable Page Size Translation Lookaside Buffer (VTLB) belonging to JTLB
- 256 dual-entry Fixed Page Size Translation Lookaside Buffer (FTLB) belonging to JTLB

5.3.1 Instruction Micro TLB

IMTLB is an 8-entry MTLB managed by hardware and is transparent to software, which is organized as per entry containing a pair of even-odd pages. IMTLB is dedicated to performing address translation for the instruction fetch with supporting variable page size from 4-KB to 64-MB. If a PC of instruction fetch cannot be translated by the IMTLB, then the JTLB is to be searched further. And if the missed page is successfully found in the JTLB, the translation information will be copied to IMTLB for future usage.

IMTLB entries are functionally refilled from the JTLB (VTLB / FTLB) when required, and automatically cleared whenever the associated VTLB/FTLB entries are modified.

5.3.2 Data Micro TLB

DMTLB is a 16-entry MTLB managed by hardware and is transparent to software, which is organized as per entry containg a pair of even-odd pages. DMTLB is dedicated to performing address translation for load/store operations with supporting variable page size from 4-KB tp 64-MB. Like the IMTLB, if a load/store virtual address cannot be translated by the DMTLB, then the JTLB is to be searched further. And if the missed page is successfully found in the JTLB, the translation information will be copied to DMTLB for future usage.

DMTLB entries are functionally refilled from the JTLB (VTLB/FTLB) when required, and automatically cleared whenever the associated VTLB/FTLB entries are modified.

5.3.3 Variable Page Size TLB (VTLB)

VTLB is a fully associative translation lookaside buffer with 32 dual-page entries and each entry may contain different page size. VTLB is used to translate a unique virtual address {ASID, VA} to a physical address meanwhile provide associative page attributes. The translation is performed by comparing the upper VPN2 bits of the virtual address (along with the ASID bits) against even and odd tag portions of each entry of VTLB, finding the unique match entry, and finally outputing the match entry's physical PPN and associative page attributes.

VTLB implements the following variable page size schemes:

- if "enable ftlb" & "ftlb page-size=4KB" , then VTLB supported page sizes include:
  16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB

- if "enable ftlb" & "ftlb page-size=16KB" , then VTLB supported page sizes include:
  64KB, 256KB, 1MB, 4MB, 16MB, 64MB
- if"disable ftlb", the VTLB supports that:
  4KB, 16KB, 64KB, 256KB, 1MB, 4MB, 16MB, 64MB

VTLB is organized as per entry containing a pair of even-odd pages. Each entry has a virtual tag corresponding to two physical pages: an even page and an odd page. The designate LSB bit of the virtual page number is used to determine which of the even-odd pages hits when the virtual page comparison matches. Since the page sizes in VTLB entries are variable, the determination of which address bits participate in the page comparison and which bit is the designate LSB bit for the even-odd page selection must be on a page-pair basis of each VTLB entry, respectively.

To make a further decription of the process of variable page size TLB lookup, please observe following TLB Tag entry format and TLB Data Entry format first.

Figure 5.2 TLB Tag Entry Format

PageMask[26:13]	G	VPN2[31:13]	ASID[7:0]
41 28	27	26 8	7 0

Figure 5.3 TLB Data Entry Format

FPN1[31:12]	RI1	XI1	C1[2:0]	D1	V1	FPN0[31:12]	RI0	XI0	C0[2:0]	D0	V0
52 34	33	32	31 29	28	27	26 7	6	5	4 2	1	0

Table 5-1 TLB Tag Entry Field Description

Field	Description
PageMask [26:13]	Mask bits for variable page size: 00_0000_0000_0000: 4KB 00_0000_0000_0011: 16KB 00_0000_0000_1111: 64KB 00_0000_0011_1111: 256KB 00_0000_1111_1111: 1MB 00_0011_1111_1111: 4MB 00_1111_1111_1111: 16MB 11_1111_1111_1111: 64MB
G	Global Bit: When set to 1, indicating that this entry is global to all processes and thus disables the effect of ASID for this entry's TLB lookup comparison.
VPN2 [31:13]	This field contains the upper bits of the virtual page number. VPN2[31 : 27] are always included in the TLB lookup comparison, while which bits of VPN2[26 :13] for comparsion are not fixed, depending on the page size defined by PageMask field.
ASID	Address Space Identifier Identifies which process this TLB entry is associated with.

Table 5-2 TLB Data Entry Field Description

Field	Description
PFN0[31:12], PFN1[31:12]	Physical Frame Number represents the upper bits of the translated physical address. For entries with page size larger than 4 KBytes, only a subset of PFN0/1 is actually available.
C0[2:0], C1[2:0]	Cacheability Contains an encoded value of the cacheability attribute and determines whether the page should be placed in the cache or not. The field is encoded as following: 000: Cacheable, write through, no write-allocate 001: Uncacheable, write accelerated 010: Uncacheable 011: Cacheable, write-back, write-allocate 100: Cacheable, write through, no write-allocate, streaming 101: Cacheable, write-back, write-allocate, streaming 110: Reserved 111: Reserved
XI0, XI1	eXecute Inhibit bit 1: Instruction Fetching from the page is inhibited 0: Instruction Fetching from the page is not inhibited
RI0, RI1	Read Inhibit bit 1: Data read from the page is inhibited 0: Data read from the page is not inhibited
D0, D1	“Dirty” or Write-enable bit 1: Indicates that the page has already been written 0: Indicates that the page has never been written before, stores to the page should cause a TLB Modified exception

V0,
V1
	Valid bit
	0: V0==0 means even-page is invalid; V1==0 means odd-page is invalid;
	1: V0==1 means even-page is valid; V1==1 means odd-page is valid;
	accessing an invalid page should cause a TLB Invalid exception

5.3.4 Fixed Page Size TLB (FTLB)

The FTLB is a 256-entry translation lookaside buffer with fixed page size, organized as 64 sets and 4-way association. Each entry contains a pair of even-odd pages. The FTLB can be implemented as the following two page size configuration:

4KB, 16KB

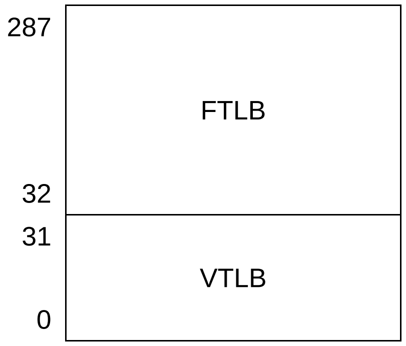
If the FTLB is implemented, the organization is as shown in Table 5-3. Note that all of the entries in the FTLB must be the same page size, either 4KB or 16KB being determined by the FTLBPageSize field of the *Config4* register.

Table 5-3 FTLB Configuration Options

FTLB Parameter	Programmable Options	CP0 Register Reference
Ways	*4 ways*	*Config4FTLB Ways*
Sets	*128 sets*	*Config4FTLB Sets*
Page Size	*4 KB* *16 KB*	*Config4FTLB Page Size*

Both TLBWI and TLBR instructions use Index register to access JTLB (VTLB + FTLB), to correctly access them respectively, programmers must pay attention to the integrated index for JTLB depicted by following Figure 5.4

Figure 5.4 VTLB And FTLB Index Organization



Please note that FTLB has the similar TLB tag entry format and TLB data entry format except PageMask field because its page size is fixed.

5.3.5 Filling JTLB Entry

In order to fill a JTLB entry, software should execute a TLBWI or TLBWR instruction. Prior to invoking one of these instructions, several CP0 registers must be preset with the information to be written to the destination TLB entry.

- *PageMask* need be set in the CP0 *PageMask* register.
- VPN2 and ASID need be set in the CP0 *EntryHi* register.
- PFN0, XI0, RI0, C0, D0, V0 and G bit need be set in the CP0 *EntryLo0* register.
- PFN1, XI1, RI1, C1, D1, V1 and G bit need be set in the CP0 *EntryLo1* register.

Note that the global bit “G” is part of both *EntryLo0* and *EntryLo1*. The resulting “G” bit in the JTLB entry is the logical AND between the two G fields in *EntryLo0* and *EntryLo1*.

5.3.5.1 TLBWI

See following table for detail of how the execution of a TLBWI instruction can access a VTLB/FTLB entry.

	access VTLB	**access FTLB**
set Index register by executing MTC0 $0,0	index.index >= 0, and index.index <= Config1.MMUsize	index.index > Config1.MMUsize, and index.index<= Config1.MMUsize + FTLB sets*FTLB ways
set Index regiser by executing TLBP		

5.3.5.2 TLBWR

See following table for detail of how the execution of a TLBWR instruction can access a VTLB/FTLB entry.

access VLTB	**access FTLB**
value of PageMask register represents a larger page size than the encoding one in Config4.FTLBpagesize. value of the *Random* register denotes the VTLB entry number to be accessed.	value of PageMask register represents the page size encoded in Config4.FTLBpagesize. if Config4.FTLBpagesize == 1, use EntryHi.VPN2[18:13] to access the FTLB set, read LRU status of total 4 ways, then choos a LRU way to overwrite it. if Config4.FTLBpagesize == 2, use EntryHi.VPN2[20:15] to access the FTLB set, read LRU status of total 4 ways, then choose a LRU way to overwrite it.

5.4 Virtual to Physical Address Translation

During virtual-to-physical address translation, XBurst®2 CPU core compares ASID and, depending on pages size, the highest 8-to-20 bits (VPN) of the virtual address will participate the match process. The following figure illustrates the TLB address translation process.

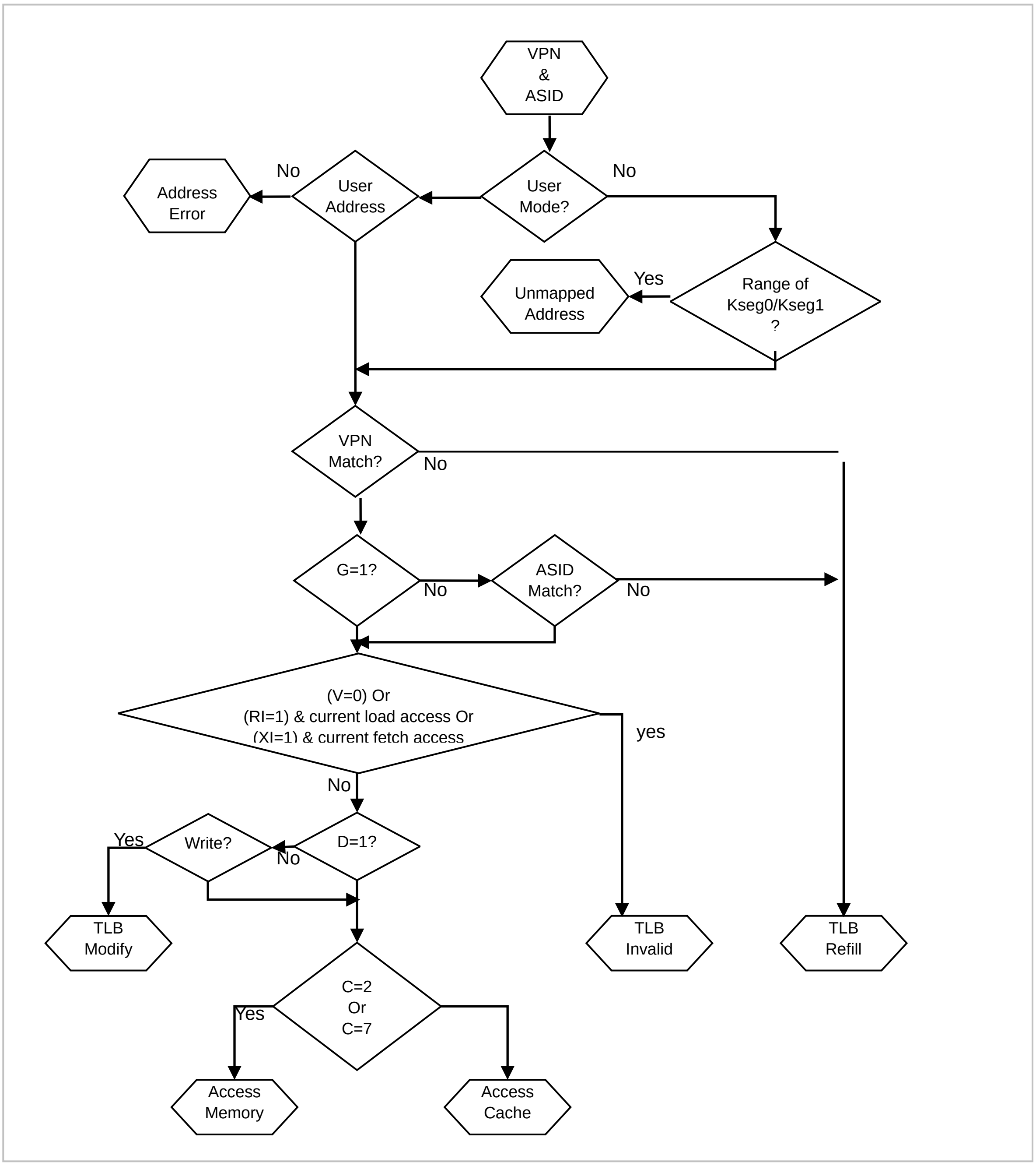


Figure 5.5 TLB Address Translation Flow

A virtual address matches content of a TLB entry when the VPN field of the virtual address equals the VPN field of the entry, and either G bit of the TLB entry is set or ASID field of the virtual address (held in the *EntryHi* register) matches the ASID field of the TLB entry.

6 Caches

6.1 L1-cache

XBurst®2 CPU has separate instruction cache (I-Cache) and data cache (D-Cache) that allows instruction and data references to proceed simultaneously. Its key features are as listed below:

Table 6-1 L1-Cache Features

Parameter	I-Cache	D-Cache
Size	Configurable: 16K/32K Bytes	Configurable: 16K/32K Bytes
Cache Line Size	32 Byte	32 Byte
Numbe of Sets	64 (16KB) /128 (32KB)	64 (16KB) /128 (32KB)
Associativity	8-way	8-way
Lookup policy	physically indexed Physically tagged	physicall indexed Physically tagged
Replace policy	Pseudo Random	Pseudo Random
Lock	N/A	N/A
Others	-	

6.1.1 Cache Coherency Attribute

Cache coherency attribute is specified by the C[2:0] field in EntryLo0 and EntryLo1 entry of the TLB table for mapped address regions including useg/kuseg,kseg2 and kseg3. For unmapped segment kseg0, Config.K0[2:0] field specifies the cache attribute. Unmapped segment kseg1 is not cacheable. The cache attribute is defined as following:

Table 6-2 Cache Coherency Attributes

CCA	Encoding	Description
0	000_2	Cacheable, write-through, write-allocate
1	001_2	Uncacheable write accelerated
2	010_2	Uncacheable
3	011_2	Cacheable, Write-back, write-allocate
4	100_2	Cacheable, Write-through, write-allocate, streaming
5	101_2	Cacheable, Write-back, write-allocate, streaming
6	110_2	Reserved
7	111_2	Reserved

6.1.2 Cache Relative CP0 Registers

Table 6-3 Cache Registers

CP0 register Number	Register Name	Function
17	LLAddr	Load linked address
26	ErrCtl	Enable CACHE Index Store Data instruction
28	TagLo/ DataLo	Cache tag/data interface

6.1.3 Cache Operation Relative Instructions

CACHE instruction format is shown below:

31 26	25 21	20 16	15 0
CACHE 101111_2	Base	op	offset

The 16-bit offset is sign-extended and added to the contents of the base register to form a virtual address. The virtual address need be translated by MMU to form a physical address. The physical address then is used in the following 2 ways based on the operation to be performed:

- The physical address is directly used to address the cache.
- The physical address is used to index the sets and ways of the cache, as shown below.

31 15	14 12	11 5	4 0
unused	Way index	Set Index	Byte offset

op[17:16] of the Cache instruction specifies the cache on which to perform the operation.

00_2 – Level 1 I-Cache

01_2 – Level 1 D-Cache

10_2 – Reserved

11_2 – Secondary Cache

Table 6-4 I-Cache Operations (op[17:16] = 00_2)

Op[20:18]	Operation	Function description
000_2	Index Invalidate	Invalidate an I-cache specified by the virtual address. Virtual address is used to directly index the specified way in specified set.Software. This function can be used by software to invalidate the entire instruction cache by stepping through all valid indices.
001_2	Index Load Tag	Read the tag for the cache block at the specified index into the *TagLo* register. Also read the word corresponding to the word offset(ignore least significant two bits of the address) into the *DataLo* register.
010_2	Index Store Tag	Write the tag for the cache block at the specified index from the *TagLo* register .
011_2	Index Store data	Write the DataLo contents to the way and word index as specified.
100_2	Hit Invalidate	If the virtual address hits I-cache, the hit line is invalidated; otherwise, nothing is done.
101_2~110_2	Reserved	
111_2	Prefetch and lock	If the virtual address misses cache, the line containing the address is fetched from memory. The lock function is not implemented.

Table 6-5 D-Cache Operations (op[17:16] = 01_2)

Op[20:18]	Operation	Function description
000_2	Index write back Invalidate	Invalidate a D-cache specified by the virtual address. Virtual address is used to directly index the specified way in specified set. If the cache line is dirty, write back the dirty data and set it invalid. This function can be used by software to invalidate the entire D-cache by stepping through all valid indices.
001_2	Index Load Tag	Read the tag for the cache block at the specified index into the *TagLo* register. Also read the word corresponding to the word index into the *DataLo* register (ignore VA[1:0]).
010_2	Index Store Tag	Write the tag for the cache block at the specified index from the *TagLo* register. This encoding may be used by software to initialize the entire D-cache by stepping through all valid indices. Doing so requires that the *TagLo* and *TagHi* registers associated with the cache be initialized first.
011_2	Reserved	
100_2	Hit Invalidate	If the virtual address hits D-cache, the hit line is invalidated; otherwise, nothing is done.
101_2	Hit write back Invalidate	If the virtual address hits D-cache, invalidate the hit cache line. If the cache line is dirty, write back the dirty data and set it invalid.
110_2	Hit write back	If D-cache hit and it is dirty, write back dirty data and leave it still valid, but clear the dirty bits Otherwise, treat as NOP.
111_2	Prefecth and lock	If the virtual address misses cache, the line containing the address is fetched from memory，The lock function is not implemented.

NOTES:

For index operation, software should use UNMAPPED address to avoid TLB exceptions.

For non-index operation, the result is UNDEFINED if the virtual address is uncacheable.

6.1.4 PREF/PREFX instruction

PREF instruction format is shown below:

31 26	25 21	20 16	15 0
PREF 110011_2	Base	hint	offset

The 16-bit offset is sign-extended and added to the value of the base register to form a virtual address.

PREFX instruction format is shown below:

31 26	25 21	20 16	15 11	10 6	5 0
COP1X 010011_2	Base	index	hint	00000	PREFX 001111

Adding the contents of the base register and the index register to form a virtual address.

Note that PREF/PREFX instruction does not cause any address-related exceptions. If the virtual address will trigger an address-related exception, the PREF/PREFX operation will be ignored directly. Similarly, PREF/PREFX instruction performs nothing if the virtual address is uncacheable.
The hint field supplies information about the manipulation way to be used. PREF/PREFX is an advisory instruction that may change the performance of the program. However, for all hint values and all virtual addresses, it neither changes the architecturally visible state nor alters the meaning of the program.

Table 6-6 Values of the *hint* Field for the PREF/PREFX Instruction

Hint	Action	Description
0	Prefetch	Prefetch data in the same way as cacheable LOAD instruction. However, it is a non-blocking operation, it does not block pipeline while waiting for the missed data to be returned from external memory. Moreover, if the VA of prefetch may trigger any address relative exception, the exception will be ignored and nothing will be done.
4	Prefetch streaming	Prefetch streaming means the prefetched data most probably will be used only once.
25	VA hit write-back and invalidate	write-back the cache line and invalidate it if hit. The function can be normally performed in user mode.
26	VA hit write-back	write-back the cache line if hit. The function can be normally performed in user mode.
27	Prefetch L2cache	Just prefetch data into L2-cache only.
30	Allocate	Allocate a line directly if it causes D-cache miss for write-only purpose since no data read from memory. Moreover, if an address relative exception may occur, the exception will be ignored and no allocation will be done.

6.1.5 SYNC instruction

31 26	25 11	10 6	5 0
SPECIAL 000000$_2$	0	stype	SYNC 001111

SYNC is used to synchronize memory hierarchy. It forces all buffered or unfinished memory access operations (may be caused by Load/Store instruction, CACHE instruction, PREF instruction, etc.) to complete before the execution of SYNC. In other words, SYNC instruction eliminates potential data coherency hazard in the memory hierarchy in a core.

6.2 L2-cache

XBurst®2 CPU has a unified L2-cache for all implemented symmetric cores. Its key features are as listed below:

Parameter	L2-Cache
Size	configurable: 0KB, 128KB, 256KB, 512KB, 1MB
Cache Line Size	64-byte
Sets/way associativity	256/8, 128/16 (128KB), 512/8, 256/16 (256KB), 1024/8, 512/16 (512KB), 2048/8, 1024/16 (1MB)
Lookup policy	physically indexed Physically tagged
Replace policy	round-robin
Lock	N/A
Others	smart HW prefetcher provides powerful streaming performance

7 Initialize Core State

7.1 Initialized Core State by Hardware

7.1.1 Coprocessor 0 State

Please refer to CP0 Register Descriptions for each CP0 register's reset value.

7.1.2 TLB Initialization

Both FTLB and VTLB are initialized by HW automatically after reset (power-on reset, watchdog reset, or soft reset for core).

7.1.3 Cache Initialization

Both D-cache and I-cache (L1-cache) are all initialized by HW automatically after reset (power-on reset, watchdog reset, or soft reset for core). However, L2-cache is initialized by HW automatically after reset (power-on reset, watchdog reset).

7.2 Initialized Core State by Software

7.2.1 General Purpose Registers Initialization

All 31 integer general purpose registers need be initialized by software after reset.

8 CCU

8.1 Overview

CCU (Core Control Unit) contains control and status registers of the entire SMP (Symmetrical Multi Processing) system. All registers of CCU are memory mapped and located in uncacheable and unmapped kseg1 region, they can be accessed by load/store instructions in kernel mode. CCU implements follwoing functions:

- Record the sleep state of a core
- Control software reset of a core
- Mailbox IRQ supporting IPI mechanism
- Flexible IRQ mask bits masking IRQ
- Hardware spinlock mechanism for atomic access of CCU by multiple cores

8.2 Register Description

Table 8-1 CCU Registers List

Name	Description	R/W	Reset Value	Address offset
CSCR	Core Sleep Control Register	RW	0x????FFFF	0x0000
CSSR	Core Sleep Status Register	R	0x????0000	0x0020
CSRR	Core Software Reset Register	RW	0x????FFFE	0x0040
MSCR	Memory Subsystem Control Register	RW	0x?????000	0x0060
MSIR	Memory Subsystem Implementation Register	R	N/A	0x0064
CCR	CPU Configuration Register	R	N/A	0x0070
PIPR	Peripheral IRQ Pending Register	R	0x????0000	0x0100
PIMR	Peripheral IRQ Mask Register	RW	0x????0001	0x0120
MIPR	Mailbox IRQ Pending Register	R	0x????0000	0x0140
MIMR	Mailbox IRQ Mask Register	RW	0x????0000	0x0160
OIPR	OST*1 IRQ Pending Register	R	0x????0000	0x0180
OIMR	OST IRQ Mask Register	RW	0x????0001	0x01a0
DIPR	Debug Interrupt Pending Register	R	0x????????	0x01c0
DIMR	Debug Interrupt Mask Register	RW	0x00000001	0x01e0
LDIMR<N>*2	Local Debug Interrupt Mask Register	RW	0x00000000	0x0300+N*32
RER	Reset Entry Register	RW	0xBFC0000	0x0f00
CSLR	CCU Spin Lock Register	RW0	0x00000000	0x0fa0
CSAR	CCU Spin Atomic Register	RW	0x????????	0x0fa4
GIMR	Core Global Interrupt Mask Register	RW	0x????1111	0x0fc0
CFCR	CPU Feature Configuration Register	RW	0x00000000	0x0fe0
MBR<N>*2	Mailbox Register	RW	0x00000000	0x1000+N*4
BCER	Bus Exception Control Register	RW	0x000FFFFF	0x1f00

*1 OST: Operating System Timer

*2 N: Core Number: 0, 1, 2, 3... . .For instance, the core3's MBR has the address offset 0x100C.

8.2.1 Cores SLeep Control Register

This register controls whether SOC can enter sleep mode when all CPU cores finish the execution of the **WAIT** instruction and no interrupt is pending. The following figure shows the register format.

CSCR **Base+0x0000**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																SM15	SM14	SM13	SM12	SM11	SM10	SM9	SM8	SM7	SM6	SM5	SM4	SM3	SM2	SM1	SM0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	SM<N>	Sleep Mask. 0: core<N>'s sleep status can't be sent to SOC's CPM*1. 1: core<N>'s sleep status can be sent to SOC's CPM. If any one of these bits is a zero, SOC can't enter sleep mode even if all cores finish the execution of the **WAIT** instruction.	RW

*1 CPM: Clock and Power Management

8.2.2 Core Sleep Status Register

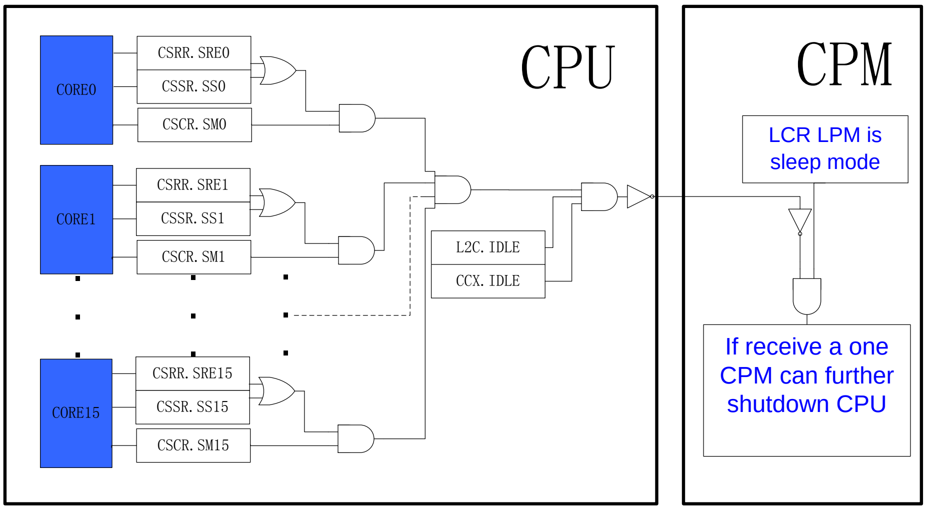
The field SS<N> in CSSR indicates the sleep status of the core<N>. A one in the corresponding bit means the corresponding core is in sleep mode. A zero means the corresponding core is not in sleep mode. The following figure shows the register format.

CSSR **Base+0x0020**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																SS15	SS14	SS13	SS12	SS11	SS10	SS9	SS8	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	SS<N>	Sleep Status. 0: Core<N> is not sleeping 1: Core<N> is sleeping. When executing a WAIT instruction in a core, the core need complete all outstanding operations, then freezes the pipeline and send a sleep status to CCU. CCU captures the signal and set the corresponding SS bit to one and then turns off Core<N>'s clock. Later, when an interrupt need be taken by core<N>, CCU should clear the corresponding SS<N> bit and restore Core<N>'s clock.	R

The following picture is the relationship between CCU's CSCR, CSSR and SOC CPM's sleep mode. Refer to the chapter about CPM of Ingenic SOC manual for more information.



8.2.3 Core Software Reset Register

This register permits system programmer to dynamically reset a core by software. One can make a 0 to 1 then to 0 transition for a SREx bit to generate a reset pulse to the corresponding logic core. It must not left any outstanding unfinished bus access in a core when doing software reset for it, otherwise, the software reset will result in bus deadlock. The following figure shows the register format.

CSRR **Base+0x0040**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																SRE15	SRE14	SRE13	SRE12	SRE11	SRE10	SRE9	SRE8	SRE7	SRE6	SRE5	SRE4	SRE3	SRE2	SRE1	SRE0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	SRE<N>	Software Reset. 0: the core is out of soft-reset state. 1: the core is in soft-reset state. After external hardware reset，software can reset core<N> by setting SRE<N> to 1. Furthermore, after setting SRE<N> to 1, setting SRE<N> to 0 will let the core get away from reset status. But the read-write property of SRE0 is controlled by CFCR. EnSRE0Wr.	RW

8.2.4 Memory Subsystem Control Register

MSCR **Base+0x0060**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved							PSel	Reserved																				QoSE	DisPFB1	DisPFB2	DisL2C
Rst	?	?	?	?	?	?	?	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0

Bits	Name	Description	R/W
31:25	Reserved	Writing has no effect, read as zero.	R
24	Psel	Parallel Performance Counter(PPC) selection. (NOTES: only for CPU internal verification) 0: PPC for L2C 1: PPC for CCX	R/W
23:4	Reserved	Writing has no effect, read as zero.	R
3	QoSE	Used to enable the dynamic QoS identifier from L2C to memory controller. 0: The QoS identifier is fixed and the highest priority. 1: The QoS identifier is dynamic and based on the status of L2C.	RW
2	DisPFB1	0: Enable the prefetcher between L1 cache and L2 cache. 1: Disable the prefetcher between L1 cache and L2 cache.	RW
1	DisPFB2	0: Enable the prefetcher between L2 cache and DDR. 1: Disable the prefetcher between L2 cache and DDR.	RW
0	DisL2C	0: Enable L2 cache controller. 1: Disable L2 cache controller. After disabling L2 cache controller, it seems that all memory access between CPU and external bus occuring normally except L2-cache disappeared.	RW

8.2.5 Memory Subsystem Implementation Register

MSIR **Base+0x0064**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																ProcessorID								Revision							
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:16	Reserved	Writing has no effect, read as zero.	R
15:8	ProcessorID	The same meaning as the field of *PRID* register	0x20
7:0	Revision	Specifies the revision number of the memory subsystem including CCX and L2C.	R

8.2.6 CPU Configuration Register

CCR **Base+0x0070**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																								TOTAL							
Rst	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	?	?	?	?	?	?	?	?

Bits	Name	Description	R/W
31:8	Reserved	Writing has no effect, read as zero.	R
7:0	TOTAL	This field specifies the amount of the core in a multi-processor system and can be used by software to distinguish the amount of the core. The total number can be from 0 to 255. In a single processor system, the value is zero. In the SMT system, the value means the total amount of the logic core. For examples, if a SMT system has 2 physical cores/4 threads, the value should be 3.	R

8.2.7 Peripheral IRQ Pending Register

PIPR **Base+0x0100**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IP15	IP14	IP13	IP12	IP11	IP10	IP9	IP8	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IP<N>	Peripheral IRQ pending. 0: no peripheral IRQ to core<N> is pending. 1: peripheral IRQ to core<N> is pending.	R

8.2.8 Peripheral IRQ Mask Register

PIMR **Base+0x0120**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IM<N>	Peripheral IRQ of Core<N> mask. 0: pending periperal IRQ can not enter its corresponding core 1: pending periperal IRQ can enter its corresponding core (GIMR IM<N> should be set 1 first, GIMR is described in later chapter)	RW

8.2.9 Mailbox IRQ Pending Register

MIPR **Base+0x0140**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IP15	IP14	IP13	IP12	IP11	IP10	IP9	IP8	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IP<N>	Indicates pending mailbox IRQ to Core<N>. 0: no mailbox IRQ to core<N> is pending. 1: mailbox IRQ to core<N> is pending. Hardware sets the IP<N> automatically when software writes nonzero value into the MBR<N>. Hardware clears the IP<N> automatically when software writes zero value into the MBR<N>.	R

8.2.10 Mailbox IRQ Mask Register

MIMR **Base+0x0160**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IM<N>	Mailbox IRQ of Core<N> mask. 0: pending Mailbox IRQ can not enter its corresponding core 1: pending Mailbox IRQ can enter its corresponding core(GIMR IM<N> should be 1 first, GIMR is described in later chapter)	RW

8.2.11 OST IRQ Pending Register

OIPR **Base+0x0180**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IP15	IP14	IP13	IP12	IP11	IP10	IP9	IP8	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IP<N>	Indicates pending OST IRQ to Core<N>. 0: no OST IRQ to core<N> is pending. 1: OST IRQ to core<N> is pending.	R

8.2.12 OST IRQ Mask Register

OIMR **Base+0x01a0**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IM<N>	OST IRQ of Core<N> mask. 0: pending OST IRQ can not enter its corresponding core 1: pending OST IRQ can enter its corresponding core (GIMR IM<N> should be 1 first, GIMR is described in later chapter)	RW

8.2.13 Debug Interrupt Pending Register

DIPR **Base+0x01c0**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IP15	IP14	IP13	IP12	IP11	IP10	IP9	IP8	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:16	Reserved	Writing has no effect, read as zero.	R
15:0	IP<N>	Interrupt Pending<N>. Every bit<N> denotes whether a debug interrupt is pending and is routed to the corresponding core<N>. 0: no debug interrupt is pending. 1: debug interrupt is pending. The debug interrupt request can be routed from external EJTAG controller or other cores.	R

8.2.14 Debug Interrupt Mask Register

The Debug Interrupt Mask Register(DIMR) contains mask bits used to control which of the cores should receive a EJTAG Debug Interrupt request(usually from a EJTAG). When DIMR.IM<N> is set. The reset value of core0's DIMR.IM0 is 1. This is used to make sure that core 0, as a default, can accept Debug Interrupt Request after reset.

DIMR **Base+0x01e0**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bits	Name	Description	R/W
31:16	Reserved	Writing has no effect, read as zero.	R
15:0	IM<N>	External Debug Interrupt Mask of core<N>. Every bit controls the enabling of the corresponding core<N>'s Debug interrupt request which comes directly from EJTAG. 0: debug interrupt disabled 1: debug interrupt enabled	R/W

8.2.15 Reset Entry Register

RER **Base+0x0f00**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	ENTRY																															
Rst	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:0	ENTRY	Reset exception handler entry of every core. The value denotes where the entry point of the reset exception handler is. The initial value is 0xbfc00000.	RW

8.2.16 Mailbox Register<N>

MBR **Base+0x1000+N*4**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	MSG																															
Rst	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:0	MSG	Message to Core<N>. The nonzero value is available for use as software flags or parameters.	RW

8.2.17 CCU Spin Lock Register

CSLR **Base+0x0fa0**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Lock	Value																														
Rst	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31	Lock	The lock flag. 1: indicates that CSLR.Value is being locked. 0: When writing CSAR by software HW will copy the CSAR.value to CSLR.Value meanwhile setting CSLR.Lock to 1; Note that CSLR.Lock can only be cleared to zero by software;	RW0
30:0	Value	Lock value. This field is updated by hardware based on CSAR.value when CSLR.Lock is zero and writing CSAR.value by software.	R

8.2.18 CCU Spin Atomic Register

CSAR **Base+0x0fa4**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved	Value																														
Rst	?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31	Reserved	Writing has no effect, read as zero.	R
30:0	Value	It's meaning is determined by software. When the CSLR.Lock is zero writing this filed by software is available, the current value to be written into CSAR.Value will be replicated into CSLR.Value by hardware automatically as well as setting CSLR.Lock to 1.	RW

8.2.19 Global Interrupt Mask Register

GIMR **Base+0x0fc0**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																IM15	IM14	IM13	IM12	IM11	IM10	IM9	IM8	IM7	IM6	IM5	IM4	IM3	IM2	IM1	IM0
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	Name	Description	R/W
31:N+1	Reserved	Writing has no effect, read as zero.	R
N:0	IM<N>	Global interrupt of core<N> mask. 0: Any IRQ can not enter its corresponding core; 1: Whether an IRQ can enter its corresponding core or not is determined by its local mask register (eg. PIMR/MIMR/OIMR)	RW

8.2.20 CPU Feature Configuration Register

CFCR **Base+0x0fe0**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved																					EnSREOWr	DisBCConfir	DisIsDone	DisPCValid	DisFuse	DisMissGate	DisLoopGate	**DisSimpleLoop**	DisFTLB	DisMC	EnTimer
Rst	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description	R/W
31:11	Reserved	Writing has no effect, read as zero.	R
10	EnSRE0Wr	0:CSRR.SRE0 is read only 1:CSRR.SRE0 are read-write	RW
9	DisBCConfirm	0: Enable confirm optimization for Base Cache in IU; 1: Disable confirm optimization for Base Cache in IU;	RW
8	DisIsDone	0: A dedicated done flag for IU pipeline; 1: No dedicated done flag for IU pipeline;	RW
7	DisPCValid	0: PC with a valid flag; 1: PC without a valid flag;	RW

6	DisFuse	0: Eanble instruction Fusion optimization;
		1: Disable instruction Fusion optimization;
			RW

5	DisMissGate	0: Enable IU low-power gate when there is a DCache Miss; 1: Disable IU low-power gate when there is a DCache Miss;	RW
4	DisLoopGate	0: Enable IFU low-power gate when enter into simple loop; 1: Disable IFU low-power gate when enter into simple loop;	RW

3	DisSimpleLoop	0: Enable IFU Simple Loop;
		1: Disable IFU Simple Loop;
			RW

2	DisFTLB	0: Enable FTLB;
		1: Disable FTLB;
			RW

1	DisMC	0: Enable machine check excepting; 1: Disable machine check excepting;	RW
0	EnTimer	0: MIPS CP0 timer IRQ can not enter its corresponding core; 1: MIPS CP0 timer IRQ can enter its corresponding core;	RW

Note: the default value for each configuration field is the best one for normal chip configuration, DO NOT modify them except for special purpose (like performance diagnosis).

8.2.21 Bus Exception Control Register

BECR **Base+0x1f00**

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	BusExp	CntEn	Reserved										BusCnt																			
Rst	0	0	?	?	?	?	?	?	?	?	?	?	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bits	Name	Description	R/W
31	BusExp	0:No bus deadlock; 1: Bus deadlock occurs, the hardware will send an exception signal to BIU; Hardware sets the BusExp automatically when CntEn is equal to 1 and an internal timeout counter is equal to 0 (the initial value of the counter comes from BusCnt). After the BIU receives an active BusExp signal, the bus deadlock will be released forcibly. Therefore, Active BusExp signal means the external bus is not usable any longer.	RW0
30	CntEn	0:Disable bus timeout counter; 1:Enable bus timeout counter;	RW
29:20	Reserved	Writing has no effect, read as zero.	R
19:0	BusCnt	When the CntEn being 1 and external bus being busy, an internal bus timeout counter will use BusCnt as its inital value and then will decrease 1 per cycle until counting to 0 or bus becoming idle again before counting to 0.	RW

8.3 Usage

8.3.1 The Configuration of CCU

CCU is shared by entire SMP system and software need by carefully operated to modify the configuration of CCU. Following are some examples of how to manipulate CCU register safely.

1) try and lock CCU

```
current_processor_id = smp_processor_id();
cslr_mask = 0x7fffffff;
while(1) {
    ccu_write(CSAR, current_processor_id);
    cslr = ccu_read(CSLR);
    if((cslr < 0) && (cslr & clsr_mask) == current_processor_id)) {
        break;
    }
}
```

2) unlock CCU

```
current_processor_id = smp_processor_id();
cslr_mask = 0x7fffffff;
cslr = ccu_read(CSLR);
if((cslr < 0) && ((cslr &clsr_mask) == crrent_processor_id)){
    ccu_write(CSLR,0);
}
```

The function smp_processor_id() can return the current core number by read CP0.EBase.CPUNum (Cp0 Register 15，Select1). The function ccu_read() can load a value from the specified CCU register. The function ccu_write() can store a value into the specified CCU register.

9 EJTAG Debug Support

The EJTAG unit provides a system debug facility for the device. The EJTAG functions are not normally controlled by the terminal users, but rather are controlled by a debugger. XBurst®2 CPU supports two modes for JTAG Debug: a compliant MIPS Mode and a custom Accelerated Mode (ACC Mode) which accelerates access to dmseg and extends dmseg region.

This chapter need to be read in conjunction with the MIPS EJTAG Specification that was included as part of the release.

9.1 Overview

The EJTAG debug logic in the XBurst2 core is compliant with EJTAG Specification 2.0 and provides the following:

1. Standard core debug features.
2. Optional hardware breakpoints.
3. Standard Test Access Port (TAP) for a dedicated connection to a debug host.

EJTAG debug resources are often controlled via high level debugger commands. The following is a brief overview of some EJTAG features.

- EJTAG TAP: The optional JTAG TAP associated with an EJTAG debug block used for communications with an EJTAG probe and debugger.
- ECR (EJTAG Control Register): This register is used mostly by probe developers and can only be accessed via a probe.
- DCR (Debug Control Register): This register is located in the drseg memory segment and can only be accessed in Debug mode.
- DINT (Debug Interrupt): an interrupt which causes a debug exception and entry into debug mode.
- DRSEG (Debug Register Segment): A memory overlay, present only while executing in debug mode, that allows access to registers controlling various EJTAG debug features.
- DMSEG (Debug Memory Segment): A memory overlay, present only while in debug mode and ECR.ProbEn is set, that an EJTAG probe emulates by satisfying processor accesses (fetches, loads, and stores.) The emulation is carried out via TAP data registers CONTROL, ADDRESS, and DATA.
- Single-Step: A debug setting that results in a debug exception after execution of a single non-debug mode instruction has completed.
- Hardware Breakpoint: A hardware resource capable of detecting execution or data access at virtual addresses.
- Software Breakpoint: The instruction “SDBBP” which causes a debug exception on execution. Debuggers will temporarily replace an instruction of your program with this instruction on setting a breakpoint in writeable memory.

9.2 Detecting Debug Mode

The DM bit of the CP0 Debug register (CP0 Register 23, Select 0) indicates if the processor is operating in debug mode. If this bit is set, the processor is operating in debug mode. This bit is set on any debug exception and is cleared by executing a DERET instruction. Refer to CP0 Registers for more information on the Debug register.

This bit is available to both probe and non-probe related configurations and can be read at any time. The user does not need to be in Debug mode in order to read this bit. This bit, along with the associated fields in this register, can be used by software to determine the conditions under which Debug mode was entered.

9.3 Ways of Entering Debug Mode

There are five ways to enter Debug mode. Most of these ways can be entered from either software, or from a debug probe. All of these ways cause the *DM* bit in the *CP0 Debug* register to be set.

1. EJTAG Debug Single Step.
2. EJTAG Debug Interrupt. Caused by setting the EJTAGBrk bit in the ECR register.
3. EJTAG debug hardware data breakpoint match.
4. EJTAG debug hardware instruction breakpoint match.
5. EJTAG Breakpoint , execution of SDBBP instruction.

9.4 Exiting Debug Mode

As described above, there are five basic ways to enter debug mode. When in debug mode, the mode can only be exited in one of three ways:

1. Execution of a Debug Exception Return (DERET) instruction.
2. Reset the core.
3. Power cycle the core.

During normal operation, exceptions are taken by the core and processed. Once the exception processing is complete, software executes an Exception Return (ERET) instruction. When in debug mode, software executes a Debug Exception Return (DERET) instruction. This causes the core to exit debug mode and return to previous mode as determined by the programmer (normal, kernel, etc.).

Note that for a DERET instruction to be executed, the core must be in a state where it is fetching instructions. If for any reason the instruction stream has been halted and can not resume, then the DERET instruction can not be executed. In this case, the only other options are resetting the core or cycling the power to the core.

9.5 Hardware Breakpoints

Hardware breakpoints provide for the comparison by hardware of executed instructions and data load/store transactions. It is possible to set instruction breakpoints on addresses even in ROM area. Data breakpoints can be set to cause a debug exception on a specific data transaction. Instruction and data hardware breakpoints are alike for many aspects, and are thus described in parallel in the following. The term "hardware" is not applied to breakpoint, unless required to distinguish it from software breakpoint.

In the XBurst2 core, there are two instruction breakpoints and two data breakpoints.

9.5.1 Instruction Breakpoints

Instruction breakpoints occur on instruction fetch operations and the break is set on the virtual address used by the instruction fetch unit. Instruction breakpoints can also be made on the ASID value used by the TLB-based MMU. Finally, a mask can be applied to the virtual address to set breakpoints on a range of instructions.

Instruction breakpoints compare the virtual address of the executed instructions (PC) and the ASID with the registers for each instruction breakpoint including masking of address and ASID. When an instruction breakpoint matches, a trigger is generated and a debug exception is optionally signaled. An internal bit in the instruction breakpoint registers is set to indicate that the match occurred.

9.5.2 Data Breakpoints

Data breakpoints occur on load/store transactions. Breakpoints are set on virtual address and ASID values, similar to the Instruction breakpoint. Data breakpoints can be set on a load, a store or both. Data breakpoints can also be set based on the value of the load/store operation. Finally, masks can be applied to both the virtual address and the load/store value.

Data breakpoints compare the transaction type (TYPE), which may be load or store, the virtual address of the transaction (ADDR), the ASID, accessed bytes (BYTELANE) and data value (DATA), with the registers for each data breakpoint including masking or qualification on the transaction properties. When a data breakpoint matches, a trigger is generated and a debug exception is optionally signaled. An internal bit in the data breakpoint registers is set to indicate that the match occurred.

9.5.3 Overview of Instruction Breakpoint Registers

Up to two instruction breakpoints are available and are numbered 0 to 1 for registers and breakpoints, and the number is indicated by n. The registers for each breakpoint are shown in Table below and IBS is the register with implementation indication and status for instruction breakpoints in general.

Table 9-1 Overview of Registers for Each Instruction Breakpoint

Register Mnemonic	Register Name and Description
IBS	Instruction Breakpoint Status
IBAn	Instruction Breakpoint Address n
IBMn	Instruction Breakpoint Address Mask n
IBASIDn	Instruction Breakpoint ASID n
IBCn	Instruction Breakpoint Control n

9.5.4 Overview of Data Breakpoint Registers

Up to two instruction breakpoints are available and are numbered 0 to 1 for registers and breakpoints, and the number is indicated by n. The registers for each breakpoint are shown in Table below and DBS is the register with implementation indication and status for data breakpoints in general.

Table 9-2 Overview of Registers for Each Data Breakpoint

Register Mnemonic	Register Name and Description
DBS	Data Breakpoint Status

DBAn	Data Breakpoint Address n
DBMn	Data Breakpoint Address Mask n
DBASIDn	Data Breakpoint ASID n
DBCn	Data Breakpoint Control n
DBVn	Data Breakpoint Value n

9.6 Conditions for Matching Breakpoints

A number of conditions must be fulfilled in order for a breakpoint to match on an executed instruction or a data access. These conditions are described in the following subsections. A breakpoint only matches for instructions executed in Non-Debug Mode, never due to instructions executed in Debug Mode.

The match of an enabled breakpoint generates a debug exception and/or a trigger indication. The BE and/or TE bits in the IBCn or DBCn registers enable the breakpoints for breaks and triggers, respectively.

9.6.1 Conditions for Matching Instruction Breakpoints

When an instruction breakpoint is enabled, that breakpoint is evaluated for the address of every executed instruction in non-debug mode, including execution of instructions at an address causing an address error on an instruction fetch.

The breakpoint is not evaluated on instructions from a speculative fetch or execution, nor for addresses which are unaligned with an executed instruction.

A breakpoint match depends on the virtual address of the executed instruction (PC), which can be masked at the bit level. The match can also include an optional compare of the ASID value. The registers for each instruction breakpoint contain the values and mask used in the compare, and the equation that determines the match is shown below in C-like notation.

```
IB_match =
          ( ! IBCnASIDuse || ( ASID == IBASIDnASID ) ) &&
          ( <all 1's> ==
            ( IBMnIBM | ~ ( PC ^ IBAnIBA ) && ( (IBMnISAM | ~(ISAMode ^ IBAnISA)))
          )
```

The match indication for instruction breakpoints is always precise, i.e., indicated on the instruction causing the `IB_match` to be true.

9.6.2 Conditions for Matching Data Breakpoints

When a data breakpoint is enabled, that breakpoint is evaluated for every data transaction due to a load/store instruction executed in non-debug mode, including coprocessor loads/stores and transactions causing an address error on data access. The breakpoint is not evaluated due to a PREF instruction or other transactions which are not part of explicit load/store transactions in the execution flow, nor for addresses which are not the explicit load/store source or destination address.

A breakpoint match depends on the transaction type (TYPE) as load or store, the address, and optionally the data value of a transaction. The registers for each data breakpoint contain the value and mask used in the compare, and the equation that determines the match is shown below in C-like notation.

The overall match equation is the DB_match.

```
DB_match =
            ( ( ( TYPE == load ) && ! DBCnNoLB ) ||
              ( ( TYPE == store ) && ! DBCnNoSB ) ) &&
            DB_addr_match && ( DB_no_value_compare || DB_value_match )
```

The match on the address part, DB_addr_match, depends on the virtual address of the transaction (ADDR), the ASID value, and the accessed bytes (BYTELANE) where BYTELANE[0] is 1 only if the byte at bits [7:0] on the bus is accessed, and BYTELANE[1] is 1 only if the byte at bits [15:8] is accessed, etc. The DB_addr_match is shown below.

```
DB_addr_match =
                ( ! DBCnASIDuse || ( ASID == DBASIDnASID ) ) &&
                ( <all 1's> == ( DBMnDBM | ~ ( ADDR ^ DBAnDBA ) ) ) &&
                ( <all 0's> != ( ~ BAI & BYTELANE ) )
```

The size of *DBCn*BAI and BYTELANE is 4 bits.

Data value compare is included in the match condition for the data breakpoint depending on the bytes (BYTELANE as described above) accessed by the transaction, and the contents of breakpoint registers. The DB_no_value_compare is shown below.

```
DB_no_value_compare =
                ( <all 1's> == ( DBCnBLM | DBCnBAI | ~ BYTELANE ) )
```

The size of *DBCn*BLM, *DBCn*BAI and BYTELANE is 4 bits.

In case a data value compare is required, DB_no_value_compare is false, then the data value from the data bus (DATA) is compared and masked with the registers for the data breakpoint. The endianess is not considered in these match equations for value, as the compare uses the data bus value directly, thus debug software is responsible for setup of the breakpoint corresponding with endianess.

```
DB_value_match =
( ( DATA[7:0] == DBVnDBV[7:0] ) || !BYTELANE[0] || DBCnBLM[0] || DBCnBAI[0] ) &&
( ( DATA[15:8] == DBVnDBV[15:8] ) || !BYTELANE[1] || DBCnBLM[1] || DBCnBAI[1] ) &&
( ( DATA[23:16] == DBVnDBV[23:16] ) || !BYTELANE[2] || DBCnBLM[2] || DBCnBAI[2] )&&
( ( DATA[31:24] == DBVnDBV[31:24] ) || !BYTELANE[3] || DBCnBLM[3] || DBCnBAI[3] ))
```

The match for a data breakpoint compare is always precise, since the match expression is fully evaluated at the time the load/store instruction is executed. A true DB_match can thereby be indicated on the very same instruction causing the DB_match to be true.

9.6.3 SIMD Load/Store Handling

There is no data compare for SIMD and FPU loads/stores, which means if DB_no_value_compare is true, DB_match will never be true.

9.7 Debug Exceptions from Breakpoints

This section describes how to set up instruction and data breakpoints to generate debug exceptions when the match conditions are true.

9.7.1 Debug Exception by Instruction Breakpoint

If the breakpoint is enabled by the *BE* bit in the *IBCn* register, then a Debug Instruction Break Exception occurs if the `IB_match` equation is true. The corresponding *BS*[n] bit in the *IBS* register is set when the breakpoint generates the debug exception.

The Debug Instruction Break Exception is always precise, so the *DEPC* register and the *DBD* bit in the *Debug* register point to the instruction that caused the `IB_match` equation to be true.

The instruction receiving the debug exception does not update any registers due to the instruction, nor does any load or store by that instruction occur. Thus a debug exception from a data breakpoint cannot occur for instructions receiving a debug instruction break exception.

The debug handler usually returns to the instruction causing the debug instruction break exception, whereby the instruction is executed. Debug software is responsible for disabling the breakpoint when returning to the instruction; otherwise the debug instruction break exception reoccurs.

9.7.2 Debug Exception by Data Breakpoint

If the breakpoint is enabled by *BE* bit in the *DBCn* register, then a Debug Data Break Exception occurs when the `DB_match` condition is true. The corresponding *BS*[n] bit in the *DBS* register is set when the breakpoint generates the debug exception.

A matching data breakpoint always generates a precise debug exception. The DEPC register and DBD bit in the Debug register points to the instruction that caused the `DB_match` equation to be true.

The instruction causing the Debug Data Break Exception does not update any registers due to the instruction, and the following applies to the load or store transaction causing the debug exception:

- A store transaction is not allowed to complete the store to the memory system.
- A load transaction is not allowed to complete the load.

The result of this is that the load or store instruction causing the debug data break exception appears as not executed.

Any *BS*[n] bit set prior to the match and debug exception are kept set, since *BS*[n] bits are only cleared by debug software.

The debug handler usually returns to the instruction causing the Debug Data Break Exception, whereby the instruction is re-executed. Debug software is responsible for disabling breakpoints when returning to the instruction, otherwise the debug data break exception reoccurs.

9.7.3 Breakpoint Used as Triggerpoint

When an enabled instruction or data breakpoint matches, the corresponding bit in the *IBS.BS* or *DBS.BS* field is set. If the breakpoints are to be used only as trigger events, the signaling of the debug exception can be suppressed by clearing the *IBCn/DBCn.BE* field and setting the *IBCn/DBCn.TE* field.

9.8 Test Access Port (TAP)

The TAP is used only when a probe is connected to the XBurst2 core.

The following main features are supported by the TAP module:

- 5-pin industry standard JTAG Test Access Port (*TCK*, *TMS*, *TDI*, *TDO*, *TRST_N*) interface which is compatible with IEEE Std. 1149.1.
- Target chip and EJTAG feature identification available through the Test Access Port (TAP) controller.

- The processor can access external memory on the EJTAG Probe serially through the EJTAG pins. This is achieved through Processor Access (PA), and is used to eliminate the use of the system memory for debug routines.
- Support for both ROM based debugger and debugging both through TAP.

9.8.1 EJTAG Internal and External Interfaces

The external interface of the EJTAG module consists of the 5 signals defined by the IEEE standard.

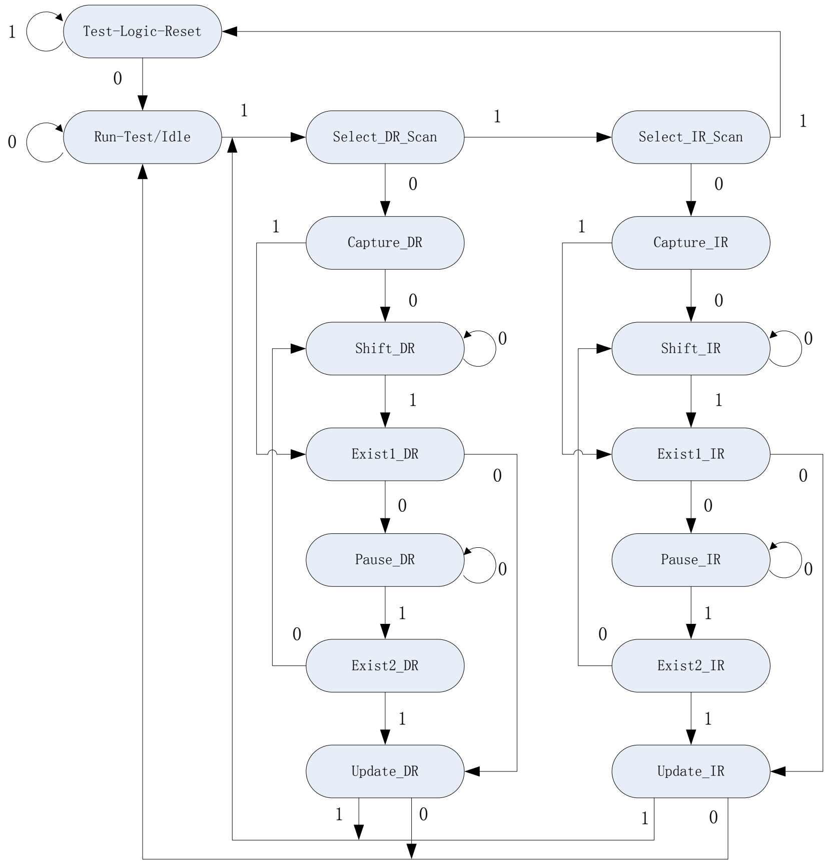
Table 9-3 EJTAG Interface Pins

Pin	Type	Description
TCK	I	Test Clock Input Input clock used to shift data into or out of the Instruction or data registers. The TCK clock is independent of the processor clock, so the EJTAG probe can drive TCK independently of the processor clock frequency. The core signal for this is called EJ_TCK
TMS	I	Test Mode Select Input The TMS input signal is decoded by the TAP controller to control test operation. TMS is sampled on the rising edge of TCK. The core signal for this is called EJ_TMS
TDI	I	Test Data Input Serial input data (*TDI*) is shifted into the Instruction register or data registers on the rising edge of the *TCK* clock, depending on the TAP controller state. The core signal for this is called *EJ_TDI*
TDO	O	Test Data Output Serial output data is shifted from the Instruction or data registers to the *TDO* pin on the falling edge of the *TCK* clock. When no data is shifted out, the *TDO* is 3-stated. The core signal for this is called *EJ_TDO*.
TRST_N	I	Test Reset Input (Optional pin) The *TRST_N* pin is an active-low signal for asynchronous reset of the TAP controller and instruction in the TAP module, independent of the processor logic. The processor is not reset by the assertion of *TRST_N*. The core signal for this is called *EJ_TRST_N* This signal is optional, but power-on reset must apply a low pulse on this signal at power-on and then leave it high, in case the signal is not available as a pin on the chip. If available on the chip, then it must be low on the board when the EJTAG debug features are unused by the probe.

9.8.2 Test Access Port Operation

The TAP controller is controlled by the Test Clock (*TCK*) and Test Mode Select (*TMS*) inputs. These two inputs determine whether an Instruction Register scan or Data Register scan is performed. The TAP consists of a small controller, driven by the *TCK* input, which responds to the *TMS* input as shown in the state diagram in figure below. The TAP uses both clock edges of *TCK*. *TMS* and *TDI* are sampled on the rising edge of *TCK*, while *TDO* changes on the falling edge of *TCK*.

Figure 9.1 TAP Controller State Diagram



At power-up the TAP is forced into the *Test-Logic-Reset* by low value on *TRST_N*. The TAP instruction register is thereby reset to IDCODE. No other parts of the EJTAG hardware are reset through the *Test-Logic-Reset* state.

When test access is required, a protocol is applied via the *TMS* and *TCK* inputs, causing the TAP to exit the *Test-Logic-Reset* state and move through the appropriate states. From the *Run-Test/Idle* state, an Instruction register scan or a data register scan can be issued to transition the TAP through the appropriate states shown in figure below.

The states of the data and instruction register scan blocks are mirror images of each other adding symmetry to the protocol sequences. The first action that occurs when either block is entered is a capture operation. For the data registers, the *Capture-DR* state is used to capture (or parallel load) the data into the selected serial data path. In the Instruction register, the *Capture-IR* state is used to capture status information into the Instruction register.

From the *Capture* states, the TAP transitions to either the *Shift* or *Exit1* states. Normally the *Shift* state follows the *Capture* state so that test data or status information can be shifted out for inspection and new data shifted in. Following the *Shift* state, the TAP either returns to the *Run-Test/Idle* state via the *Exit1* and *Update* states or enters the *Pause* state via *Exit1*. The reason for entering the *Pause* state is to temporarily suspend the shifting of data through either the Data or Instruction Register while a required operation, such as refilling a host memory buffer, is performed. From the Pause state shifting can resume by re-entering the *Shift* state via the *Exit2* state or terminate by entering the *Run-Test/Idle* state via the *Exit2* and *Update* states.

Upon entering the data or Instruction register scan blocks, shadow latches in the selected scan path are forced to hold their present state during the Capture and Shift operations. The data being shifted into the selected scan path is not output through the shadow latch until the TAP enters the *Update-DR* or *Update-IR* state. The *Update* state causes the shadow latches to update (or parallel load) with the new data that has been shifted into the selected scan path.

9.8.2.1 Test-Logic-Reset State

In the Test-Logic-Reset state the boundary scan test logic is disabled. The test logic enters the Test-Logic-Reset state when the TMS input is held HIGH for at least five rising edges of TCK. The BYPASS instruction is forced into the instruction register output latches during this state. The controller remains in the Test-Logic-Reset state as long as TMS is HIGH.

9.8.2.2 Run-Test/Idle State

The controller enters the Run-Test/Idle state between scan operations. The controller remains in this state as long as TMS is held LOW. The instruction register and all test data registers retain their previous state. The instruction cannot change when the TAP controller is in this state.

When TMS is sampled HIGH on the rising edge of TCK, the controller transitions to the Select_DR state.

9.8.2.3 Select_DR_Scan State

This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state. If TMS is sampled LOW at the rising edge of TCK, then the controller transitions to the Capture_DR state.

A HIGH on TMS causes the controller to transition to the Select_IR state. The instruction cannot change while the TAP controller is in this state.

9.8.2.4 Select_IR_Scan State

This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state. If TMS is sampled LOW on the rising edge of TCK, the controller transitions to the Capture_IR state. A HIGH on *TMS* causes the controller to transition to the *Test-Reset-Logic* state. The instruction cannot change while the TAP controller is in this state.

9.8.2.5 Capture_DR State

In this state the boundary scan register captures the value of the register addressed by the Instruction register, and the value is then shifted out in the *Shift_DR*. If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Shift_DR* state. A HIGH on *TMS* causes the controller to transition to the *Exit1_DR* state. The instruction cannot change while the TAP controller is in this state.

9.8.2.6 Shift_DR State

In this state the test data register connected between *TDI* and *TDO* as a result of the current instruction shifts data one stage toward its serial output on the rising edge of *TCK*. If *TMS* is sampled LOW on the rising edge of *TCK*, the controller remains in the *Shift_DR* state. A HIGH on *TMS* causes the controller to transition to the *Exit1_DR* state. The instruction cannot change while the TAP controller is in this state.

9.8.2.7 Exit1_DR State

This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state. If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Pause_DR* state. A HIGH on *TMS* causes the controller to transition to the *Update_DR* state which terminates the scanning process. The instruction cannot change while the TAP controller is in this state.

9.8.2.8 Pause_DR State

The *Pause_DR* state allows the controller to temporarily halt the shifting of data through the test data register in the serial path between *TDI* and *TDO*. All test data registers selected by the current instruction retain their previous state.

If *TMS* is sampled LOW on the rising edge of *TCK*, the controller remains in the *Pause_DR* state. A HIGH on *TMS* causes the controller to transition to the *Exit2_DR* state. The instruction cannot change while the TAP controller is in this state.

9.8.2.9 Exit2_DR State

This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state. If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Shift_DR* state to allow another serial shift of data. A HIGH on *TMS* causes the controller to transition to the *Update_DR* state which terminates the scanning process. The instruction cannot change while the TAP controller is in this state.

9.8.2.10 Update_DR State

When the TAP controller is in this state the value shifted in during the *Shift_DR* state takes effect on the rising edge of the *TCK* for the register indicated by the Instruction register.

If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Run-Test/Idle* state. A HIGH on *TMS* causes the controller to transition to the *Select_DR_Scan* state. The instruction cannot change while the TAP controller is in this state and all shift register stages in the test data registers selected by the current instruction retain their previous state.

9.8.2.11 Capture_IR State

In this state the shift register contained in the Instruction register loads a fixed pattern (00001_2) on the rising edge of *TCK*. The data registers selected by the current instruction retain their previous state.

If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Shift_IR* state. A HIGH on *TMS* causes the controller to transition to the *Exit1_IR* state. The instruction cannot change while the TAP controller is in this state.

9.8.2.12 Shift_IR State

In this state the instruction register is connected between *TDI* and *TDO* and shifts data one stage toward its serial output on the rising edge of *TCK*. If *TMS* is sampled LOW at the rising edge of *TCK*, the controller remains in the *Shift_IR* state. A HIGH on *TMS* causes the controller to transition to the *Exit1_IR* state.

9.8.2.13 Exit1_IR State

This is a temporary controller state in which all registers retain their previous state. If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Pause_IR* state. A HIGH on *TMS* causes the controller to transition to the *Update_IR* state which terminates the scanning process. The instruction cannot change while the TAP controller is in this state and the instruction register retains its previous state.

9.8.2.14 Pause_IR State

The *Pause_IR* state allows the controller to temporarily halt the shifting of data through the instruction register in the serial path between *TDI* and *TDO*. If *TMS* is sampled LOW at the rising edge of *TCK*, the controller remains in the *Pause_IR* state. A HIGH on *TMS* causes the controller to transition to the *Exit2_IR* state. The instruction cannot change while the TAP controller is in this state.

9.8.2.15 Exit2_IR State

This is a temporary controller state in which the instruction register retains its previous state. If *TMS* is sampled LOW at the rising edge of *TCK*, then the controller transitions to the *Shift_IR* state to allow another serial shift of data. A HIGH on *TMS* causes the controller to transition to the *Update_IR* state which terminates the scanning process. The instruction cannot change while the TAP controller is in this state.

9.8.2.16 Update_IR State

The instruction shifted into the instruction register takes effect on the rising edge of *TCK*.
If *TMS* is sampled LOW at the rising edge of *TCK*, the controller transitions to the *Run-Test/Idle* state. A HIGH on *TMS* causes the controller to transition to the *Select_DR_Scan* state.

9.8.3 Test Access Port (TAP) Instructions

The TAP Instruction register allows instructions to be serially input into the device when TAP controller is in the *Shift-IR* state. Instructions are decoded and define the serial test data register path that is used to shift data between *TDI* and *TDO* during data register scanning.
The Instruction register is a 5-bit register. In the current EJTAG implementation only some instructions have been decoded; the unused instructions default to the BYPASS instruction.

Table 9-4 Implemented EJTAG Instructions

Value	Instruction	Function
0x01	IDCODE	Select Chip Identification data register
0x03	IMPCODE	Select Implementation register
0x08	ADDRESS	Select Address register

0x09	DATA	Select Data register
0x0A	CONTROL	Select EJTAG Control register
0x0B	ALL	Select the Address, Data and EJTAG Control registers
0x0C	EJTAGBOOT	Set EjtagBrk, ProbEn and ProbTrap to 1 as reset value
0x0D	NORMALBOOT	Set EjtagBrk, ProbEn and ProbTrap to 0 as reset value
0x1F	BYPASS	Bypass mode

9.8.3.1 BYPASS Instruction

The required BYPASS instruction allows the processor to remain in a functional mode and selects the Bypass register to be connected between *TDI* and *TDO*. The BYPASS instruction allows serial data to be transferred through the processor from *TDI* to *TDO* without affecting its operation. The bit code of this instruction is defined to be all ones by the IEEE 1149.1 standard. Any unused instruction is defaulted to the BYPASS instruction.

9.8.3.2 IDCODE Instruction

The IDCODE instruction allows the processor to remain in its functional mode and selects the Device Identification (ID) register to be connected between *TDI* and *TDO*. The Device ID register is a 32-bit shift register containing information regarding the IC manufacturer, device type, and version code. Accessing the Identification Register does not interfere with the operation of the processor. Also, access to the Identification Register is immediately available, via a TAP data scan operation, after power-up when the TAP has been reset with on-chip power-on or through the optional *TRST_N* pin.

9.8.3.3 IMPCODE Instruction

This instruction selects the Implementation register for output, which is always 32 bits.

9.8.3.4 ADDRESS Instruction

This instruction is used to select the Address register to be connected between *TDI* and *TDO*. The EJTAG Probe shifts 32 bits through the *TDI* pin into the Address register and shifts out the captured address via the *TDO* pin.

9.8.3.5 DATA Instruction

This instruction is used to select the Data register to be connected between *TDI* and *TDO*. The EJTAG Probe shifts 32 bits of *TDI* data into the Data register and shifts out the captured data via the *TDO* pin.

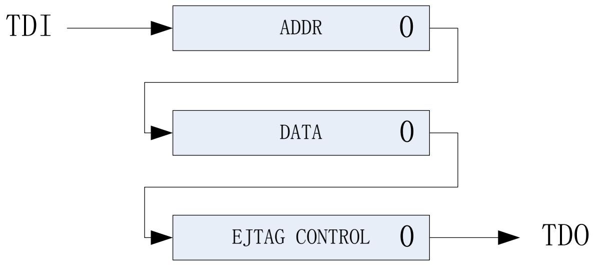
9.8.3.6 CONTROL Instruction

This instruction is used to select the EJTAG Control register to be connected between *TDI* and *TDO*. The EJTAG Probe shifts 32 bits of *TDI* data into the EJTAG Control register and shifts out the EJTAG Control register bits via *TDO*.

9.8.3.7 ALL Instruction

This instruction is used to select the concatenation of the Address and Data register, and the EJTAG Control register (ECR) between *TDI* and *TDO*. It can be used in particular to minimize the overhead in switching the instruction in the instruction register. The first bit shifted out is bit 0 of the ECR.

Figure 9.2 Concatenation of the EJTAG Address, Data and Control Registers



9.8.3.8 EJTAGBOOT Instruction

EJTAGBOOT provides a means to enter debug mode just after a reset, without fetching or executing any instructions from the normal memory area. This can be used for download of code to a system which has no code in ROM.

When the EJTAGBOOT instruction is given and the Update-IR state is left, the EJTAGBOOT indication will become active. When EJTAGBOOT is active, a core reset will set the ProbTrap, ProbEn and EjtagBrk bits in the EJTAG Control register to 1. This will cause a debug exception that is serviced by the probe immediately after reset is deasserted.

This EJTAGBOOT indication is effective until a NORMALBOOT instruction is given, *TRST_N* is asserted or a rising edge of *TCK* occurs when the TAP controller is in Test-Logic-Reset state.

Each has its own TAP controller and thus EJTAGBOOT can be set independently per .

The Bypass register is selected when the EJTAGBOOT instruction is given.

9.8.3.9 NORMALBOOT Instruction

When the NORMALBOOT instruction is given and the Update-IR state is left, then the EJTAGBOOT indication will be cleared. When NORMALBOOT is active (EJTAGBOOT is not active), a core reset will set the ProbTrap, ProbEn, and EjtagBrk bits in the EJTAG Control register to 0.

The Bypass register is selected when the NORMALBOOT instruction is given.

9.8.4 TAP Processor Accesses

The TAP modules support handling of fetches, loads and stores from the CPU through the dmseg segment, whereby the TAP module can operate like a *slave unit* connected to the on-chip bus. The core can then execute code taken from the EJTAG Probe and it can access data (via a load or store) which is located on the EJTAG Probe. This occurs in a serial way through the EJTAG interface: the core can thus execute instructions e.g. debug monitor code, without occupying the memory.

Accessing the dmseg segment (EJTAG memory) can only occur when the processor accesses an address in the range from 0xFF20.0000 to 0xFF2F.FFFF, the ProbEn bit is set, and the processor is in debug mode (DM=1). In addition the LSNM bit in the CP0 Debug register controls transactions to/from the dmseg.

When a debug exception is taken, while the ProbTrap bit is set, the processor will start fetching instructions from address 0xFF20.0200.

A pending processor access can only finish if the probe writes 0 to PrAcc or by a reset.

9.8.4.1 Fetch/Load and Store From/To the EJTAG Probe Through dmseg

1. The internal hardware latches the requested address into the Address register (in case of the Debug exception:0xFF20.0200).

2. The internal hardware sets the following bits in the EJTAG Control register:

    PrAcc = 1 (selects Processor Access operation)

    PRnW = 0 (selects Processor Read operation)

    Psz[1:0] = value depending on the transfer size

3. The EJTAG Probe selects the EJTAG Control register, shifts out this control register's data and tests the PrAcc status bit (Processor Access): when the PrAcc bit is found 1, it means that the requested address is available and can be shifted out.

4. The EJTAG Probe checks the PRnW bit to determine the required access.

5. The EJTAG Probe selects the Address register and shifts out the requested address.

6. The EJTAG Probe selects the Data register and shifts in the instruction corresponding to this address.

7. The EJTAG Probe selects the EJTAG Control register and shifts a PrAcc = 0 bit into this register to indicate to the processor that the instruction is available.

8. The instruction becomes available in the instruction register and the processor starts executing.

9. The processor increments the program counter and outputs an instruction read request for the next instruction. This starts the whole sequence again.

Using the same protocol, the processor can also execute a load instruction to access the EJTAG Probe's memory. For this to happen, the processor must execute a load instruction (e.g. a LW, LH, LB) with the target address in the appropriate range.

Almost the same protocol is used to execute a store instruction to the EJTAG Probe's memory through dmseg. The store address must be in the range: 0xFF20.0000 to 0xFF2F.FFFF, the ProbEn bit must be set and the processor has to be in debug mode (DM=1). The sequence of actions is found below:

1. The internal hardware latches the requested address into the Address register

2. The internal hardware latches the data to be written into the Data register.

3. The internal hardware sets the following bits in the EJTAG Control register:

    PrAcc = 1 (selects Processor Access operation)

    PRnW = 1 (selects processor write operation)

    Psz[1:0] = value depending on the transfer size

4. The EJTAG Probe selects the EJTAG Control register, shifts out this control register's data and tests the PrAcc status bit (Processor Access): when the PrAcc bit is found 1, it means that the requested address is available and can be shifted out.

5. The EJTAG Probe checks the PRnW bit to determine the required access.

6. The EJTAG Probe selects the Address register and shifts out the requested address.

7. The EJTAG Probe selects the Data register and shifts out the data that was written.

8. The EJTAG Probe selects the EJTAG Control register and shifts a PrAcc = 0 bit into this register to indicate to the processor that the write access is finished.

9. The EJTAG Probe writes the data to the appropriate address in its memory.

10. The processor detects that PrAcc bit = 0, which means that it is ready to handle a new access.

The above examples imply that no reset occurs during the operations, and that Rocc is cleared.

9.9 EJTAG Registers

The following subsections describe the EJTAG register interface.

9.9.1 General Purpose Control and Status

The following register provide general control and status information for EJTAG.

9.9.1.1 Debug Control Register

The Debug Control Register (DCR) controls and provides information about debug issues. The width of the register is 32 bits. The DCR is located in the drseg segment at offset 0x0.

The Debug Control Register (DCR) provides the following key features:

- Interrupt control when in Non-Debug Mode
- Availability indicator of instruction and data hardware breakpoints

The DataBrk and InstBrk bits within the DCR indicate the types of hardware breakpoints implemented. Debug software is expected to read hardware breakpoint registers for additional information on the number of implemented breakpoints.

Hardware and software interrupts can be disabled in Non-Debug Mode using the DCR’s IntE bit. This bit is a global interrupt enable used along with several other interrupt enables that enable specific mechanisms. The NMI interrupt is not implemented. Hardware and software interrupts are always disabled in Debug Mode. Pending interrupts are indicated in the Cause register.

The ProbEn bit reflects the state of the ProbEn bit from the EJTAG Control register (ECR). Through this bit, the probe can indicate to the debug software running on the CPU if it expects to service dmseg segment accesses.

Figure 9.3 shows the format of the DCR register; Table 9-5 describes the DCR register fields.

Figure 9.3 DCR Register Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="2">0</td><td>ENM</td><td colspan="11">0</td><td>DataBrk</td><td>InstBrk</td><td>IVM</td><td>DVM</td><td colspan="8">0</td><td>PCSe</td><td>IntE</td><td colspan="3">0</td><td>ProbEn</td></tr>
</table>

Table 9-5 DCR Register Field Descriptions

Name	Bits	Description	R/W	Reset
0	31:30	Must be written as zero; returns zero on read.	R	0
ENM	29	Endianess in which the processor is running in kernel and Debug Mode: 0: Little endian 1: Big endian	R	0
0	28:18	Must be written as zero; returns zero on read.	R	0
DataBrk	17	Indicates if data hardware breakpoint is implemented: 0: No data hardware breakpoint implemented	R	1

		1: Data hardware breakpoint implemented		
InstBrk	16	Indicates if instruction hardware breakpoint is implemented: 0: No instruction hardware breakpoint implemented 1: Instruction hardware breakpoint implemented	R	1
IVM	15	Indicates if inverted data value match on data hardware breakpoints is implemented: 0: No inverted data value match on data hardware breakpoints implemented 1: Inverted data value match on data hardware breakpoints implemented	R	0
DVM	14	Indicates if a data value store on a data value breakpoint match is implemented: 0: No data value store on a data value breakpoint match implemented 1: Data value store on a data value breakpoint match implemented	R	1
0	13:5	Must be written as zero; returns zero on read.	R	0
IntE	4	Hardware and software interrupt enable for Non-Debug Mode, in conjunction with other disable mechanisms: 0: Interrupt disabled. 1: Interrupt enabled depending on other enabling mechanisms.	R/W	1
0	3:1	Must be written as zero; returns zero on read.	R	0
ProbEn	0	Indicates value of the ProbEn value in the DCR register: 0: No access should occur to the dmseg segment 1: Probe services accesses to the dmseg segment	R	Same value as ProbEn in ECR

9.9.2 Instruction Breakpoint Registers

The registers for instruction breakpoints are described below. These registers have implementation information and are used to set up the instruction breakpoints. All registers are in drseg with addresses as shown in below. n is breakpoint number in range 0 to 1.

Table 9-6 Addresses for Instruction Breakpoint Registers

Offset in drseg	Register Mnemonic	Register Name and Description
0x1000	IBS	Instruction Breakpoint Status
0x1100 + n*0x100	IBAn	Instruction Breakpoint Address n
0x1108 + n*0x100	IBMn	Instruction Breakpoint Address Mask n

0x1110 + n*0x100	IBASIDn	Instruction Breakpoint ASID n
0x1118 + n*0x100	IBCn	Instruction Breakpoint Control n

An example of some of the registers; *IBA0* is at offset 0x1100 and *IBC2* is at offset 0x1318.

9.9.2.1 Instruction Breakpoint Status (IBS) Register

The Instruction Breakpoint Status (*IBS*) register holds implementation and status information about the instruction breakpoints.

The ASID applies to all the instruction breakpoints.

Figure 9.4 IBS Register Format

31	30	29 28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2	1 0
0	ASIDsup	0	BCN	0	BS

Table 9-7 IBS Register Field Descriptions

Name	Bits	Description	R/W	Reset
ASIDsup	30	Indicates if ASID compare is supported in instruction breakpoints: 0: No ASID compare 1: ASID compare (IBASIDn register implemented) ASID support indication does not guarantee a TLB-type MMU, because the same breakpoint implementation can be used with processors having all different types of MMUs.	R	1
0	29:28	Must be written as zero; returns zero on read.		
BCN	27:24	Number of instruction breakpoints implemented: 0: Reserved 1-15: Number of instructions breakpoints	R	2
BS	1:0	Break Status (BS) bit for breakpoint n is at BS[n], where n is 0 to 1. A bit is set to 1 when the condition for its corresponding breakpoint has matched. The number of BS bits implemented corresponds to the number of breakpoints indicated by the BCN field. Debug software is expected to clear the bits before use, because reset does not clear these bits. Bits not implemented are read-only (R) and read as zeros.	R/W0	0
0	31, 29:28, 23:2	Must be written as zero; returns zero on read.	R	0

9.9.2.2 Instruction Breakpoint Address n (IBAn) Register

The Instruction Breakpoint Address *n* (*IBAn*) register has the address used in the condition for instruction breakpoint *n*.

Figure 9.5 IBAn Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

IBA

Table 9-8 IBAn Register Field Description

Name	Bits	Description	R/W	Reset
IBA	31:0	Instruction breakpoint address for condition.	R/W	Undefined

9.9.2.3 Instruction Breakpoint Address Mask n (IBMn) Register

The Instruction Breakpoint Address Mask *n* (*IBMn*) register has the mask for the address compare used in the condition for instruction breakpoint *n*.

Figure 9.6 IBMn Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

IBM

Table 9-9 IBMn Register Field Description

Name	Bits	Description	R/W	Reset
IBM	31:0	Instruction breakpoint address mask for condition: 0: Corresponding address bit compared. 1: Corresponding address bit masked.	R/W	Undefined

9.9.2.4 Instruction Breakpoint ASID n (IBASIDn) Register

This register is used to define an ASID value to be used in the match expression.

Figure 9.7 IBASIDn Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	ASID

Table 9-10 IBASIDn Register Field Description

Name	Bits	Description	R/W	Reset
0	31:8	Must be written as zero; returns zero on read.		
ASID	7:0	Instruction breakpoint ASID value for compare.	R/W	Undefined

9.9.2.5 Instruction Breakpoint Control n (IBCn) Register

The Instruction Breakpoint Control *n* (*IBCn*) register controls the setup of instruction breakpoint *n*.

Figure 9.8 IBCn Register Format

31 30 29 28 27 26 25 24	23	22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3	2	1	0
0	ASIDuse	0	TE	0	BE

Table 9-11 IBCn Register Field Description

Name	Bits	Description	R/W	Reset
ASIDuse	23	Use ASID value in compare for instruction breakpoint *n*: 0: do note use ASID value in compare. 1: Use ASID value in compare.	R/W	0
TE	2	Use instruction breakpoint *n* as triggerpoint: 0: do not use it as triggerpoint. 1: use it as triggerpoint.	R/W	0
BE	0	Use instruction breakpoint *n* as breakpoint: 0: do not use it as breakpoint. 1: use it as breakpoint.	R/W	0
0	31:24, 22:3, 1	Must be written as zero; returns zero on read.	R	0

9.9.3 Data Breakpoint Registers

The registers for data breakpoints are described below. These registers have implementation information and are used to setup the data breakpoints. All registers are in drseg, and the addresses are shown in below. n is breakpoint number in range 0 to 1.

Table 9-12 Addresses for Data Breakpoint Registers

Offset in drseg	Register Mnemonic	Register Name and Description
0x2000	DBS	Data Breakpoint Status
0x2100 + n*0x100	DBAn	Data Breakpoint Address n
0x2108 + n*0x100	DBMn	Data Breakpoint Address Mask n
0x2110 + n*0x100	DBASIDn	Data Breakpoint ASID n
0x2118 + n*0x100	DBCn	Data Breakpoint Control n
0x2120 + n*0x100	DBVn	Data Breakpoint Value n
0x2ff0	LDV	Load Data Value

An example of some of the registers; *DBM0* is at offset 0x2108 and *DBV1* is at offset 0x2220.

9.9.3.1 Data Breakpoint Status (DBS) Register

The Data Breakpoint Status (*DBS*) register holds implementation and status information about the data breakpoints.

The ASIDsup field indicates whether ASID compares are supported.

Figure 9.9 DBS Register Format

31	30	29	28	27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2	1 0
0	ASIDsup	NoSVmatch	NoLVmatch	BCN	0	BS

Table 9-13 DBS Register Field Description

Name	Bits	Description	R/W	Reset
ASIDsup	30	Indicates if ASID compare is supported in instruction breakpoints: 0: No ASID compare 1: ASID compare (IBASIDn register implemented) ASID support indication does not guarantee a TLB-type MMU, because the same breakpoint implementation can be used with processors having all different types of MMUs.	R	1
NoSVmatch	29	Indicates if a value compare on a store is supported in data breakpoints: 0: data value and address in condition on store. 1: address compare only in condition on store.	R	0
NoLVmatch	28	Indicates if a value compare on a load is supported in data breakpoints: 0: data value and address in condition on load. 1: address compare only in condition on load.	R	0

BCN	27:24	Number of data breakpoints implemented: 0: Reserved 1-15: Number of instructions breakpoints	R	2
BS	1:0	Break Status (BS) bit for breakpoint n is at BS[n], where n is 0 to 1. A bit is set to 1 when the condition for its corresponding breakpoint has matched. The number of BS bits implemented corresponds to the number of breakpoints indicated by the BCN field. Debug software is expected to clear the bits before use, because reset does not clear these bits. Bits not implemented are read-only (R) and read as zeros.	R/W0	0
0	31, 23:2	0	R	0

9.9.3.2 Data Breakpoint Address n (DBAn) Register

The Data Breakpoint Address n (*DBAn*) register has the address used in the condition for data breakpoint n.

Figure 9.10 DBAn Register Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="32">DBA</td></tr>
</table>

Table 9-14 DBAn Register Field Description

Name	Bits	Description	R/W	Reset
DBA	31:0	Data breakpoint virtual address for condition.	R/W	Undefined

9.9.3.3 Data Breakpoint Address Mask n (DBMn) Register

The Data Breakpoint Address Mask n (*DBMn*) register has the mask for the address compare used in the condition for data breakpoint n.

Figure 9.11 DBMn Register Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="32">DBM</td></tr>
</table>

Table 9-15 DBMn Register Field Description

Name	Bits	Description	R/W	Reset
DBM	31:0	Data breakpoint address mask for condition: 0: Corresponding address bit compared. 1: Corresponding address bit masked.	R/W	Undefined

9.9.3.4 Data Breakpoint ASID n (DBASIDn) Register

For processors with a TLB based MMU, this register is used to define an ASID value to be used in the match expression.

Figure 9.12 DBASIDn Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	ASID

Table 9-16 DBASIDn Register Field Description

Name	Bits	Description	R/W	Reset
ASID	7:0	Data Breakpoint ASID value for compare.	R/W	Undefined
0	31:8	0	R	0

9.9.3.5 Data Breakpoint Control n (DBCn) Register

The Data Breakpoint Control *n* (*DBCn*) register controls the setup of data breakpoint *n*. In the P5600 core, this register is 64 bits and is accessed as two consecutive 32-bit registers.

Figure 9.13 DBCn Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	ASIDuse	0	BAI	NoSB	NoLB	0	BLM	0	TE	0	BE

Table 9-17 DBCn Register Field Description

Name	Bits	Description	R/W	Reset
ASIDuse	23	Use ASID value in compare for data breakpoint *n*: 0: do note use ASID value in compare. 1: Use ASID value in compare.	R/W	0

BAI	17:14	Byte access ignore. Each bit of this field determines whether a match occurs on an access to a specific byte of the database (BAI[0] controls matching for data bus bits 7:0; BAI[1] controls matching for data bus bits 15:8, etc.)., with the polarity of each bit, as follows: 0: condition depends on access to corresponding byte 1: access for corresponding byte is **ignored**	R/W	0
NoSB	13	Controls whether condition for data breakpoint is ever fulfilled on a store access: 0: Condition can be fulfilled on store access 1: Condition is never fulfilled on store access	R/W	0
NoLB	12	Controls whether condition for data breakpoint is ever fulfilled on a load access: 0: Condition can be fulfilled on load access 1: Condition is never fulfilled on load access	R/W	0
BLM	7:4	Byte lane mask for value compare on data breakpoint. BLM[0] masks byte at bits [7:0] of the data bus, BLM[1] masks byte at bits [15:8], etc.: 0 : Compare corresponding byte lane 1: Mask corresponding byte lane Debug software must adjust for endianess when programming this field.	R/W	0
TE	2	Use data breakpoint *n* as triggerpoint: 0: do not use it as triggerpoint. 1: use it as triggerpoint.	R/W	0
BE	0	Use data breakpoint *n* as breakpoint: 0: do not use it as breakpoint. 1: use it as breakpoint.	R/W	0
0	31:24, 22:18, 11:8, 3, 1	Must be written as zero; returns zero on read.	R	0

9.9.3.6 Data Breakpoint Value n (DBVn) Register

The Data Breakpoint Value n (*DBVn*) register has the value used in the condition for data breakpoint n.

Figure 9.14 DBVn Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DBV

Table 9-18 DBVn Register Field Description

Name	Bits	Description	R/W	Reset
DBV	31:0	Data breakpoint data value for condition. Debug software must adjust for endianess when programming this field.	R/W	Undefined

9.9.3.7 Load Data Value Register

The Load Data Value Register has the value when a Load Data Breakpoint matches.

Figure 9.15 LDV Register Format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LDV																															

Table 9-19 LDV Register Field Description

Name	Bits	Description	R/W	Reset
LDV	31:0	Load data when a data breakpoint is fulfilled on a load.	R/W	Undefined

9.9.4 EJTAG TAP Registers

The EJTAG TAP Module has one Instruction register and a number of data registers, all accessible through the TAP.

9.9.4.1 Instruction Register

The Instruction register is accessed when the TAP receives an Instruction register scan protocol. During an Instruction register scan operation the TAP controller selects the output of the Instruction register to drive the *TDO* pin. The shift register consists of a series of bits arranged to form a single scan path between *TDI* and *TDO*. During an Instruction register scan operations, the TAP controls the register to capture status information and shift data from *TDI* to *TDO*. Both the capture and shift operations occur on the rising edge of *TCK*. However, the data shifted out from the *TDO* occurs on the falling edge of *TCK*. In the Test-Logic-Reset and *Capture-IR* state, the instruction shift register is set to 00001_2, as for the IDCODE instruction. This forces the device into the functional mode and selects the Device ID register. The Instruction register is 5 bits wide. The instruction shifted in takes effect for the following data register scan operation.

9.9.4.2 Data Registers Overview

The EJTAG uses several data registers that are arranged in parallel from the primary *TDI* input to the primary *TDO* output. The Instruction register supplies the address that allows one of the data registers to be accessed during a data register scan operation. During a data register scan operation, the addressed scan register receives TAP control signals to capture the register and shift data from *TDI* to *TDO*. During a data register scan operation, the TAP selects the output of the data register to drive the *TDO* pin. The register is updated in the *Update-DR* state with respect to the write bits.

This description applies in general to the following data registers:

- Bypass Register
- Device Identification Register
- Implementation Register
- EJTAG Control Register (ECR)
- Address Register
- Data Register

9.9.4.3 Bypass Register

The Bypass register is a one-bit read-only register, which provides a minimum shift path through the TAP. When selected, the Bypass register provides a single bit scan path between *TDI* and *TDO*. The Bypass register allows abbreviating the scan path through devices that are not involved in the test. The Bypass register is selected when the Instruction register is loaded with a pattern of all ones to satisfy the IEEE 1149.1 Bypass instruction requirement.

9.9.4.4 Device Identification (ID) Register

The *Device Identification* register is defined by IEEE 1149.1, to identify the device's manufacturer, part number, revision, and other device-specific information. Table below shows the bit assignments defined for the read-only Device Identification Register, and inputs to the core determine the value of these bits. These bits can be scanned out of the *ID* register after being selected. The register is selected when the Instruction register is loaded with the IDCODE instruction. Note that this register contains only device manufacturer information and should not be used in an attempt to determine the EJTAG revisions of the device.

Figure 9.16 Device Identification Register Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="4">Version</td><td colspan="16">PartNumber</td><td colspan="11">ManufID</td><td>0</td></tr>
</table>

Table 9-20 Device Identification Register Field Description

Name	Bits	Description	R/W	Reset
Version	31:28	**Version** (4 bits) This field identifies the version number of the processor derivative.	R	0
PartNumber	27:12	Part Number (16 bits) This field identifies the part number of the processor	R	0

		derivative.		
ManufID	11:1	Manufacturer Identity (11 bits) Accordingly to IEEE 1149.1-1990, the manufacturer identity code shall be a compressed form of the JEDEC Publications 106-A.	R	0
0	0	0	R	0

9.9.4.5 Implementation Register

This 32-bit read-only register is used to identify the features of the EJTAG implementation. Some of the reset values are set by inputs to the core. The register is selected when the Instruction register is loaded with the IMPCODE instruction. The EJTAG probe uses this TAP register to determine the EJTAG version of the device. Software has no access to this register and must use the CP0 Debug register to determine the EJTAG version.

Figure 9.17 Implementation Register Format

31 30 29	28	27 26 25	24	23	22 21	20 19 18 17	16	15	14	13 12 11 10 9 8 7 6 5 4 3 2 1	0
EJTAGver	R4k/R3k	0	DINTsup	0	ASIDsize	0	MIPS16e	0	NoDMA	0	MIPS32/64

Table 9-21 Implementation Register Field Description

Name	Bits	Description	R/W	Reset
EJTAGver	31:29	Indicates the EJTAG version: 0: Version 1 and 2.0 1: Version 2.5 2: Version 2.6 3: Version 3.1 4: Version 4.0 5: Version 5.0 6-7: Reserved	R	0
DINTsup	24	Indicates support for DINT signal from probe: 0: DINT signal from the probe is not supported by this processor. 1: Probe can use DINT signal to make debug interrupt on this processor.	R	1
ASIDsize	22:21	Indicates size of the ASID field: 0: No ASID in implementation 1: 6-bit ASID 2: 8-bit ASID 3: Reserved	R	1

MIPS16e	16	Indicates MIPS16e™ ASE support in the processor: 0: No MIPS16e support 1: MIPS16e is supported	R	0
NoDMA	14	Indicates no EJTAG DMA support: 0: Reserved 1: No EJTAG DMA support	R	0
MIPS32/64	0	Indicates 32-bit or 64-bit processor: 0: 32-bit processor 1: 64-bit processor	R	0
0	28:25, 23, 20:17, 13:1	Must be written as zero; returns zero on read.	R	0

9.9.4.6 EJTAG Control Register

This 32-bit register controls the various operations of the TAP modules. This register is selected by shifting in the CONTROL instruction. Bits in the EJTAG Control register can be set/cleared by shifting in data; status is read by shifting out the contents of this register. This EJTAG Control register can only be accessed by the TAP interface.

The EJTAG Control register is not updated in the *Update-DR* state unless the Reset occurred (Rocc) bit 31, is either 0 or written to 0. This is in order to ensure proper handling of processor accesses.

The value used for reset indicated in the table below takes effect on CPU resets, but not on TAP controller resets (e.g. *TRST_N*). *TCK* clock is not required when the CPU reset occurs, but the bits are still updated to the reset value when the *TCK* is supplied. The first 5 *TCK* clocks after CPU reset may result in reset of the bits, due to synchronization between clock domains.

Figure 9.18 EJTAG Control Register Format

31	30 29	28 27 26 25 24 23	22	21	20	19	18	17	16	15	14	13	12	11 10 9 8 7 6 5 4	3	2 1 0
Rocc	Psz	0	Doze	Halt	PerRst	PrnW	PrAcc	0	PrRst	ProbEn	ProbTrap	0	EjtagBrk	0	DM	0

Table 9-22 EJTAG Control Register Field Description

Name	Bits	Description	R/W	Reset
Rocc	31	Reset Occurred The bit indicates if a CPU reset has occurred: The Rocc bit will remain set to 1 as long as reset is applied. This bit must be cleared by the probe to acknowledge	R/W	1

<table>
<tr><td></td><td></td><td>that the incident was detected.
The EJTAG Control register is not updated in the <i>Update-DR</i> state unless Rocc is 0 or written to 0, in order to ensure proper handling of processor access following reset.</td><td></td><td></td></tr>
<tr><td>Psz</td><td>30</td><td>Processor Access Transfer Size
These bits are used in combination with the lower two address bits of the Address register to determine the size of a processor access transaction. The bits are only valid when processor access is pending.
<table>
<tr><th>PAA[1:0]</th><th>Psz[1:0]</th><th>Transfer Size</th></tr>
<tr><td>00</td><td>00</td><td>Byte (LE, byte 0; BE, byte 3)</td></tr>
<tr><td>01</td><td>00</td><td>Byte (LE, byte 1; BE, byte 2)</td></tr>
<tr><td>10</td><td>00</td><td>Byte (LE, byte 2; BE, byte 1)</td></tr>
<tr><td>11</td><td>00</td><td>Byte (LE, byte 3; BE, byte 0)</td></tr>
<tr><td>00</td><td>01</td><td>Halfword (LE, byte 1:0; BE, byte 3:2)</td></tr>
<tr><td>10</td><td>10</td><td>Halfword (LE, byte 3:2; BE, byte 1:0)</td></tr>
<tr><td>00</td><td>11</td><td>Word(LE, BE: bytes 3, 2, 1, 0)</td></tr>
<tr><td colspan="2">All others</td><td>Reserved</td></tr>
</table>
Note: LE=little endian, BE=big endian, the byte# refers to the byte number in a 32-bit register, where byte 3 = bits 31:24; byte 2 = bits 23:16; byte 1 = bits 15:8; byte 0=bits 7:0, independently of the endianess.</td><td>R</td><td>Undefined</td></tr>
<tr><td>Doze</td><td>22</td><td>Doze state
The Doze bit indicates any type of low-power mode. The value is sampled in the Capture-DR state of the TAP controller:
0: CPU not in low power mode
1: CPU is in low power mode</td><td>R</td><td>0</td></tr>
<tr><td>Halt</td><td>21</td><td>Halt state
The Halt bit indicates if the internal system bus clock is running or stopped. The value is sampled in the Capture-DR state of the TAP controller:
0: Internal system clock is running
1: Internal system clock is stopped</td><td>R</td><td>0</td></tr>
<tr><td>PerRst</td><td>20</td><td>Peripheral reset
This bit has no effect.</td><td>R</td><td>0</td></tr>
<tr><td>PrnW</td><td>19</td><td>Processor Access Read and Write
This bit indicates if the pending processor access is</td><td>R</td><td>Undefined</td></tr>
</table>

		for a read or write transaction, and the bit is only valid while *PrAcc* is set: 0: Read transaction 1: Write transaction		
PrAcc	18	Processor Access (PA) Read value of this bit indicates if a Processor Access (PA) to the EJTAG memory is pending: 0: No pending processor access 1: Pending processor access The probe's software must clear this bit to 0 to indicate the end of the PA. A write of 1 is ignored. A pending Processor Access is cleared when *Rocc* is set, but another PA may occur just after the reset if a debug exception occurs. Finishing a Processor Access is not accepted while the *Rocc* bit is set. This is to avoid a Processor Access occurring after the reset is finished because of an indication of a Processor Access that occurred before the reset.	R/W0	0
PrRst	16	Processor Reset (Implementation dependent behavior) When the bit is set to 1, then it is only guaranteed that this setting has taken effect in the system when the read value of this bit is also 1. This is to ensure that the setting from the *TCK* clock domain gets effect in the CPU clock domain, and in peripherals. When the bit is written to 0, then the bit must also be read as 0 before it is guaranteed that the indication is cleared in the CPU clock domain also. This bit controls the *EJ_PrRst* signal. If the signal is used in the system, then it must be ensured that both the processor and all devices required for a reset are properly reset. Otherwise the system may fail or hang. The bit resets itself, since the *EJTAG Control* register is reset by a reset.	R/W	0
ProbEn	15	Probe Enable This bit indicates to the CPU if the EJTAG memory is handled by the probe so processor accesses are answered: 0: Probe does not handle EJTAG memory transactions 1: Probe does handle EJTAG memory transactions It is an error by the software controlling the probe if it	R/W	0 or 1 from EJTAGBOOT

		sets the Prob-Trap bit to 1, but resets the *ProbEn* to 0. The operation of the processor is UNDEFINED in this case. The ProbEn bit is reflected as a read-only bit in the ProbEn bit, bit 0, in the Debug Control Register (DCR). The read value indicates the effective value in the DCR, due to synchronization issues between *TCK* and CPU clock domains; however, it is ensured that change of the ProbEn prior to setting the EjtagBrk bit will have effect for the debug handler executed due to the debug exception. The reset value of the bit depends on whether the EJTAGBOOT indication is given or not: No EJTAGBOOT indication given: 0 EJTAGBOOT indication given: 1		
ProbTrap	14	Probe Trap This bit controls the location of the debug exception vector: 0: In normal memory. 1: In EJTAG memory at 0xFF20.0200 in dmseg Valid setting of the ProbTrap bit depends on the setting of the ProbEn bit, see comment under ProbEn bit. The ProbTrap should not be set to 1 unless the ProbEn bit is also set to 1 to indicate that the EJTAG memory may be accessed. The read value indicates the effective value to the CPU, due to synchronization issues between *TCK* and CPU clock domains; however, it is ensured that change of the ProbTrap bit prior to setting the EjtagBrk bit will have effect for the EjtagBrk. The reset value of the bit depends on whether the EJTAGBOOT indication is given or not.	R/W	0 or 1 from EJTAGBOOT
EjtagBrk	12	EJTAG Break Setting this bit to 1 causes a debug exception to the processor, unless the CPU was in debug mode or another debug exception occurred. When the debug exception occurs, the processor core clock is restarted if the CPU was in low power mode. This bit is cleared by hardware when the debug exception is taken. The reset value of the bit depends on whether the	R/W	0 or 1 from EJTAGBOOT

		EJTAGBOOT indication is given or not: 0: No EJTAGBOOT indication given 1: EJTAGBOOT indication given		
DM	3	Debug Mode This bit indicates the debug or non-debug mode: 0: Processor is in non-debug mode 1: Processor is in debug mode The bit is sampled in the *Capture-DR* state of the TAP controller.	R	0
0	28:23, 17, 13, 11:4, 2:0	Must be written as zero; returns zero on read.	R	0

9.9.4.7 Processor Access Address Register

The Address register is used to provide the address of the processor access in the dmseg, and the register is only valid when a processor access is pending. The length of the Address register is 32 bits, and this register is selected by shifting in the ADDRESS instruction.

Figure 9.19 Processor Access Address Register Format

<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="32">Address</td></tr>
</table>

Table 9-23 Processor Access Address Register Field Description

Name	Bits	Description	R/W	Reset
Address	31:0	Address used by processor access.	R	Undefined

9.9.4.8 Processor Access Data Register

The Data register is used to provide data value to and from a processor access. The length of the Data register is 32 bits, and this register is selected by shifting in the DATA instruction.

The register has the written value for a processor access write due to a CPU store to the dmseg, and the output from this register is only valid when a processor access write is pending. The register is used to provide the data value for a processor access read due to a CPU load or fetch from the dmseg. The register will be updated with a new value when a processor access write is pending.

The Data register is 32 bits wide. Data alignment is not used for this register, so the value in the Data register matches data on the internal bus. The unused bytes for a processor access write are undefined, and for a Data register read, 0(zero) must be shifted in for the unused bytes.

The bytes in the Data Register are organized in little-endian.

The size of the transaction and thus the number of bytes available/required for the Data register is determined by the Psz field in the *ECR*.

Figure 9.20 Processor Access Data Register Format

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Data

Table 9-24 Processor Access Data Register Field Description

Name	Bits	Description	R/W	Reset
Data	31:0	Data used by processor access.	R/W	Undefined

9.10 Debug Exception

This section describes issues related to debug exceptions. Debug exceptions bring the processor from Non-Debug Mode into Debug Mode.

Exceptions can occur in Debug Mode, and these are denoted as debug mode exceptions. These exceptions are handled differently from exceptions that occur in Non-Debug Mode.

9.10.1 Debug Exception Priorities

Table bellow lists the exceptions that can occur in Non-Debug Mode in order of priority, from highest to lowest. The table also categorizes each exception with respect to type (debug or non-debug). Each debug exception has an associated status bit in the Debug register (indicated in the table in parentheses).

Though in the following table Debug Data Break on Load Address and Data Value have a lower priority than Watch on Data Access, the Debug Data Break on Load with Data Value has a higher priority than Watch on Data. Putting the Debug Data Break on Load with Data Value at the last is because one can not detect a Debug Data Break on Load with Data Value if the load trigger a TLB related exception.

Table 9-25 Priority of Debug and Non-Debug Exceptions

Priority	Exception	Type of Exception
Highest	Reset	Non-Debug
	Debug Single Step	Debug
	Debug Interrupt; from EJTAG TAP.	
	Machine Check	Non-Debug
	Interrupt	
	Deferred Watch	
	Debug Instruction Break	Debug
	Watch on Instruction Fetch	Non-Debug
	Address Error on Instruction Fetch	

	TLB Refill on Instruction Fetch	
	TLB Invalid on Instruction Fetch	
	Debug Breakpoint; execution of SDBBP.	Debug
	Other execution-based exceptions	Non-Debug
	Debug Data Break on Load/Store address match only	Debug
	Debug Data Break on Store address + data value match	
	Watch on Data Access	Non-Debug
	Address Error on Data Access	
	TLB Refill on Data Access	
	TLB Invalid on Data Access	
	TLB Modify on Data Access	
Lowest	Debug Data Break on Load address + data value match	Debug

9.10.2 Debug Exception Vector Location

The same vector is used for all debug exceptions. The vector location can be controlled from the TAP through the EJTAG Control Register (ECR) ProbTrap bit.

Table 9-26 Debug Exception Vector Location

$\text{ECR}_{\text{ProbTrap}}$	Debug Exception Vector Address
0	0xBFC00480
1	0xFF200200 in dmseg

9.10.3 General Debug Exception Processing

All debug exceptions have the same basic processing flow:

- The DEPC register is loaded with the PC at which execution can be restarted, and the DBD bit is set to indicate whether the last debug exception occurred in a branch delay slot. The value loaded into the DEPC register is either the current PC (if the instruction is not in the delay slot of a branch) or the PC of the branch or jump (if the instruction is in the delay slot of a branch or jump).
- The DSS, DBp, DDBL, DDBS, DIB, and DINT bits in the Debug register are updated appropriately depending on the debug exception.
- DExcCode field in the Debug register is unchanged.
- IEXI bit is set to inhibit imprecise exceptions in the start of the debug handler.
- DM bit in the Debug register is set to 1.
- The processor begins fetching instructions from the debug exception vector.

The value loaded into the DEPC register represents the restart address from the debug exception and does not need to be modified by the debug exception handler software. Debug software need only look at the DBD bit in the Debug register to identify the address of the instruction that actually caused a precise debug exception.

The DSS, DBp, DDBL, DDBS, DIB, and DINT bits in the Debug register indicate the occurrence of distinct debug exceptions. Note that the occurrence of an exception while in Debug mode will clear these bits. The handler can thereby determine whether a debug exception or an exception in Debug Mode occurred.

Also note that multiple cause bits may be set, but the priority of the debug exception or interrupt dictates the order in which they are handled. For example, because DSS is the highest priority Debug exception, if it occurs, it will always be taken first. Then, after it DERETS, other debug exceptions can be taken. For example, assume that the processor is in single-step mode in a branch delay slot, and waiting to go past the delay slot to enter the DSS exception. At the branch delay slot, it could get a DINT or other lower priority Debug exception. In this case, it would not take the lower exception, but enter Debug Mode past the delay slot. The entry into Debug Mode will clear the DINT. It would process the single-step exception and DERET to normal non-debug mode. Note that in practice, not many cores set multiple cause bits in the Debug register since the highest priority debug exception is taken, and the others are cleared on entry to Debug Mode.

No other CP0 registers or fields are changed due to the debug exception, thus no additional state is saved.

9.10.4 Debug Single Step Exception

When single-step mode is enabled, a Debug Single Step exception occurs each time the processor has taken a single execution step in Non-Debug Mode. An execution step is a single instruction, or an instruction pair consisting of a jump/branch instruction and the instruction in the associated delay slot. The SSt bit in the Debug register enables Debug Single Step exceptions. They are disabled on the first execution step after a DERET.

The DEPC register points to the instruction on which the Debug Single Step exception occurred, which is also the next instruction to execute when returning from Debug Mode. The debug software can examine the system state before this instruction is executed. Thus the DEPC will not point to the instruction(s) that have just executed in the execution step, but rather the instruction following the execution step. The Debug Single Step exception never occurs on an instruction in a jump/branch delay slot, because the jump/branch and the instruction in the delay slot are always executed in one execution step; thus the DBD bit in the Debug register is never set for a Debug Single Step exception.

Exceptions occurring on the instruction(s) in the execution step are taken regardless, so if a non-debug exception occurs (other than reset), a Debug Single Step exception is taken on the first instruction in the non-debug exception handler. The non-debug exception occurs during the execution step, and the instruction(s) that received a non-debug exception counts as the execution step.

Debug exceptions are unaffected by single-step mode; returning to an SDBBP instruction with single step enabled causes a Debug Breakpoint exception with the DEPC register pointing to the SDBBP instruction. Also, returning to an instruction (not jump/branch) just before the SDBBP instruction causes a Debug Single Step exception with the DEPC register pointing to the SDBBP instruction.

To ensure proper functionality of single-step execution, the Debug Single Step exception has priority over all exceptions, except reset.

Debug Single Step exception is only possible when the NoSSt bit in the Debug register is 0.

Debug Register Debug Status Bit Set

DSS

Additional State Saved

None

Entry Vector Used

Debug exception vector

9.10.5 Debug Interrupt Exception

The Debug Interrupt exception is an asynchronous debug exception that is taken as soon as possible, but with no specific relation to the executed instructions. The DEPC register and the DBD bit in the Debug register reference the instruction at which execution can be resumed after Debug Interrupt exception service.

Debug interrupt requests are ignored when the processor is in Debug Mode, and pending requests are cleared when the processor takes any debug exception, including debug exceptions other than Debug Interrupt exceptions.

A debug interrupt restarts the pipeline if stopped by a WAIT instruction and the processor clock is restarted if it was stopped due to a low-power mode.

The sources for debug interrupts are only from EJTAG TAP.

Debug Register Debug Status Bit Set

DINT

Additional State Saved

None

Entry Vector Used

Debug exception vector

9.10.6 Debug Instruction Break Exception

A Debug Instruction Break exception occurs when an instruction hardware breakpoint matches an executed instruction. The DEPC register and DBD bit in the Debug register indicate the instruction that caused the instruction hardware breakpoint match.

Debug Register Debug Status Bit Set

DIB

Additional State Saved

None

Entry Vector Used

Debug exception vector

9.10.7 Debug Breakpoint Exception

A Debug Breakpoint exception occurs when an SDBBP instruction is executed. The contents of the DEPC register and the DBD bit in the Debug register indicate that the SDBBP instruction caused the debug exception.

Debug Register Debug Status Bit Set

DBp

Additional State Saved

None

Entry Vector Used

Debug exception vector

9.10.8 Debug Data Break on Load/Store Exception

A Debug Data Break Load/Store exception occurs when a data hardware breakpoint matches the load/store address of an executed load/store instruction. The DEPC register and DBD bit in the Debug register indicate the load/store instruction that caused the data hardware breakpoint to match, as this is a precise debug exception. The load/store instruction that caused the debug exception has not completed (it has not updated the destination register or memory location), and the instruction therefore is executed on return from the debug handler.

Debug Register Debug Status Bit Set

DDBL for a load instruction or DDBS for a store instruction

Additional State Saved

None

Entry Vector Used

Debug exception vector

9.11 Debug Mode Exceptions

The handling of exceptions generated in Debug Mode, other than through resets, differs from those exceptions generated in Non-Debug Mode in that only the Debug and DEPC registers are updated. All other CP0 registers are unchanged by an exception taken in Debug Mode. The exception vector is equal to the debug exception vector, and the processor stays in Debug Mode.

Reset and soft reset are handled as when occurring in Non-Debug Mode.

9.11.1 Exceptions Taken in Debug Mode

Only some Non-Debug Mode exception events cause exceptions in Debug Mode. Remaining events are blocked. Exceptions occurring in Debug Mode have the same relative priorities as the Non-Debug Mode exceptions for the same exception event. These exceptions are called Debug Mode <Non-Debug Mode exception name>. For example, a Debug Mode Breakpoint exception is caused by execution of a BREAK instruction in Debug Mode, and a Debug Mode Address Error exception is caused by an address error due to an instruction executed in Debug Mode.

Table bellow lists all the Debug Mode exceptions with their corresponding non-debug exception event names, priorities, and handling.

Table 9-27 Exception Handling in Debug Mode

Priority	Exception	Type of Exception
Highest	Reset	Handled as for Non-Debug Mode
	Debug Single Step	Blocked
	Debug Interrupt; from EJTAG TAP.	

	Machine Check	Re-enter Debug Mode
	Interrupt	Blocked
	Deferred Watch	
	Debug Instruction Break	
	Watch on Instruction Fetch	
	Address Error on Instruction Fetch	Re-enter Debug Mode
	TLB Refill on Instruction Fetch	
	TLB Invalid on Instruction Fetch	
	Debug Breakpoint; execution of SDBBP.	Re-enter Debug Mode as for execution of the BREAK instruction
	Other execution-based exceptions: Syscall, Break, CpU, RI, Ov, Tr, FPE, MSAFPE, MSADis	Re-enter Debug Mode
	Debug Data Break on Load/Store address match only	Blocked
	Debug Data Break on Store address + data value match	
	Watch on Data Access	
	Address Error on Data Access	Re-enter Debug Mode
	TLB Refill on Data Access	
	TLB Invalid on Data Access	
	TLB Modify on Data Access	
Lowest	Debug Data Break on Load address + data value match	Blocked

Exceptions that are blocked in Debug Mode are simply ignored, not causing updates in any state.

9.11.2 Debug Mode Exception Processing

All exceptions that are allowed in Debug Mode (except for reset) have the same basic processing flow:

- The DEPC register is loaded with the PC at which execution can be restarted, and the DBD bit is set to indicate whether the last debug exception occurred in a branch delay slot. The value loaded into the DEPC register is either the current PC (if the instruction is not in the delay slot of a branch) or the PC of the branch or jump (if the instruction is in the delay slot of a branch or jump).
- The DSS, DBp, DDBL, DDBS, DIB, and DINT bits in the Debug register are all cleared to differentiate from debug exceptions where at least one of the bits are set.
- The DExcCode field in the Debug register is updated to indicate the type of exception that occurred.
- The IEXI bit is set to inhibit imprecise exceptions at the start of the debug handler.
- The DM bit in the Debug register is unchanged, leaving the processor in Debug Mode.
- The processor is started at the debug exception vector.

The value loaded into the DEPC register represents the restart address for the exception; typically debug software does not need to modify this value at the location of the debug exception. Debug software need not look at the DBD bit in the Debug register unless it wishes to identify the address of the instruction that actually caused the exception in Debug Mode.

It is the responsibility of the debug handler to save the contents of the Debug, DEPC, and DESAVE registers before nested entries into the handler at the debug exception vector can occur. The handler returns to the debug exception handler by a jump instruction, not a DERET, in order to keep the processor in Debug Mode.

The cause of the exception in Debug Mode is indicated through the DExcCode field in the Debug register, and the same codes are used for the exceptions as those for the ExcCode field in the Cause register when the exceptions with the same names occur in Non-Debug Mode, with addition of the code 30 (decimal) with the mnemonic CacheErr for cache errors.

No other CP0 registers or fields are changed due to the exception in Debug Mode. The write of TS bit is suppressed when the machine check exception occurs in Debug mode.

9.12 MIPS EJTAG Compliant Mode

XBurst2 Core implemented two Debug Modes, MIPS EJTAG compliant mode and Accelerated Mode(ACC Mode).

For more details of MIPS EJTAG compliant mode, please refer to MIPS EJTAG Specification.

9.13 Accelerated EJTAG Mode

This section describes the behavior and organization of Accelerated EJTAG Mode(ACC Mode). ACC Mode is designed to accelerate the data access from probe in debug mode. Debugger can use EJTAG instruction EJTAGBOOTA to enter into ACC mode, and exit ACC mode by execution of NORMALBOOT or a TAP reset.

The overall features of ACC Mode are:

- Address range 0xFF00 0000 to 0xFF1F FFFF and 0xFF40 0000 to 0xFFFF FFFF are extended as dmseg and they are both Unmapped and Cacheable that one can use burst read to accelerate dmseg read but write still is not bust.
- Bit1-0 of ECR are defined as PRW and PA to indicated a access in ACC Mode?

9.13.1 ACC Mode Flag

AM: 1 – ACC mode; 0 – MIPS mode.

Its reset value is 0. It is invisible for JTAG probe and software. The bit is set by expanded instruction EJTAGBOOTA, cleared by NORMALBOOT or TAP reset.

9.13.2 EJTAG Control Register in ACC mode (ECR_A)

It is connected between TDI and TDO by instruction CONTROL or ALL in ACC mode. Probe polls this 2-bit register to service the processor access to dmseg region.

1	0
PRW	PA

Field	BITS	Description	Read/write	Reset value
PA	0	Processor Access (PA) 0: No pending processor access 1: Pending processor access	R	0
PRW	1	Processor access is read or write 0: read access 1: write access	R	Undefined

9.13.3 Processor Access Address Register in ACC mode (ADDRESS_A)

It is connected between TDI and TDO by instruction ADDRESS or ALL in ACC mode. This register is used to provide the accessed address of dmseg region and more data bus operation information.

36 35	34	33 32	31 0
BST	R	SZ	PAA

Field	BITS	Description	R/W	Reset value
PAA	31: 0	Processor Access address	R	Undefined
SZ	33:32	Processor Access size 00: byte 01: half word 10: word 11: reserved	R	Undefined
R	34	Reserved		
BST	36:35	Processor Access burst pattern 00: single 01: 4-beat wrapping burst 10: 8-beat wrapping burst 11: 8-beat incrementing burst Note: 4-beat wrapping burst will never occur in this implementation. And 8-beat incrementing burst may occur only for burst read access.	R	Undefined

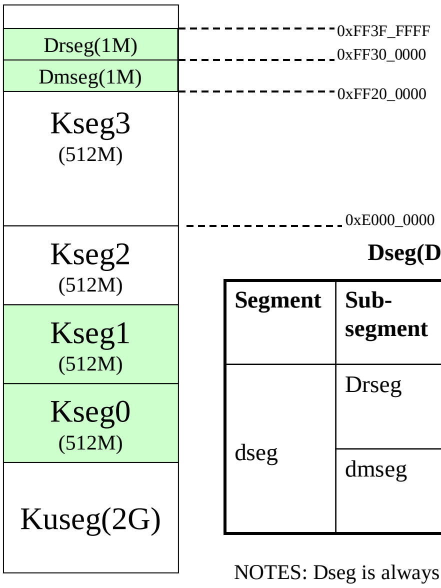
9.13.4 Processor Access Data Register in ACC mode (DATA_A)

It is connected between TDI and TDO by instruction DATA or ALL in ACC mode. This register is similar to PAD in MIPS mode, except it has one more RDY bit.

32	31 0
RDY	PAD

Field	BITS	Description	R/W	Reset value
PAD	31:0	Processor Access data The register has the written value for a write access to the dmseg. And it is also used to register the data value for load access or fetch instruction from the dmseg.	R/W	Undefined
RDY	1	Pipeline lock label. 1- processor can proceed due to processor access to dmseg done 0- processor should be locked due to unfinished processor access to dmseg	W	Undefined

9.13.5 Debug Mode Address Space in Compliant Mode (AM = 0)



unmapped 🟩

mapped □

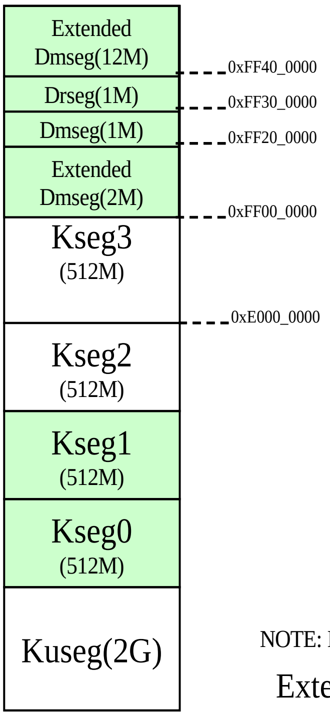
Dseg(Drseg + Dmseg) Address Space and Cache Attributes

Segment	Sub-segment	Virtual address	Physical address	Cache attribute
dseg	Drseg	0xFF30_0000 – 0xFF3F_FFFF	0xFF30_0000-0xFF3F_FFFF	2
	dmseg	0xFF20_0000 – 0xFF2F_FFFF	0xFF20_0000-0xFF2F_FFFF	2

NOTES: Dseg is always unmapped in debug mode in spite of attribute of Kseg3 .

Dseg space in MIPS mode

9.13.6 Debug Mode Address Space in ACC Mode (AM = 1)



unmapped 🟩

mapped □

Dseg (Drseg + Dmseg) Address Space and Cache Attributes

Segment	Sub segment	Virtual address	Physical address	Cache attribute
Dseg	extended Dmseg (2M)	0xFF00_0000-0xFF1F_FFFF	0xFF00_0000-0xFF1F_FFFF	0
	Dmseg (1M)	0xFF20_0000-0xFF2F_FFFF	0xFF20_0000-0xFF2F_FFFF	2
	Drseg (1M)	0xFF30_0000-0xFF3F_FFFF	0xFF30_0000-0xFF3F_FFFF	2
	extended Dmseg (12M)	0xFF40_0000-0xFFFF_FFFF	0xFF40_0000-0xFFFF_FFFF	0

NOTE: Dseg is always unmapped in debug mode in spite of attribute of Kseg3.

Extended Dseg space in ACC mode

9.13.7 Supported JTAG Instructions in ACC Mode

Value	Instruction	Function
0x01	IDCODE	Select Chip Identification data register.
0x03	IMPCODE	Select implementation register.
0x08	ADDRESS	ADDRESS register is selected in MIPS mode while ADDRESS_A register is selected in ACC mode.
0x09	DATA	DATA register is selected in MIPS mode while DATA_A register is selected in ACC mode.
0x0A	CONTROL	ECR register is selected in MIPS mode while ECR_A register is selected in ACC mode.
0x0B	ALL	In MIPS mode, Selects the ADDRESS, DATA and ECR register. The scan sequence is TDI-> ADDRESS-> DATA->ECR->TDO. In ACC mode, Selects the DATA_A, ADDRESS_A, and ECR_A register. The scan sequence is TDI-> DATA_A ->ADDRESS_A ->ECR_A->TDO.
0x0C	EJTAGBOOT	Boot from probe host in MIPS mode by setting ECREjtagbrk, ECRProbEn and ECRProbTrap when reset Bypass register is selected.
0x0D	NORMALBOOT	Boot in normal way by clearing Ejtagbrk, ProbEn and ProbTrap when reset Bypass register is selected.
0x1C	EJTAGBOOTA	Boot from probe host in ACC mode by setting ECREjtagbrk, ECRProbEn, ECRProbTrap and AM when reset. Bypass register is selected.
0x1F	BYPASS	Select Bypass register.

Revision History

Revision	Date	Description
00.00	July 6, 2016	● internal release
00.01	July 21, 2016	● Add the feature about SMT ● Update the MSIR, CNCR register of the CCU ● Add the description about Multi-core debug
00.02	September 18, 2016	~~● Add more hardware spin lock~~ ● Change CSCR into CCCR
00.03	June 2, 2017	● Update about debug description ● Update CCU's spin clock and spin atomic register
00.04	Auguest 30, 2018	● Revised MMU and Cache ● Part of CCU
00.10	April 20, 2020	● Remove sophisticated but efficentless EJTAG debug mechanism for SMP system ● Some syntax error fixed