LEXRA

LX4580 Data Sheet

Revision 3.1

October 11, 2002

Lexra, Inc.

LX4580 Data Sheet Revision 3.1 October 11, 2002

.

Table of Contents

List of Figures

List of Tables

Chapter 1. LX4580 Product Overview

1.1. Introduction

The LX4580, designed to support Lexra's family of network communication ICs, is the highest performance MIPS32™ processor optimized for "High-Touch" packet processing applications. Based on Lexra's 3rd generation 7-stage pipeline, the LX4580 is able to achieve 3x the performance of other 32-bit CPU platforms.

Enabling this technology break-through is Lexra's innovative fine-grained Hardware Multi-Threading (HMT). HMT dramatically improves system performance by its ability to alternate the execution of four hardware threads. The processor stays 100% utilized even while multiple threads are servicing cache misses. From a system perspective, the LX4580 overcomes the fundamental bottleneck to application performance - memory latency.

Additional benefits due to HMT are an increase in pipeline efficiency as well as removal of timing-critical forwarding paths. Stalls due to load-to-use are typically eliminated while commonly used features like branch prediction are altogether no longer needed. Collectively this results in die area saving and an increase in processor performance.

The LX4580 includes features like the Coprocessor Interface (CI) and User Defined Instruction (UDI) interface are used to specialize the processor to the target application.

The LX4580 supports the new MIPS32™ Release 2.0 ISA to leverage a rich legacy of tools, application software, and operating systems available for the MIPS architecture. With its optional Memory Management Unit (MMU) the LX4580 supports Linux® SMP running on the hardware threads, in single and multi-processor systems.

Software views the LX4580 as four virtual CPUs or hardware threads running in parallel. Cache coherency is maintained for data accesses among the multiple contexts. Applications using concurrent kernel threads or user processes will run on the LX4580 virtually unchanged while taking full advantage of HMT. Alternatively, a single threaded RTOS such as VxWorks® can run on one thread while other threads function as coprocessors.

The LX4580 CPU provides the ultimate in both performance and flexibility required to execute demanding high touch applications. The LX4580 CPU implements the MIPS32™ ISA, with additional specialized instructions for optimized packet processing. Peak processor performance is 700 Dhrystone 2.1 MIPS. The CPU incorporates Lexra's innovative fine-grained Hardware Multi-Threading (HMT) technology. As a result, high CPU performance can be sustained even while L1 cache misses are serviced.

The CPU includes a 64-bit MIPS EC^{tm} interface for connection to main memory and other peripherals. This interface provides access to a wide range of SoC interconnect and modules such as DRAM controllers, PCI and AMBA bridges and custom I/O interfaces. The optional CBUS interface provides an easy to use but powerful alternative for connection to proprietary or third party buses.

Target applications for the LX4580 include:

- **Enterprise Security Systems**
  These systems provide specialized services such as VPN, firewall and intrusion detection for traffic between the enterprise LAN or data center and WAN. The LX4580 can be used for either Linux-based application services or for "front-end" network processing in these systems.
- **Web Appliances**
  Typical products include Switches, Web Directors, Web Caches. The LX4580 provides consolidated functionality with strong Layer 4-7 decision making for policy and content-base load balancing, security, session and site persistence based on HTTP cookies, IP address, etc.

- **Network Attached Storage (NAS) Servers**
  New generation NAS Servers provide remote storage-on-demand while lowering administration costs and leverage the learning curve of 3rd party internet technology. The LX4580 provides TCP termination, iSCSI protocol conversion and security services.

1.2. Key Features

Processor

- 7-stage execution pipeline
- MIPS32™ Release 2.0 ISA
  - Specialized instructions for packet processing
- Lexra ISA extensions:
  - Hash with key size 4 to 24 bits
  - Dual 16-bit ones-complement add
- Fine-grained Hardware Multi-Threading (HMT)
  - Four hardware threads of execution
  - Unique register set, interrupts and TLB per thread

Local Memory Interface

- Up to 64 KB L1 Instruction Cache
- Up to 64 KB L1 Data Cache
- 4-way set associative
- Configurable cache line size (16, 32, 64 and 128 bytes)
- Conforms to MESI coherency protocol
- Write-through or write-back Data Cache
- Up to 1 MB fixed local instruction/data memory

System Bus Interface

- Optional CBUS or EC interfaces
- Split transaction (CBUS) or pipelined (EC)
- 64-bit data-path
- Multi-master support
- Burst mode

Memory Management Unit (MMU)

- 4 KB page size
- Supports 36-bit physical addresses
- 2-entry ITLB, 4-entry DTLB, 24-entry JTLB (per thread)

Multiply-Accumulate Unit (MAC)

- Two-cycle 32x32b multiply, multiply-accumulate

Enhanced JTAG (EJTAG) 2.0.0 Debug

- Single-stepping
- Address and data breakpoints
- Full-speed real-time PC trace
- Supports multi-processor debug

User Defined Instructions (UDI)

- Application-specific ALU instruction extensions
- Supports 3-register operand and 2-register/immediate formats

Coprocessors

- An application-specific coprocessor can be attached to the Coprocessor Interface (CI)

1.3. Specifications

- Technology: 0.13µm CMOS
- 500 MHz Processor Clock (worst-case)
- 3.8 mm^2 Core Area
- 360 mW Power (worst-case)
- Operating Temperature: 125° C junction
- Supply Voltage: 1.2 V nominal

1.4. LX4580 Architecture

1.4.1. LX4580 CPU

The LX4580 incorporates four (4) LX4580 CPUs, illustrated in Figure 1. The LX4580 is a complete RISC processor subsystem, optimized for high-performance packet processing.

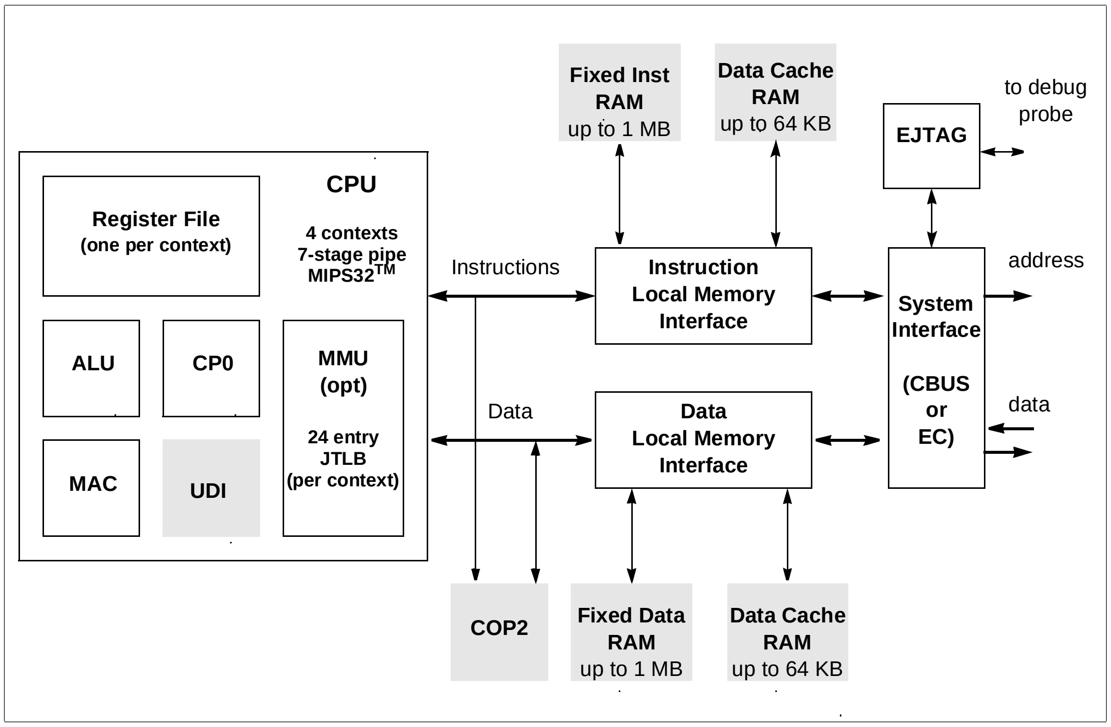
Fixed Inst RAM up to 1 MB
Data Cache RAM up to 64 KB
to debug probe
EJTAG
CPU
Register File (one per context)
4 contexts 7-stage pipe MIPS32™
Instructions
Instruction Local Memory Interface
System Interface (CBUS or EC)
address
ALU
CP0
MMU (opt) 24 entry JTLB (per context)
Data
Data Local Memory Interface
data
MAC
UDI
COP2
Fixed Data RAM up to 1 MB
Data Cache RAM up to 64 KB

Figure 1: LX4580 Diagram

The major blocks are the Register file and ALU (RALU), Coprocessor 0 (CP0), the local memory interfaces (LMI) to up to 64KB instruction cache and 6K4B data cache.

Lexra's LX4580 CPU implements the full Release 2 MIPS32™ instruction set. The MIPS32 *optional* and *recommended* features included in the CPU are detailed in Chapter 2. A number of implementation-specific issues are also documented in Chapter 2. Lexra has extended the MIPS32 ISA with additional instructions for optimized packet processing. These instructions are described in Chapter 2. The CPU includes an MMU and support 36-bit physical addresses.

The CPU execution pipeline is 7-stage and exclusively uses the rising edge of the processor clock. The 7-stage pipeline permits a full cycle for address register to data output register for both instruction cache read and data cache read. As a result the CPU pipeline achieves maximum performance for its implementation technology and methodology and will readily port to future technologies.

1.4.2. Fine-Grained Hardware Multi-Threading (HMT)

The LX4580 CPU incorporates Lexra's proprietary implementation of *fine-grained Hardware Multi-Threading (HMT)*. Although HMT is transparent to software it provides significant performance advantages to the LX4580 customer that deserve attention in this overview.

In Lexra's implementation, instructions are issued round-robin from four alternate contexts. Each context has an independent program counter and general register file. Other software visible state is also replicated as detailed in Chapter 2. In the absence of an L1 cache miss, the four contexts support four independent execution threads.

In typical network processing programs data cache miss rates are high and a single-threaded processor is idle much of the time. The problem can be mitigated somewhat with a second level cache, offering faster service time than main memory. However, in Lexra's implementation of HMT, as few as two active threads can maintain 100% CPU utilization while cache misses from the other two threads are being served.

Additional performance benefits from HMT result from the following:

- All timing-critical internal forwarding paths are eliminated. As a result, for a specific technology and design methodology, high processor clock speed is achieved.
- Branch prediction is not required. There are sufficient cycles between issue slots so that branch outcome can be correctly resolved without prediction. Other high-end RISC architectures have devoted significant silicon area and power to minimizing stalls from branch prediction failures.
- Load-to-use delay is minimized. The 7-stage pipeline would normally require two load-to-use delay cycles. With HMT, the load-to-use delay is zero or one cycle depending on the number of actively executing threads. As a result, the frequency of load interlock stalls is reduced.

Lexra's simulations indicate that if 3% of instructions cause an cache miss, HMT delivers a 3X performance benefit compared to a similar single issue CPU. This performance benefit can be realized in applications with sufficient thread parallelism. Assigning each thread one or more independent packet flows allows HMT to be fully exploited in the LX4580 target applications.

1.5. Interfaces

Table 1 summarizes the interfaces provided by the LX4580.

Table 1: Summary of LX4580 Interfaces

Name	Qty	Performance	Function
EC Interface[a]	1	500 MHz (CPU clock)	Interface option to system devices. See Chapter 6 and MIPS “EC^{tm} Interface Specification”, Revision 1.05.
CBUS Interface[a]	1	500 MHz (CPU clock)	Interface option to system devices. See Chapter 5.
CI	1	500 MHz (CPU clock)	Coprocessor interface (COP2). See XREF.
UDI	1	500 MHz (CPU clock)	User Defined Instructions. See XREF.
EJTAG	1	40 MHz clock.	Scan chain debug. Conforms to EJTAG 2.0. Provides PC-trace. Multi-processor support.

a. RTL is configurable to support EC interface or CBUS interface.

1.6. Software Support

1.6.1. Operating Systems

Two operating systems are provided for the LX4580:

- Linux®, version 2.4 and higher, with full SMP support running on all CPUs and hardware threads. Full source code for the Linux® kernel is available from MontaVista™ and the Hardware Abstraction Layer is available from Lexra and MontaVista™.
- VxWorks® version 5.4 running on one thread. The Hardware Abstraction Layer is available from Wind River.

1.6.2. Development Tools

Extensive On-Chip Debug Features

- EJTAG
- Performance Counters

Complete Development Platform

- C/C++ development tool chain
- Development board
- Device drivers and abstraction layer
- Sample code

2.1. MIPS32 Implementation Specifics Overview

The MIPS32 architecture defines certain features as optional (or recommended), in which case they may be completely omitted from a compliant implementation. Other MIPS32 features are defined as implementation dependent, in which case one or more choices must be supported for compliance. Finally there are optional extensions that an implementation may provide.

The purpose of this chapter is to detail the implementation dependent features of the LX4580 CPU. The specifics of each of the following areas is discussed in its own section:

- Instructions
- CP0 Registers
- Interrupts
- Exceptions
- Address Spaces
- Endianness
- EJTAG
- CP0 Hazards
- Release 2 Features

2.2. MIPS32 Instructions

This section describes implementation specific details of the following MIPS32 instructions:

- LL/SC
- SYNC
- PREF
- CACHE
- WAIT
- Divide (all variants)
- UDI

2.2.1. LL/SC

The unit of memory that is used to determine whether the SC should fail is one cache line. That is, after the LL, a write to any byte in the line by any other entity will cause the SC to fail. In addition:

- Any load, store or CACHE instruction between the LL and the SC by the same context, when not in debug mode, will cause the SC to fail.
- Any ERET between the LL and SC by the same context will cause the SC to fail.
- Any store to the cache line by a different context in the same CPU between the LL and SC will cause the SC to fail.
- A load or store between the LL and the SC by the same context in debug mode, may cause the SC to fail in rare instances. The precise conditions are described below.

The remaining implementation of this feature relies solely on the state of the cache line within the L1 data cache in the CPU as follows:

```
On LL, if cache miss, request line Shared
  else line is already Shared or Modified (okay)

On SC, if dcache miss.............................              SC fails
  else if already Modified........................              SC passes
    else request line upgrade to Modified
    if invalidated before request completes........             SC fails
      else ........................................             SC passes
```

Note that there is no architecturally visible CP0 LLAddr register.

For HMT, a variant of the LLAddr register (just the data cache Way and Index) is used for two purposes:

- Another context is not allowed to cause a replacement eviction of the line between the LL and SC. To prevent this, the particular data cache Way and Index (of the line used by the LL) are saved while the SC is pending or until it is guaranteed to fail, whichever comes first.
- Another context in the processor can store to the line, forcing the SC to fail. This is detected by the L1 data cache using the saved Way and Index.
- If all four contexts have a pending SC for the same Index (each to a different Way), then no Way of that Index is available for replacement eviction. Any load or store by any context that is not in debug mode, will enable a Way for eviction without impacting the other contexts because it can cause its own context SC to fail. However, a load or store in debug mode that requires a replacement eviction in the same Index will use the saved Way of the context that is executing in debug mode. This rare case will also cause the SC to fail for the context in question.

2.2.2. SYNC

There is only one outstanding data cache miss (for either loads or stores) at a time for each context.

An uncached load prevents further progress by a context until the load data returns.

Therefore all cached loads/stores and uncached loads are strongly ordered for any given context.

To cover the ordering of uncached stores, SYNC flushes uncached stores previously executed by the same context, preventing forward progress by context executing the SYNC until all such stores are Acked by their targets.

2.2.3. PREF

The instruction is treated as NOP.

2.2.4. CACHE

The following operations are supported:

• I	Index Invalidate
• D	Index WritebackInvalidate / Index Invalidate
• I,D	Index Store Tag
• I,D	Hit Invalidate
• D	Hit WritebackInvalidate / Hit Invalidate
• D	Hit Writeback

The following are not implemented:

• S,T	anything
• I	Fill
• I,D	Index Load Tag
• I,D	Fetch and Lock (there are no Locks in instruction or data cache)

Since the instruction and data caches are shared by all four contexts in the CPU, it is the responsibility of software to avoid conflicting CACHE instruction execution. Note that the Data cache Writeback operations and the Instruction cache Invalidate operations are generally safe across contexts since they do not discard potentially modified data. If the Store Tag operation is only used for initialization, that too should be safe to use. Finally, the Data cache Hit Invalidate should be used with caution since it discards data that may have been written by a context different than the one executing the CACHE instruction.

2.2.5. WAIT

Only Code 0 is supported.

When the WAIT instruction is executed by a context in the LX4580 CPU, that context is suspended from further execution. The only way to restart a context after completion of a WAIT instruction is with an enabled interrupt to that context. The EPC will point to the instruction after the WAIT.

Since the other contexts continue execution, the WAIT instruction does not cause the CPU clocks to stop nor are the CPU caches disabled. Any power savings from the WAIT instruction would be on a gate-level basis in that reduced pipeline activity would reduce the circuit switching current. The primary benefit of the WAIT instruction is to reduce contention among contexts for the CPU pipeline slots while one or more contexts are merely waiting for some external event. For this reason it is preferred to a software spin loop.

2.2.6. Divide (all variants)

The divider detects when the dividend has leading zeroes, reducing its latency in such cases.

2.2.7. UDI

The following User Defined Instructions are implemented:

- HASH rd, rt, keysize
  Hash to Key. The 5-bit keysize is a value k in the range 4-24. The 32 source bits contained in rt are hashed to form a key of k bits which is stored in rd[k-1:0]. The remaining bits of rd are zeroed. If k is not in the range 4-24, the results are unpredictable.

  Format: 31:26 011100 (Special2), 25:11 zero,rt,rd, 10:6 keysize - 1, 5:0 110000 (Hash)

- ACS2 rd, rs, rt
  Dual Add for Checksum. This instruction performs dual 16-bit ones complement addition. Considering all quantities as unsigned 16-bit integers, add rs[15:0] to rt[15:0] and independently add rs[31:16] to rt[31:16]. For each addition if there is a carry out of the most significant bit of its result, add one to that result to form its final result. The final results are stored in rd[15:0] and rd[31:16]

  Format: 31:26 011100 (Special2), 25:11 rs,rt,rd, 10:6 zero, 5:0 110001 (Acs2)

2.3. CP0 Registers

This section describes implementation specific details of the CP0 registers. In Table 2 each of the standard (MIPS32 Release 2) CP0 registers is listed, together with an indication if the register is not implemented. If it is implemented there may be details on how the implementation handles certain fields in the register. For registers that are implemented, the column labeled HMT indicates whether it is implemented independently for each context (4) or just once per CPU (1). In Table 3, the implementation specific CP0 registers are described. All of the implementation specific CP0 registers are implemented independently for each context under HMT except CVSTag and CXCtrl (although in CXCtrl the context bits in the CPUNum field are in fact unique by context).

Table 2: Standard CP0 Registers

Name	Num	Sel	HMT	Field	Implementation Specific Information
Index	0	0	4		6-bits
Random	1	0	4		see note[a]
EntryLo0,1	2,3	0	4		
				PFN	36-bit PA supported
				C	only values 2 or 3 supported
Context	4	0	4		
PageMask	5	0	4		only 4KB and 64MB pages, see note[b]
				MaskX	always 2#11 (no 1KB pages)
PageGrain	5	1			not implemented
Wired	6	0	4		
HWREna	7	0	4		
BadVAddr	8	0	4		
Count	9	0	1		counts cpu clocks
EntryHi	10	0	4		
Compare	11	0	4		

Table 2: Standard CP0 Registers (Continued)

Name	Num	Sel	HMT	Field	Implementation Specific Information
Status	12	0	4		
				CU321	always 0 (no FPU, no coprocessors)
				RE	always 0 (no ReverseEndian)
				RP,FR,MX,PX	always 0
				TS	always 0
				SR	always 0
				Impl	always 0
				KX,SX,UX	always 0
				R0	always 0
IntCtl	12	1	4		
				IPTI	always 7 (Timer interrupt in IP7)
				IPPCI	always 7 (PerfCnt interrupt in IP7)
				EIC, VS	always 0
SRSCtl	12	2	1		always 0
SRSMap	12	3			not implemented
Cause	13	0	4		
				DC	all contexts must set this to stop Count
				WP	always 0
				ExcCode	see note[c]
EPC	14	0	4		
PRId	15	0	1		(lx4580): 0x000bd101
EBase	15	1	4		
				CPUNum	same value as CXCtrl.CPUNum
Config	16	0	4		
				M	1
				BE	reset to 1 or 0 per config pin
				KU,K23	value 3 (for Fixed Mapping Table, when MT=3)
				AT	always 0
				AR	always 1
				MT	reset to 1 or 3 per config pin
				VI	always 0
				K0	reset to 2, may be written values 2 or 3

Table 2: Standard CP0 Registers (Continued)

Name	Num	Sel	HMT	Field	Implementation Specific Information
Config1	16	1	1		
				M	0
				MMU-1	(24 entry) 23
				IS,IL,IA	64-byte linesize, 64KB or 32KB or 16KB Icache size, 4 ways
				DS,DL,DA	64-byte linesize, 64KB or 32KB or 16KB Dcache size,4 ways
				C2,MD	always 0
				PC	1
				WR,CA	always 0
				EP	1
				FP	always 0
Config2	16	2			not implemented
Config3	16	3			not implemented
LLAddr	17	0			not implemented
WatchLo	18				not implemented
WatchHi	19				not implemented
Debug	23	0	4		
DEPC	24	0	4		
PerfCnt	25	0-7	1		4 counters with controls. See Section 2.10
ErrCtl	26	0	1		not implemented
CacheErr	27		1		not implemented
TagLo	28	0,2	1		always 0
DataLo	28	1,3			not implemented
TagHi	29	0,2			always 0
DataHi	29	1,3			not implemented
ErrorEPC	30	0	4		
DESAVE	31	0	1		

a. The Random register is decremented on every instruction completion. The two most recently used values by TLBWR in a TLBRefill exception are saved, and are never used in a subsequent TLBWR. TBD: an LFSR is used to further control the decrement of Random.

b. Only bits 26:25 of the Mask field in the PageMask register are implemented. Writing ones to these bits signifies a 64MB page. All other bit positions return zeroes on reads.

c. The Cause register ExcCode field can have the following values: Int, Mod, TLBL, TLBS, AdEL, AdES, IBE, DBE, Sys, Bp, RI, CpU, OV, Tr, CacheErr.
The Cause register ExcCode can never have the following values: FPE, C2E, MDMX, WATCH, MCheck.

Table 3: Implementation Dependent CP0 Registers

Name	Num	Sel	Bits	Field	Implementation Specific Information
CXCtrl	16	6			
			31	CXTaS	Test and Set bit. Is set to 1 after any read[a]
			30:24		0
			23:16	SW	Software usable Read/Write field
			15:12		0
			11:8	DC3:0	Disable context in this CPU
			7:5		0
			4:0	CPUNum	Same as EBASE.CPUNum[a]
CVSTag	16	7			
			31:0	CVSTag	ReadOnly for Lexra Internal Use

a. The CXTaS bit allows atomic updates to the remaining fields of the CXCtrl register. A context which reads this bit as one, should not update any fields. A context which reads this bit as zero should restore it to zero whether or not it updates other fields.

The CPUNum is a chip-wide unique identifier for the current thread of execution. The two least significant bits are the context number within the CPU. This value is also readable from the CPUNum field of the EBase CP0 register that is defined by MIPS32 Release 2 or, if enabled in User mode, using the Release 2 RDHWR instruction specifying the CPUNum hardware register. Using those other methods of obtaining CPUNum avoids the need to check and possibly clear CXTaS.

2.4. Interrupts

The MIPS32 architecture defines eight interrupts, which are visible as the interrupt pending bits IP7:0 in the CP0 Cause register. In Table 4 the source of each of these pending interrupts is indicated.

Table 4: Interrupt Sources

Interrupt	Definition	Generation
IP0	Software 0	Write to Cause IP0
IP1	Software 1	Write to Cause IP1
IP2	Hardware 0	CPU cross context interrupts (See Section 4.2.3)
IP3	Hardware 1	External interrupts (See Section 4.2.1)
IP4	Hardware 2	External interrupts (See Section 4.2.1)
IP5	Hardware 3	External interrupts (See Section 4.2.1)
IP6	Hardware 4	External interrupts (See Section 4.2.1)
IP7	Hardware 5	Logically OR Timer interrupt with Performance Counter interrupt.

2.5. Exceptions

All of the exceptions that are defined by the MIPS32 architecture are in Table 5. The relevant implementation specific aspects are indicated.

Table 5: Exception List

Exception		Implementation Specifics
Reset		
SoftReset		implemented like Reset
Debug SingleStep		
Debug Interrupt		
Imprecise DebugDataBreak		Loads with address+data match not implemented
NMI		External input pin
MachineCheck		not implemented
Interrupt		see Section 2.4, "Interrupts"
Deferred Watch		not implemented
Debug InstructionBreak		
Watch	Ifetch	not implemented
Address Error	Ifetch	
TLB Refill	Ifetch	
TLB Invalid	Ifetch	
Cache Error	Ifetch	not implemented
Bus Error	Ifetch	
SDBBP		
Coproc Unusable		
Reserved Inst		
Execution Exception		Overflow, Trap, BREAK, SYSCALL
Precise Debug DataBreak1		Loads with address match only. All stores
Watch		not implemented
Address Error	Data	
TLB Refill	Data	
TLB Invalid	Data	
TLB Modified	Data	
Cache Error	Data	not implemented
Bus Error	Data	
Precise Debug DataBreak2		not implemented (Loads with address+data match are always treated as Imprecise Debug DataBreak exceptions)

2.5.1. Reset Context Wait and EJBOOT

When the CPU is reset, only Context 0 is enabled. This is accomplished by the hardware initializing the value of the DC bits in the CXCTRL register so that all contexts other than 0 are disabled. It is the responsibility of the Reset handler that runs in Context 0 to enable the other contexts by clearing their DC bits. When its DC bit is cleared, each of the other contexts will begin execution of the Reset handler. As indicated in Table 2 each context has its own ErrorEPC (used to “return” from the reset exception) and each context can control where it begins execution after it completes the Reset handler.

As described in Section 3.3, the CPU can optionally begin execution at the time of reset by fetching instructions in debug mode from the EJTAG probe. From the CPU point of view, this functionality is similar to the EJBOOT feature of EJTAG 2.5. The extensions to EJTAG 2.0 that control this feature are described in Chapter 8. As in the case of all Reset exceptions, only Context 0 begins execution. Hence only Context 0 enters debug mode in this case. The other contexts begin execution at the standard Reset exception vector in normal mode after their DC bits are cleared.

2.5.2. DM Wait and EJTAG (Debug) Exceptions

The LX4580 CPU implements a DM Wait feature which prevents more than one context from executing in Debug mode at any given time. As noted in Table 2 there is only one instance of various CP0 registers (such as DESAVE) used to support Debug mode. Furthermore, the EJTAG probe software is unlikely to support intermixed accesses to the Dmseg and Drseg regions. Therefore, after one context begins executing in Debug Mode (due to an EJTAG exception) any other context which takes an EJTAG exception is placed in the DM Wait queue. While in the DM Wait state, the context does not issue any instructions. When the first context leaves Debug Mode (by executing a DERET instruction), the next context in the DM Wait queue resumes execution (in the EJTAG exception handler).

Furthermore, the EJTAG implementation for the LX4580 CPU has an additional feature which optionally allows an EJTAG exception in one context to immediately place all other contexts in the CPU into the DM Wait queue, suspending their execution. When the first context leaves Debug Mode (by executing its DERET), the other contexts resume execution (at whatever point they were suspended). This feature allows the EJTAG probe software to gain control of the entire CPU without needing to put all contexts into Debug Mode simultaneously.

An additional feature of the LX4580 CPU to be noted is that Debug Mode for a context overrides the DC bit for that context. This allows the EJTAG probe to force a context to enter Debug Mode (using the DINT EJTAG exception) even if the context is disabled for normal execution. It also prevents a context that is executing in Debug Mode from being disabled by another context, which could hang the EJTAG probe.

2.6. Address Spaces

Supervisor Mode is not supported.

Kseg2 is supported (instead of Ksseg).

36 Physical Address bits are supported.

The only Memory access types supported are values 2 (uncached) and 3 (cacheable).

Kseg0 can be either uncached or cacheable according to the K0 field of the CP0 Config register.

When the ERL field of the CP0 Status register has value 1, Kuseg is an unmapped, uncached segment and all 2**31 bytes are translated. This is the situation upon reset.

As noted in the CP0 Config register MT field, at reset, the TLB can be disabled in which case the Fixed Mapping Table will be used. In this case, as noted in the CP0 Config register KU,K23 field definitions, kuseg, kseg2 and kseg3 will always be cacheable (field value 3).

2.6.1. Non-Coherence for Different Access Types

The MIPS32 architecture specifies that results of loads or stores to a location using one memory access type that follow loads or stores to the same location using a different memory access type are unpredictable in general. The architecture states that an implementation specific sequence can enforce coherence between such accesses. For the LX4580 CPU, the only two access types are cacheable and uncached. By performing a CACHE instruction with the Hit Writeback Invalidate operation between the accesses, the coherence can be enforced. The address used for the CACHE instruction may have either the cacheable or uncached access type. This implies that the required CACHE instruction may use the address from either of the accesses so that it can be done immediately after the first access or immediately before the second access, in either case using the same base register and offset as the access in question.

2.7. Endianness

At reset BigEndian or LittleEndian is selected via an external configuration pin.

Reverse endianness is not supported. In MIPS32 Release 2 the WSBH instruction can be used when endian swap is needed. See Section 2.11.3.

2.8. EJTAG

The CPU generally supports the EJTAG 2.0 specification in a manner consistent with the MIPS32 architecture. The exceptions to this are in the following areas:

- PC Trace
- Data Break Exceptions
- HMT Extensions

For PC Trace, the EJTAG 2.0 concept of external trace signals is not supported. This is due to the higher speed of the CPU and the multi-context nature of the LX4580 CPU pipeline. Instead, an on-chip trace buffer is used to capture information about instruction execution. The controls for the trace buffer allow tracing of a single context or tracing of all contexts of the LX4580 CPU simultaneously. The trace buffer and associated controls are described Section 8.2.8.

For Debug Data Break exceptions, the CPU implements the concept of Precise Data Breaks that is defined in the EJTAG 2.5 specification. In particular, for Loads, only the address match applies. For Stores, both the address and data match (if enabled) apply. Imprecise Data Breaks, which would require data match for Loads, are not supported because the LX4580 CPU often resolves Loads for a given context in the background of execution of other contexts.

As noted in Section 2.5.2 only a single context of the LX4580 CPU is allowed to execute in Debug Mode at any given time. Furthermore, if the EJTAG Control Register “Disable Other Contexts” (DOC) bit is set when any context enters Debug Mode, all other contexts suspend execution. As indicated in Table 2 each context has its own CP0 Debug and DEPC registers to provide independent context control of EJTAG and to hold the DEPC for each context that is in DM Wait state. On the other hand, there is only one DESAVE register that is shared by all contexts since it is only needed during execution in Debug Mode.

For both Instruction and Data Breaks, the match logic is extended to include an optional match against the context number.

For EJTAG Breaks, an additional field in the EJTAG Control Register is used to indicate whether all contexts are to be interrupted, or just a specific context is to be interrupted.

2.9. CP0 Hazards

In all cases the implementation meets or exceeds the "typical" requirements for instruction spacing to avoid CP0 hazards as described in the MIPS32 architecture specification.

2.10. Performance Counters

The LX4580 CPU implements four performance counters, as noted in Table 2. Each counter can select from the same set of events to count, and each counter can count the selected event for all contexts, or for one particular context. The format of the counters and their control registers follows the MIPS32 Release 2 specification, with one Lexra extension (the CntxSel field) in bits 13:11 of the control registers, as defined in Table 6. The Event field (bits 10:5) of the MIPS32-specified counter control registers is defined in Table 7.

Because there is only one set of performance counters, they are shared by all contexts and it is the responsibility of software to control their use by more than one context, if so desired. One particular aspect of the MIPS32-specified control registers that is specific to the LX4580 CPU is the IE field, which indicates that a Performance Counter Interrupt should become pending under certain conditions. In the LX4580 CPU*, the IE field applies to the context that most recently wrote the control register in question*.

Table 6: CntxSel (bits 13:11) Field of PerfCnt Control Registers

Value (bits 13:11)	Context to Count
000	Count Events for all Contexts
100	Count Events for Context 0
101	Count Events for Context 1
110	Count Events for Context 2
111	Count Events for Context 3
others	reserved

Table 7: Event Field of PerfCnt Control Registers

Value (bits 10:5)	Event Counted
000000	retired instructions
000001	replayed instructions
000010	instruction fetch (valid new D-stage)
000011	Icache instruction fetch
000100	Icache miss
000101	Uncached instruction fetch

Table 7: Event Field of PerfCnt Control Registers

Value (bits 10:5)	Event Counted
000110	Dcache load[a]
000111	Dcache store[a]
001000	Dcache load miss
001001	Dcache store miss
001010	Dcache load or store[a]
001011	Dcache load or store miss
001100	Uncached load or store[a]
001101	Writeback for replacement
001110	Writeback for inquiry
001111	Invalidate for inquiry
010000	Nop for inquiry
010001	Pipeline stall for any reason
010010	Pipeline stall for Icache fill
010011	Pipeline stall for Dcache fill
010100	Pipeline stall for store from store queue
010101	Pipeline stall for write buffer full
010110	execution exception (Ov,Trap,BREAK,SYSCALL)
010111	TLB refill exception - Instruction
011000	TLB refill exception - Data
011001	TLB invalid exception - Instruction
011010	TLB invalid exception - Data
011011	TLB modified exception
011100	any TLB exception
011101	ITLB miss
011110	DTLB miss
011111	Interrupt
100001	any exception
100010	Store Conditional instruction (pass or fail)
100011	Store Conditional Fail
others	reserved (no count)

a. Counts replayed and retired versions. The number of replays does not significantly contribute to the overall count.

2.11. Release 2 Architecture Support

The LX4580 CPU supports the MIPS32 Release 2 Architecture Changes. Those changes include numerous optional and implementation dependent features as well as several required features. The following sections provide detail on the LX4580 implementation. As a quick summary, the following is a list of all of the Release 2 features and their support in the LX4580 CPU:

• Vectored Interrupts	(not supported)
• External Interrupt Controller	(not supported)
• Programmable Exception Vector Base	(supported)
• Atomic Interrupt Enable/Disable	(supported)
• Disable Count register	(supported)
• GPR Shadow Registers	(not supported)
• Field, Rotate, Shuffle Instructions	(supported)
• Hazard Barrier Instructions	(supported)
• User Hardware Register access	(supported)
• CP0 register changes	(supported)
• 64-bit FPU	(not supported)
• 1KB page size	(not supported)

2.11.1.Release 2 Interrupt Modes, Exceptions, Shadow GPRs

The Release 2 Architecture defines a Compatibility interrupt mode which is equivalent to the Release 1 interrupt mode. This is the only Release 2 interrupt mode supported by the LX4580 CPU. The Vectored and External Interrupt Controller (EIC) modes are not supported. GPR Shadow Registers are not supported.

As noted in Section 2.4, the Timer and Performance Counter interrupts are presented as IP7. Therefore the IPTI and IPPCI fields of the Release 2 IntCtl register have that value. The other fields of IntCtl are always zero. The Cause register fields TI and PCI are implemented to provide a direct indication of Timer and Performance Counter interrupts. Since EIC mode is not supported, the Status and Cause registers never use the IPL and RIPL formats for interrupt priority levels.

Because there is only a single Count register shared by all contexts, the DC (Disable Count) bit in the Cause register only has an effect if all contexts set their individual DC bit. Otherwise the Count register continues to run.

The Release 2 EBase register is fully implemented. Within the EBase register, the least significant bits of the CPUNum field reflect the context number within the LX4580 CPU. That is, each context reads a unique CPUNum value from its EBase register. There is one EBase register per context so that each can independently set its exception base value.

The Release 2 EI and DI (Enable and Disable Interrupt) instructions are implemented as required.

Because Shadow Register and Vectored Interrupts are not implemented, the SRSCtl register is always read as zeroes and SRSMap is not implemented. Furthermore, the RDPGPR and WRPGPR instructions simply move the contents of one GPR to another within the executing context's GPR register set.

2.11.2. Hazard Barrier Instructions

The Release 2 instructions EHB, JALR.HB, JR.HB and SYNCI are implemented by the LX4580 CPU to eliminate execution and instruction hazards as described in the Release 2 Architecture.

One implementation dependent aspect of the SYNCI instruction concerns address exceptions. The LX4580 CPU will *not* take an AdEL for a reference by SYNCI to kernel space while in user mode.

2.11.3. Field, Rotate, Shuffle Instructions

The following Release 2 instructions are implemented as required. It is worth noting that a programming note in the Release 2 specification indicates how the WSBH instruction can be used to swap endianness.

• EXT	Extract Bit Field
• INS	Insert Bit Field
• ROTR	Rotate Right
• ROTRV	Rotate Right Variable
• SEB	Sign-Extend Byte to Word
• SEH	Sign-Extend Halfword to Word
• WSBH	Word Swap Bytes Within Halfwords

2.11.4. User Access to Hardware Registers

The Release 2 instruction RDHWR (Read Hardware Register) is implemented as required. The CP0 register HWREna is also implemented as required to conditionally enable a User mode program to read one or more of the defined registers. The values that are supplied when the RDHWR instruction is executed (if the relevant register is enabled for reading) are shown in Table 8.

Table 8: Hardware Register Values

Number	Name	HMT	Implementation Specific Information
0	CPUNum	4	Same as CP0 EBASE.CPUNum
1	SYNCI_Step	1	64
2	CC	1	Same as CP0 Count register
3	CCRes	1	1

2.11.5. CP0 Register Changes

All of the changes and additions to CP0 registers that are associated with the Release 2 architecture are reflected in Table 2, "Standard CP0 Registers". Beyond the changes and additions associated with Release 2 features that are described in other sections of this document, a few more CP0 register changes are included in the LX4580 CPU to be compliant with the Release 2 architecture.

In particular, the Config, Config2, and Config3 have a few more fields defined. Since the Config2 and Config3 fields all refer to features that are not supported in the LX4580 CPU, these registers are not implemented.

The optional WatchHI register has some fields added, but since the LX4580 does not implement the Watch registers, these are not implemented.

The PerfCnt control registers have a W-bit added which only applies to MIPS64 implementations and so is always 0 on the LX4580 CPU.

2.11.6. 64-bit Coprocessor (FPU)

Since the LX4580 does not support any coprocessors, the Release 2 changes to support 64-bit coprocessors, and in particular a 64-bit FPU, are not implemented. The instructions associated with this Release 2 feature will all take Coprocessor Unusable exceptions as required.

2.11.7. 1KB Pages

The Release2 architecture extends the PageMask register by a pair of bits and several other CP0 registers are extended or modified if 1KB pages are to be supported. Also, if 1KB pages are to be supported, a PageGrain register is required.

The LX4580 CPU does not support the 1KB page feature. Therefore, the PageGrain register is not implemented and the changed formats of other registers are not implemented. The extra two bits of PageMask are hard coded to 2#11 as seems to be required.

Chapter 3. Reset (RST)

3.1. Reset Overview

The LX4580 employs a locally sampled reset strategy - synchronous resets registered at the block level.

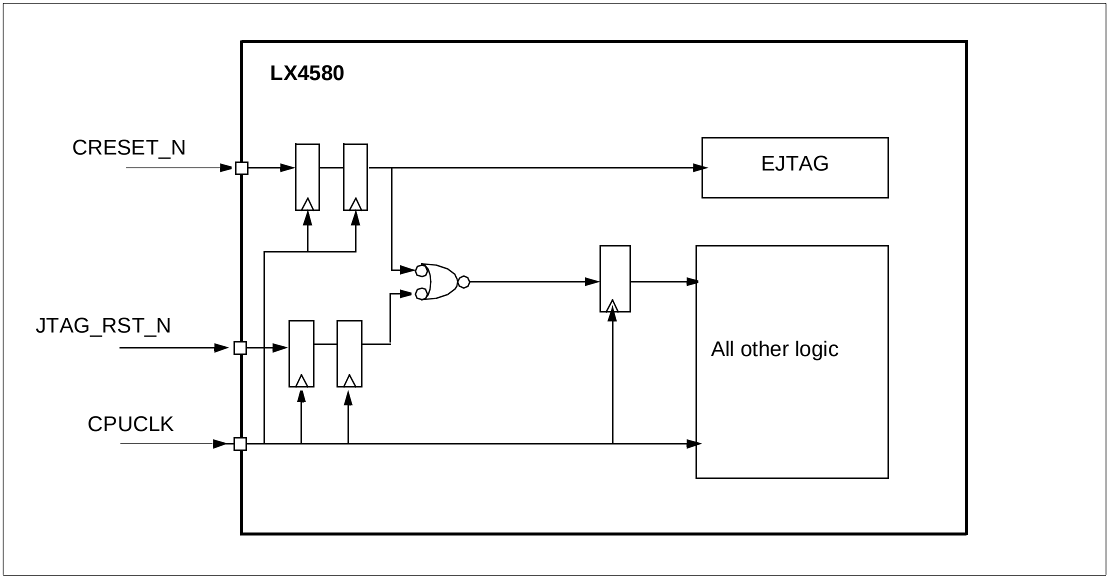

Figure 2: Reset Overview

The reset strategy ensures the following:

- Complete initialization of the LX4580 by the assertion of one external pin.
- Reset debug by EJTAG. The CPU can be placed in a state whereby they will all receive a debug exception when reset and will fetch their reset vector from EJTAG probe space.
- Reset of flip-flops in multiple clock domains. For this reason reset must remain asserted until it is registered in each domain in the design.
- Due to the synchronous nature of resets all clocks must be running when reset is asserted. This includes external interface clocks.
- When reset is asserted all block-level signals must go to an inert state (e.g. for a bus the arbiter must have grant de-asserted during resets). This allows blocks in different clock domains to come out of reset at different times.
- Two external reset pins are provided for power up and debug reset.

3.2. Reset Distribution

The reset system is distributed across the design. Each block will contains a reset flip-flop which samples the chip-level reset in its clock domain. The output of the reset flip-flop is fed to all the flip-flops in that block.

3.3. Reset Operation

A cold-boot sequence would be as follows:

1. CRESET_N is asserted.
2. CRESET_N is de-asserted.
3. Context 0 of the CPU starts executing instructions from the reset vector.
4. The software will initialize the multiple context environment and the peripherals.

A typical multiple CPU EJTAG boot sequence would be as follows:

1. CRESET_N is asserted.
2. CRESET_N is de-asserted.
3. The EJTAG probe is connected to the CPU in turn setting the ProbeEn bit of the CPU’s EJTAG Control Register.
4. The Probe then asserts JTAG_RST_N. This resets everything in the LX4580 apart from the EJTAG ProbeEn flop.
5. JTAG_RST_N is de-asserted the CPU jumps to the debug exception vector at 0xFF200200 from where the system is under EJTAG probe control.

3.4. Reset External LX4580 Interfaces

Table 9: Reset External Interface

Signal Name	Direction	Description
CRESET_N	input	Cold Reset.
JTAG_RST_N	input	Connection from the EJTAG probe.

4.1. LX4580 CPU Overview

This chapter describes the CPU's caches.

The LX4580 CPU includes the following features:

- 500 MHz operation.
- 7-stage pipeline.
- Supports Release 2 MIPS32™ instruction set.
- Four hardware contexts with fine-grained Hardware Multi-Threading (HMT).
- 64, 32, or 16 KByte 4-way set associative instruction cache.
- 64, 32, or 16 KByte 4-way set associative writeback, allocate on write, data cache.
- Performance counters.

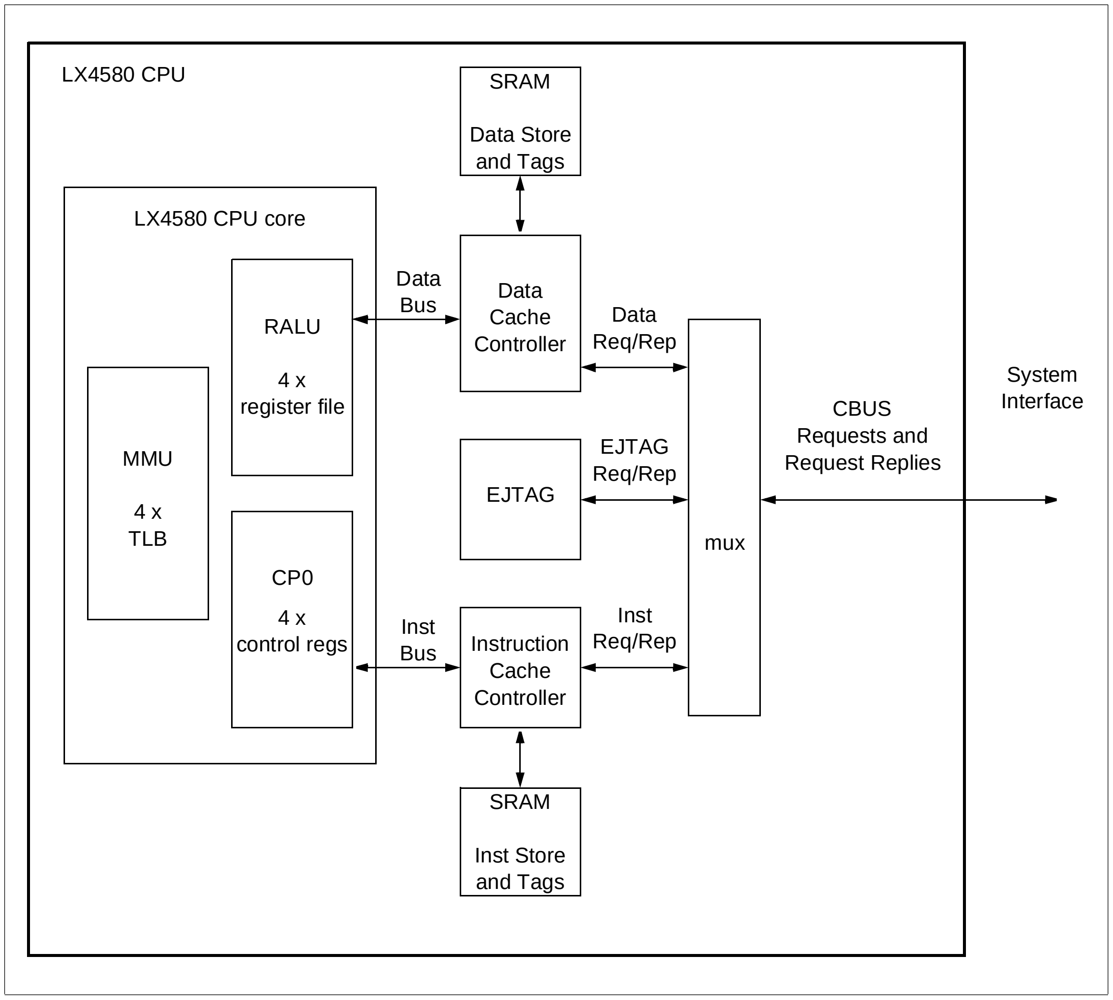

Figure 3: LX4580 CPU and System Interface

4.2. LX4580 CPU Core

The LX4580 CPU core implements the full Release 2 MIPS32™ instruction set as described in Chapter 2. The major blocks of the CPU core are the Register file and ALU (RALU), Control Processor (CP0) and

Memory Management Unit (MMU). Architecturally visible registers in these blocks are replicated to provide a separate copy for each of the CPU contexts.

4.3. Instruction Cache

The LX4580 CPU includes a 4-way set associative instruction cache that operates at the processor clock speed. The instruction cache is organized in 64-byte lines, with Valid and Invalid states.

4.4. Data Cache

The LX4580 CPU includes a 4-way set associative data cache that operates at the processor clock speed. Data in the cache is organized in 64-byte lines.

4.5. Cache Line Replacement Algorithm

When a new line must be brought into the instruction cache or data cache, it may be necessary to evict a line that is currently held. The caches use a 2 bit Most Recently Filled (MRF) field to implement the replacement algorithm. This value is stored as an extra two bits in tag 0 RAM and is updated any time fill data is returned to the cache. On a fill, the stored MRF value indicates which way is currently being filled, so at any point in time this value represents the most recently filled line.

When the data cache needs to allocate a location for a new line, it first examines the valid bits of all 4 ways. If any of the 4 ways are invalid, the smallest number way (0->3) that is invalid is selected. If all 4 ways are valid, the way equal to ((MRF + 1) mod 4) is selected.

The instruction cache does not examine the valid bits in its replacement algorithm. It simply selects the way equal to ((MRF + 1) mod 4) as shown in the last 4 rows of Table 10.

When the data cache has misses for more than one context to the same cache index, it tracks the replacement Way for each of them and must update the MRF bit in the order that the misses were allocated. When more than one context misses to the same cache index, the instruction cache simply suspends the second context without making a request. After the first miss is resolved, the suspended context resumes execution and (assuming it misses again) then makes its request.

Table 10: Cache Line Replacement Algorithm

Tag state	MRF	Way selected
Way 0 invalid	xx	Way 0[a]
Way 0 valid, Way 1 invalid	xx	Way 1[a]
Way 0,1 valid, Way 2 invalid	xx	Way 2[a]
Way 0,1,2 valid Way 3 invalid	xx	Way 3[a]
Way 0-3 valid	00	Way 1
Way 0-3 valid	01	Way 2
Way 0-3 valid	10	Way 3
Way 0-3 valid	11	Way 0

a. This row does not apply to the instruction cache, which ignores the valid bits in its replacement algorithm.

Within the data cache, lines may be locked using the Load Linked (LL) instruction. When one thread executes an LL, that line is locked until a Store Conditional (SC) instruction is executed or some other operation breaks the lock. (See Section 2.2.1) If a line is locked, it cannot be replaced. If the algorithm above selects a line that is locked, the algorithm will increment the way by 1 (way + 1 mod 4) and choose that way. If that way is also locked the algorithm increments again until it finds a way that does not have a locked line.

4.6. CPU Error Handling

4.6.1. Bus Error Handling (IBE and DBE)

When a Bus Error is detected, it is reported to the CPU and context associated with the request. Within the CPU the request which caused the Bus Error must have been one of the following:

- Instruction Fetch (cached or uncached)
- Data Access with Response (typically a read or cache fill)
- Data Access without Response (typically a write or eviction)

For an instruction fetch, the Bus Error replaces the response that would have contained the expected data. The context is marked as IBE Pending. The Icache state machine for the context is released (with no change in the cache if the request was a cached instruction fetch). The context is released from its suspended state so that it can take the IBE exception, as described below.

For a data access with response, the Bus Error replaces the response that would have contained the expected data. The context is marked as DBE Pending. The Dcache state machine for the context is released. If a replacement eviction was required, the cache line will be left in the Invalid state. If no eviction was required, the data cache is unchanged. If a GPR was to be loaded with the data, the value of the GPR is unpredictable (writing zeroes or leaving the GPR unchanged would not be any more useful to software). The context is released from its suspended state so that it can take the DBE exception, as described below.

For a data access without response, the Bus Error is an essentially asynchronous event. The context is marked DBE Pending. If the context is suspended (for cache misses or any other reason) at the time it is marked DBE pending, it remains suspended until the associated conditions are resolved. At that time it takes the DBE exception, as described below.

When a context that is IBE or DBE pending resumes execution and is selected for issue to the pipeline, its instruction fetch is inhibited. Instead, a benign instruction is inserted into the pipeline. When that instruction reaches the end of the pipeline, the context takes an IBE or DBE exception, as appropriate (with IBE having higher priority if both are pending). The pending state is cleared and the context begins execution of the exception handler as per the MIPS32 standard.

Note that it is possible for a context to become IBE or DBE pending while it is executing an exception handler. Nothing in the MIPS32 architecture prevents such a scenario. In this case, a second level of exception handler is entered for the IBE or DBE.

4.6.2. Interrupt Error Response (NMI)

An external NMI signal is an input to the LX4580 CPU. When the signal makes a 0 to 1 transition, all contexts are set to the NMI Pending state (not architecturally visible to software). A context in that state will take an NMI exception the next time that it issues an instruction flow that is not in Debug Mode. In particular, if a context is suspended (for a cache miss, for example) or disabled (via the CXCTRL register) it will not take the NMI exception until it is no longer suspended or disabled.

When a context that is NMI Pending is selected for issue to the pipeline, its instruction fetch is inhibited. Instead, a benign instruction is inserted into the pipeline. When that instruction reaches the end of the pipeline, the context takes an NMI exception, the NMI pending state is cleared (for that context) and the context begins execution of the NMI exception handler as per the MIPS32 standard.

Note that the MIPS32 standard for NMI exceptions requires that the ERL and NMI bits be set in the CP0 Cause register. Also note that the entry point for the NMI exception handler is located in uncached space (it is the same entry point as the Reset handler). The ERL bit forces accesses to kuseg to be unmapped and uncached as if they were accesses to kseg1, which allows saving registers using R0, and avoidance of data cache errors if present.

Since all contexts become NMI pending simultaneously, it is the responsibility of the NMI exception handler to determine the first context to take the exception and, if desired, to disable the other contexts when they take the NMI exception. If it is desired to perform EJTAG debug of the NMI exception handler, an Instruction Break can be placed at the point where the NMI handler branches out of the Reset handler.

Chapter 5. CBUS_Z Interface (ZBI)

5.1. CBUS_Z Interface Overview

The chapter describes the LX4580's CBUS_Z interface option, which is one of two interfaces options supported by the LX4580. The CBUS_Z Interface (ZBI) translates the CPU's internal busses into a unified interface that may connect directly to the user system. The ZBI can also connect to an application specific translation layer to support other bus protocols. The LX4580 supports the MIPS EC^{tm} Interface as an alternative to the CBUS_Z interface. See Chapter 6.

The LX4580 CBUS_Z interface provides a 64-bit data path and operates at 1x the CPU core clock rate.

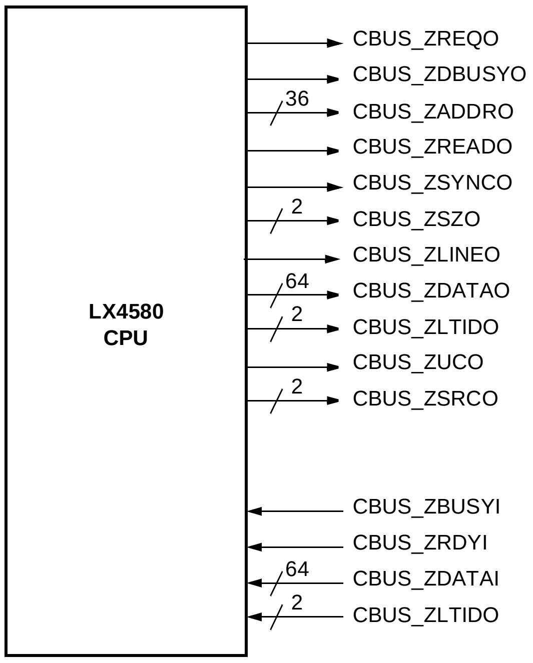

Figure 4: CBUS_Z Interface (ZBI)

5.2. CBUS_Z Interface Signal List

Table 11 summarizes the LX4580's CBUS_Z signals.

Table 11: CBUS_Z Signal List

Name	Direction	Function
CBUS_ZREQO	output	0 - no request, 1 - processor is initiating a request
CBUS_ZBUSYI	input	1 - External logic cannot accept request. The current CBUS_Z request, if any, is ignored by external logic. 0 - External logic is ready to accept a request.
CBUS_ZADDRO[35:0]	output	Address
CBUS_ZREADO	output	1=Read, 0=Write
CBUS_ZSYNCO	output	1=Sync request, 0=Normal Request (CBUS_ZREAD will indicate write on sync cycles)

Name	Direction	Function
CBUS_ZSZO[1:0]	output	Transfer size 2'b00 - 1 byte 2'b01 - 2 bytes 2'b10 - 3 bytes 2'b11 - 1 word This signal is don't care when CBUS_ZLINEO is asserted.
CBUS_ZLINEO	output	1 - line access, 0 - single access
CBUS_ZDATAO[63:0]	output	Write Data
CBUS_ZLTIDO[1:0]	output	Local thread ID
CBUS_ZUCO	output	1 - uncached access, 0 - cached access
CBUS_ZSRCO[1:0]	output	transaction source (within LX4580): 2'b00 Instruction Cache 2'b01 Data Cache 2'b10 EJTAG 2'b11 reserved
CBUS_ZRDYI	input	Read data is available
CBUS_ZDBUSYO	output	1 - LX4580 is not ready to receive Data. External logic must hold CBUS_ZDATAI, CBUS_ZLTIDI, and CBUS_ZVALTYPEI until CBUS_ZDBUSYO is deasserted. 0 - LX4580 is ready to receive Data.
CBUS_ZDATAI[63:0]	input	Read Data
CBUS_ZLTIDI[1:0]	input	Thread associated with Read Data
CBUS_ZVALTYPEI[1:0]	input	Indicates read data type: 2'b00 Instruction Cache 2'b01 Data Cache 2'b10 EJTAG 2'b11 reserved

5.3. CBUS_Z Endian Mode

The LX4580 is bi-endian. A static input pin, CFG_BIGENDIAN, determines the configured endian mode of the processor and the CBUS_Z interface. The memory contents shown below apply to the examples presented in this section.

byte address	hex contents
0	88
1	99
2	AA
3	BB
4	CC
5	DD
6	EE
7	FF

The program visible behavior of all cacheable and uncacheable loads and stores conform to the formats presented in Table 12. The CBUS_Z command is not shown in the table and can be inferred from the size of the valid data in the CBUS_Z data patterns.

Table 12: Effect of Endian Mode on CBUS_Z

Inst and addr[2:0]		Big Endian		Little Endian	
		CBUS_Z Addr/Data[a]	Reg[b]	CBUS_Z Addr/Data[a]	Reg[b]
SB/LB	0	0 / 88------ --------	00000088	0 / -------- ------88	00000088
	1	1 / --99---- --------	00000099	1 / -------- ----99--	00000099
	2	2 / ----AA-- --------	000000AA	2 / -------- --AA----	000000AA
	3	3 / ------BB --------	000000BB	3 / -------- BB------	000000BB
	4	4 / -------- CC------	000000CC	4 / ------CC --------	000000CC
	5	5 / -------- --DD----	000000DD	5 / ----DD-- --------	000000DD
	6	6 / -------- ----EE--	000000EE	6 / --EE---- --------	000000EE
	7	7 / -------- ------FF	000000FF	7 / FF------ --------	000000FF
SH/LH	0	0 / 8899---- --------	00008899	0 / -------- ----9988	00009988
	2	2 / ----AABB --------	0000AABB	2 / -------- BBAA----	0000BBAA
	4	4 / -------- CCDD----	0000CCDD	4 / ----DDCC --------	0000DCC
	6	6 / -------- ----EEFF	0000EEFF	6 / FFEE---- --------	0000FFEE
SW/LW	0	0 / 8899AABB --------	8899AABB	0 / -------- BBAA9988	BBAA9988
	4	4 / -------- CCDDEEFF	CCDDEEFF	4 / FFEEDDCC --------	FFEEDDCC
double word[c]		0 / 8899AABB CCDDEEFF	n/a	0 / FFEEDDCC BBAA9988	n/a
LWL/SWL	0	0 / 8899AABB --------	8899AABB	0 / -------- ------88	88------
	1	1 / --99AABB --------	99AABB--	0 / -------- ----9988	9988----
	2	2 / ----AABB --------	AABB----	0 / -------- --AA9988	AA9988--
	3	3 / ------BB --------	BB------	0 / -------- BBAA9988	BBAA9988
	4	4 / -------- CCDDEEFF	CCDDEEFF	4 / ------CC --------	CC------
	5	5 / -------- --DDEEFF	DDEEFF--	4 / ----DDCC --------	DDCC----
	6	6 / -------- ----EEFF	EEFF----	4 / --EEDDCC --------	EEDDCC--
	7	7 / -------- ------FF	FF------	4 / FFEEDDCC --------	FFEEDDCC
LWR/SWR	0	0 / 88------ --------	------88	0 / -------- BBAA9988	BBAA9988
	1	0 / 8899---- --------	----8899	1 / -------- BBAA99--	--BBAA99
	2	0 / 8899AA-- --------	--8899AA	2 / -------- BBAA----	----BBAA
	3	0 / 8899AABB --------	8899AABB	3 / -------- BB------	------BB
	4	4 / -------- CC------	------CC	4 / FFEEDDCC --------	FFEEDDCC
	5	4 / -------- CCDD----	----CCDD	5 / FFEEDD-- --------	--FFEEDD
	6	4 / -------- CCDDEE--	--CCDDEE	6 / FFEE---- --------	----FFEE
	7	4 / -------- CCDDEEFF	CCDDEEFF	7 / FF------ --------	------FF

a. A dash in this column indicates the data is undefined.
b. A dash in this column indicates the register contents are not affected (loads) or are ignored (stores).
c. Double word transfers only occur as part of a line read or write.

For the purpose of the unaligned load and store instructions (LWL and LWR) the address bits shown in CBUS_Z columns of Table 12 are adjusted values, not the raw address computed by the instruction. For LWL in little endian mode and LWR in big endian mode, the two low order address bits are forced to zero. For all other cases the address bits are unchanged.

5.4. CBUS_Z Line Read Interleave Order

The line read operation reads a sequence of data beats from memory corresponding to the size of a cache line. The cache line size affects how many cycles are required to transfer the entire line. A 32 byte line size is assumed here.

The CBUS_Z target transfers read data starting with *word zero first*. With word zero first operation, the target transfers eight 64-bit beats of data in sequence, starting at the nearest 32-byte-aligned address smaller or equal to the address that the initiator drives. In other words, the target starts the transfer at the beginning of the line containing the requested address.

5.5. CBUS_Z Read Completion

External logic may manage CBUS_Z read completion in one of the following ways:

1. Pended. One read request is outstanding at a time. The CBUS_Z is unavailable for other requests between the time the read request is issued and read data is returned. External logic keeps CBUS_ZBUSYI asserted until the read response.

2. Pipelined. Multiple read requests are outstanding a time. Read responses occur in the same order that the read request is made.

3. Split. Multiple read requests are outstanding at a time. Read responses can be returned in a different order than the request.

When the read management technique is Pipelined or Split, write requests can be issued while reads are pending. For read response data, 2.5 lines of data buffering is provided. Read requests can be issued until the amount of pending data is equal to the available data buffering. Pending reads can be any combination of the following:

- One ICACHE read per thread.
- One DCACHE read per thread.
- ONE EJTAG read.

5.6. CBUS_Z Transaction Types

The following transaction types are supported by the CBUS_Z interface:

- Sub-line read.
- Line read.
- Sub-line write.
- Line write.
- Sync. This transaction occurs when CBUS_ZSYNCO is asserted. Other signals behave as in a sub-line write transaction. Address and data should be ignored.

5.7. CBUS_Z Protocol

The transaction request protocol is controlled with CBUS_ZREQO output and CBUS_ZBUSYI input.

1. The CBUS_ZREQO output is asserted by the ZBI to initiate an access to external logic. Additional CBUS_Z* outputs are driven by the ZBI to provide the transaction details.

2. CBUS_ZREQO remains asserted until the CBUS_ZBUSYI is not asserted by external logic.

For a write transaction, the transaction is completed after step 2. For a read transaction, additional steps control the return of read data by the external logic, using the CBUS_ZRDYI input and the CBUS_ZDBUSYO output.

1. If CBUS_ZVALTYPEI indicates Instruction Cache or EJTAG data is present on CBUS_ZDATAI, the data is always accepted by the LX4580.

2. If CBUS_ZVALTYPEI indicates that Data Cache data is present on CBUS_ZDATAI and CBUS_ZDBUSYO is asserted, the external logic must continue to drive CBUS_ZVALTYPEI and CBUS_ZDATAI until CBUS_ZDBUSYO deasserts.

5.8. CBUS_Z Transaction Timing Diagrams

Note: All of the following timing diagrams assume a line size of 32 bytes. For reads, the transaction request is shown in a different timing diagram than the returning read data as there is no protocol link between the two.

5.8.1. Back-to-Back Sub-Line Writes with Busy

In cycle 1 the write to address A is accepted by the external logic. In cycle 2 the external logic asserts CBUS_ZBUSYI which causes the LX4580 to hold its request. In cycle 3, the external logic de-asserts CBUS_ZBUSYI and accepts the request.

In this example, cycle 4 could be used by the processor to initiate another request.

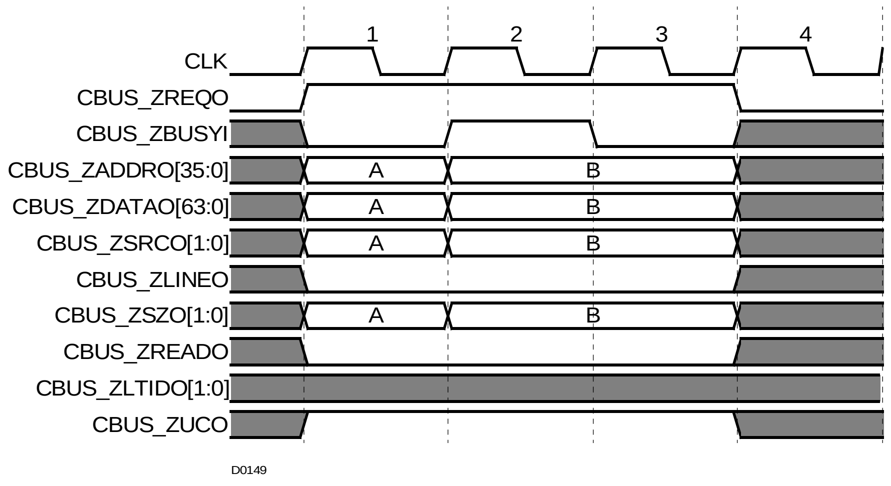

Figure 5: CBUS_Z Back-to-Back Sub-Line Writes with Busy

5.8.2. Line Writes

During a line write the address is given in cycle 1. External logic signals that it is able to accept a line write request by de-asserting CBUS_ZBUSYI. External logic does not honor a line write request when CBUS_ZBUSYI is asserted.

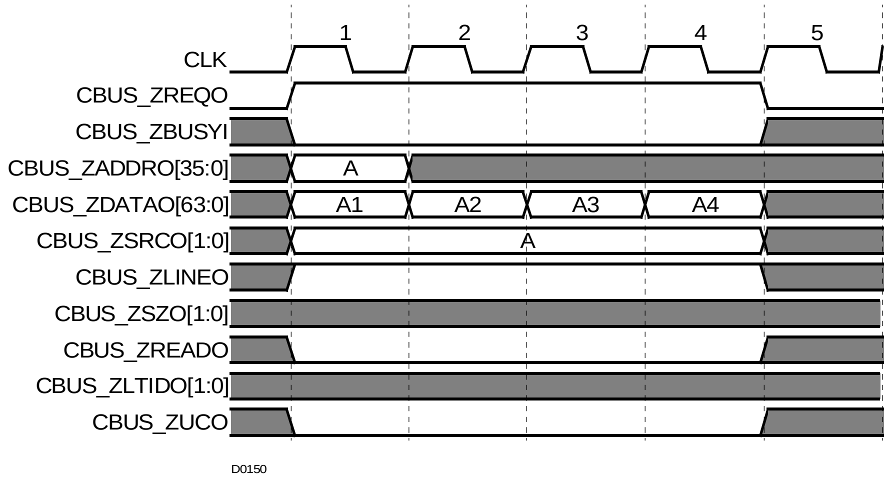

Figure 6: CBUS_Z Line Write

5.8.3. Read Request

A line read request takes only one cycle with the data being returned later by the external logic.

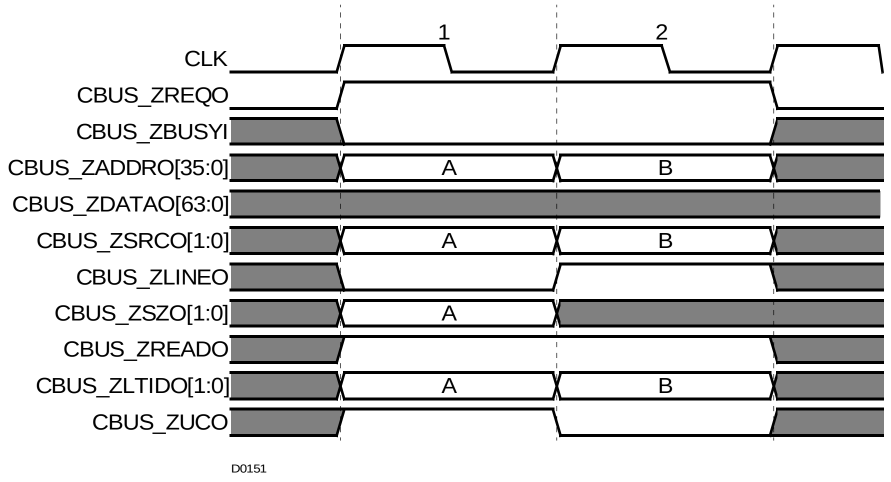

Figure 7: CBUS_Z Read Requests

5.8.4. Returning Read Data

External logic supplies read data on the CBUS_ZDATAI and CBUS_ZLTIDI inputs while asserting a bit within CBUS_ZVALTYPEI. If CBUS_ZVALTYPEI indicates Data (2’b01), the LX4580 only accepts the read data if it has de-asserted CBUS_ZDBUSYO. If CBUS_ZDBUSYO is asserted with CBUS_ZVALTYPEI = 2’b01, the external logic must maintain CBUS_ZVALTYPEI and CBUS_ZDATAI until CBUS_ZDBUSYO is deasserted

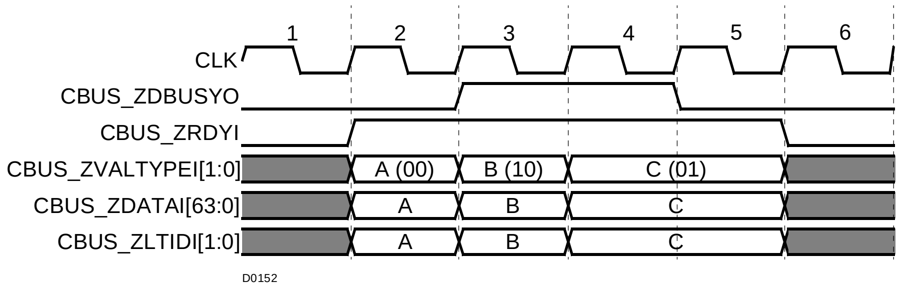

Figure 8: CBUS_Z Sub-Line Read Data and DBUSY

A read line data return is illustrated below. The external device asserts the appropriate bit of CBUS_ZVALTYPEI for each data beat. Assertion of CBUS_ZDBUSYO is also illustrated.

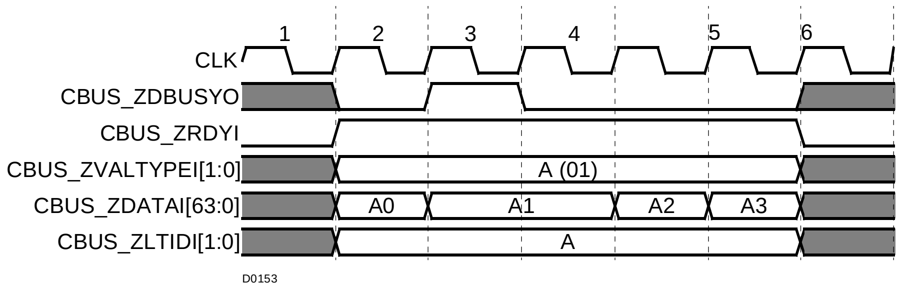

Figure 9: Read Data for a Line Read Request

6.1. Overview

The *EC Interface (ECI)* is used in the LX4580 as the interface between the CPU and memory and IO devices.

The chapter describes the LX4580's *EC Interface (ECI)* option, which is one of two interfaces options supported by the LX4580. The ECI translates the CPU's internal busses into a unified 64-bit interface that may connect directly to the user's system. The EC protocol is described in the MIPS "EC^{tm} Interface Specification", Revision 1.05. The LX4580 supports Lexra's CBUS_Z Interface (ZBI) as an alternative to the EC interface. See Chapter 5.

The LX4580 EC interface provides a 64-bit data path and operates at 1x or 1/2x the CPU core clock rate. If the LX4580 is configured for 64 or 128-byte cache lines, multiple 32-byte EC interface bursts are required to transfer one cache line.

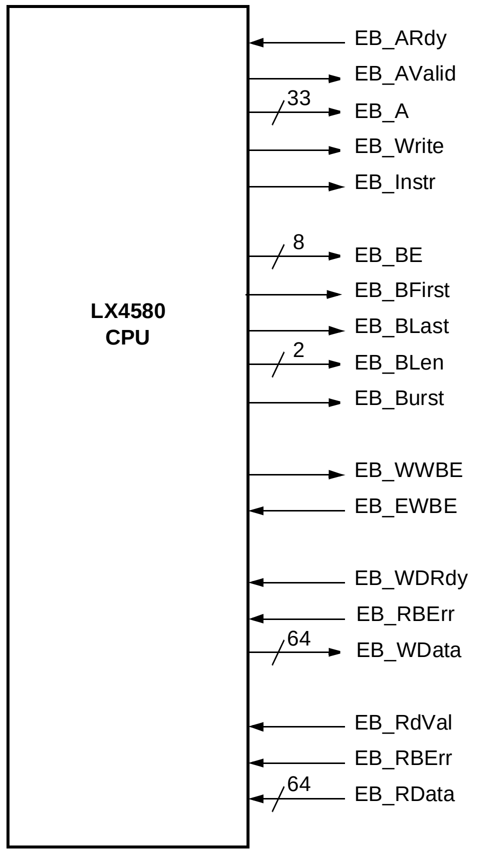

Figure 10: EC Interface (ECI)

6.2. EC Interface Signals

Table 13: EC Interface Signals

Signal Name	Direction	Description
Command		
EB_ARdy	input	Slave can accept new request (address phase)
EB_AValid	output	Request valid
EB_A[35:3]	output	Address bus
EB_Write	output	Current address phase is for a write request (else read)
EB_Instr	output	Current address phase is for an instruction fetch (else data or EJTAG)
Burst Control		
EB_BE[7:0]	output	Byte enable
EB_BFirst	output	First address phase of burst
EB_BLast	output	Last address phase of burst
EB_BLen[1:0]	output	Burst length (beats)
EB_Burst	output	Current address phase is part of burst
Write Buffer Synchronization		
EB_WWBE	output	Master is waiting for external write buffer to empty (SYNC instruction executed)
EB_EWBE	input	External write buffer is empty
Write Data		
EB_WDRdy	input	Slave is ready to accept write data
EB_WBErr	input	Bus error on write
EB_WData[63:0]	output	Write data bus
Read Data		
EB_RdVal	input	Read data is valid
EB_RBErr	input	Bus error on read
EB_RData[63:0]	input	Read data bus

6.3. EC Interface Endian Mode

The LX4580 is bi-endian. A static input pin, CFG_BIGENDIAN, determines the endian mode of the processor and the EC interface. The memory contents below apply to the examples presented in this section.

byte address	hex contents
0	88
1	99
2	AA
3	BB
4	CC
5	DD
6	EE
7	FF

The program visible behavior of all cacheable and uncacheable loads conform to the formats presented in Table 14. The EC byte enable flags (EC_BE[7:0]) can be inferred from EC data patterns.

Table 14: Effect of Endian Mode on EC Interface

Inst and addr[2:0]		Big Endian		Little Endian	
		EC Data[a]	Reg[b]	EC Data[a]	Reg[b]
SB/LB	0	88------ --------	00000088	-------- ------88	00000088
	1	--99---- --------	00000099	-------- ----99--	00000099
	2	----AA-- --------	000000AA	-------- --AA----	000000AA
	3	------BB --------	000000BB	-------- BB------	000000BB
	4	-------- CC------	000000CC	------CC --------	000000CC
	5	-------- --DD----	000000DD	----DD-- --------	000000DD
	6	-------- ----EE--	000000EE	--EE---- --------	000000EE
	7	-------- ------FF	000000FF	FF------ --------	000000FF
SH/LH	0	8899---- --------	00008899	-------- ----9988	00009988
	2	----AABB --------	0000AABB	-------- BBAA----	0000BBAA
	4	-------- CCDD----	0000CCDD	----DDCC --------	0000DDCC
	6	-------- ----EEFF	0000EEFF	FFEE---- --------	0000FFEE
SW/LW	0	8899AABB --------	8899AABB	-------- BBAA9988	BBAA9988
	4	-------- CCDDEEFF	CCDDEEFF	FFEEDDCC --------	FFEEDDCC
double word[c]		8899AABB CCDDEEFF	n/a	FFEEDDCC BBAA9988	n/a
LWL/SWL	0	8899AABB --------	8899AABB	-------- ------88	88------
	1	--99AABB --------	99AABB--	-------- ----9988	9988----
	2	----AABB --------	AABB----	-------- --AA9988	AA9988--
	3	------BB --------	BB------	-------- BBAA9988	BBAA9988
	4	-------- CCDDEEFF	CCDDEEFF	------CC --------	CC------
	5	-------- --DDEEFF	DDEEFF--	----DDCC --------	DDCC----
	6	-------- ----EEFF	EEFF----	--EEDDCC --------	EEDDCC--
	7	-------- ------FF	FF------	FFEEDDCC --------	FFEEDDCC
LWR/SWR	0	88------ --------	------88	-------- BBAA9988	BBAA9988
	1	8899---- --------	----8899	-------- BBAA99--	--BBAA99
	2	8899AA-- --------	--8899AA	-------- BBAA----	----BBAA
	3	8899AABB --------	8899AABB	-------- BB------	------BB
	4	-------- CC------	------CC	FFEEDDCC --------	FFEEDDCC
	5	-------- CCDD----	----CCDD	FFEEDD-- --------	--FFEEDD
	6	-------- CCDDEE--	--CCDDEE	FFEE---- --------	----FFEE
	7	-------- CCDDEEFF	CCDDEEFF	FF------ --------	------FF

a. A dash in this column indicates the data is undefined.

b. A dash in this column indicates the register contents are not affected (loads) or are ignored (stores).

c. Double word transfers only occur as part of a line read or write.

Note that the EC interface signals do not intrinsically employ the concept of endian mode. The LX4580 uses the internally generated address bits 2:0 to make the determination of what EC byte enable signals to assert. The three low-order address bits are omitted from the EC interface address lines. There is no way to determine the endian mode employed for a transfer over the EC interface.

For the purpose of the unaligned load and store instructions (LWL and LWR) the address bits shown in CBUS columns of Table 14 are adjusted values, not the raw address computed by the instruction. For LWL in little endian mode and LWR in bit endian mode, the two low order address bits are forced to zero. For all other cases the address bits are unchanged.

6.4. EC Interface Pending Requests

The EC Interface Specification allows for an unlimited number of outstanding requests. In the ECI, the number of outstanding requests are limited by the amount of internal buffering available. The following rules apply:

- For reads, up to 2.5 lines of data buffering is provided. Read requests can be issued until the amount of pending data is equal to the available data buffering.
- For sub-line writes, one word of buffering is provided. Write requests can be issued until the amount of pending data is equal to the available data buffering.
- For line writes, no buffering is provided. No other requests of any type can be issued until the last beat of write data is accepted by the slave.
- The 4580 CPU has an architectural limitation on the number of reads that may be pending, independent of ECI buffering. Those limits are:
    - One ICACHE read per thread
    - One DCACHE read per thread
    - ONE EJTAG read
- When the 4580 executes a SYNC instruction, no other requests of any type can be issued until all pending writes complete, and the slave asserts EB_WEBE.

THE REMAINDER OF THIS CHAPTER IS FOR INTERNAL LEXRA USE.

6.5. EC Interface Gasket Overview

The *EC Interface Gasket (ECI)* is used in the LX4580 as the interface between the CPU and memory and IO devices. The CPU connection is provided by the LX4580's CBUS and IBUS interfaces. These interfaces are described in the remaining sections of this chapter. The ECI interface accepts read data destined for the CBUS reply interface, and provides write data from the CBUS request interface or IBUS reply interface.

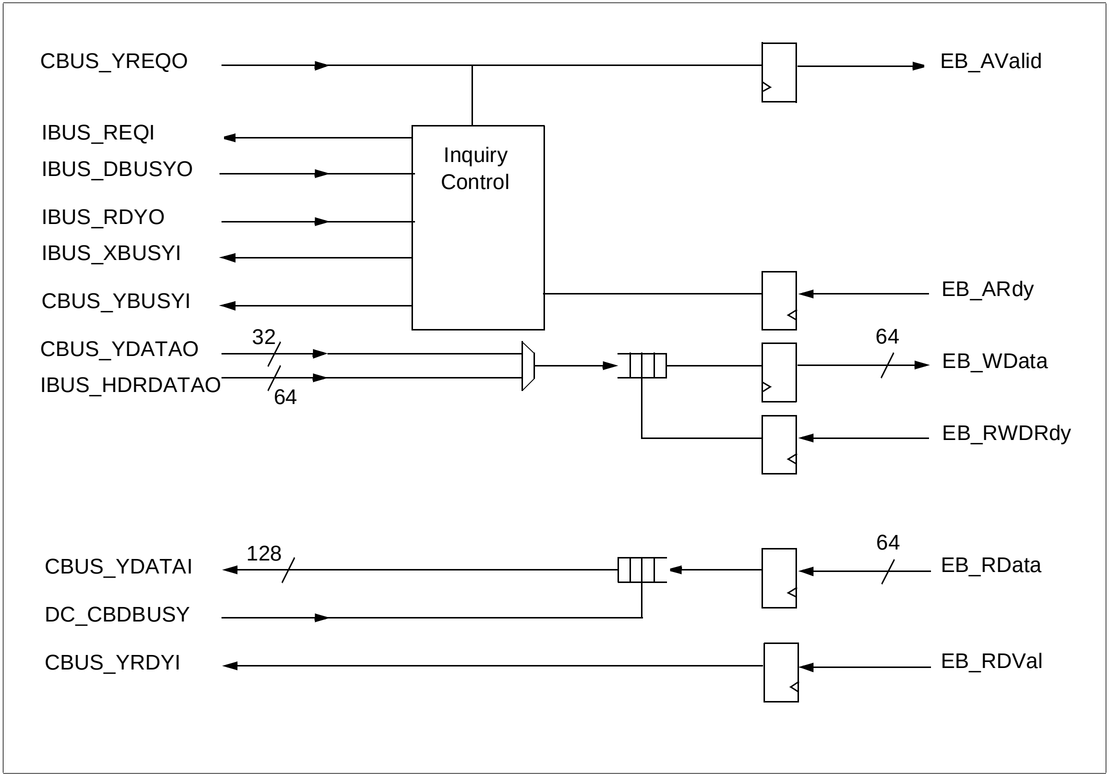

Figure 11: EC Interface Gasket (ECI)

6.6. Supported Configurations

Table 15: Supported Configurations

EC Bus Width (bits)	Cache Line Size (bytes)	Beats per Burst	ECI/Core Speed Ratio	Theoretical BW (Gb/s, assumes 500 Mhz core)	Compatibility
64	64	4[1]	1	32	5KC, 5KF, EC-64
64	64	4[1]	1/2	16	5KC, 5KF, EC-64

[1] Two burst sequences are required to transfer a cache line on the EC interface.

6.7. Implementation Guidance for Endian Mode

From a logic design and verification perspective, there are trade-offs over how far the endian mode should reach into the processor. While for the purposes of verification and the simulation environment it may be preferable for the entire processor to reflect the configured endian mode, this is by no means necessary in terms of architecture.

Table 20 in Section 6.3 provides the complete architectural definition of endian modes in the processor. The material in the section provides background and implementation information.

6.7.1. Consistency of Endian Mode in the System

The motivation of providing a bi-endian CPU is to permit a consistent system view of the endian mode. That is, software, the CPU and system bus devices all should implement the same endian mode. If this is not the case, the issues that arise are more a matter of software and system level design that of CPU architecture.

It makes little or no sense for software to operate with an endian mode that differs from the CPU. Caches operate in a specific way in response to the endian mode. It is more practical and straightforward for applications and the operating system to be consistent with the CPU's endian mode. (Or, vice-versa.)

In some cases, software might perform manual conversions of data structures between endian modes when accessing specific devices or data structures that have a different endian mode than the CPU. The best way to treat this is explicitly as needed.

Lastly, while it is possible to build systems in which the endian mode of a CPU and system devices differ, it is presumed that the bi-endian LX4580 will be configured to match the mode of the system devices.

For the above reasons, there is no mixed-endian support required in the LX4580.

6.7.2. Address Invariance

For any address visible to software or external hardware, the CPU's implementation of the endian modes *must* obey the principle of address invariance. This means that for a reference of a given size and address the same address is employed in software and all user visible hardware, *regardless of endian mode*.

If the CPU responds to the endian mode throughout its entire implementation, address invariance will be satisfied without any additional effort.

In contrast, if the CPU maintains big endian mode internally and an endian switch is implemented at some internal interface (for example within the ECI), then the conversion must take care to ensure address invariance.

6.7.3. Data Invariance

Data invariance is analogous to address invariance. That is, for a reference of a given size, address, *and endian mode*, data is driven on different busses using a consistent alignment.

In a pure architectural sense, the only data alignment that matters is the data which is driven through the system bus interface. This specification is *neutral* on the extent to which data invariance should be supported on the processor's internal data paths. As suggested in the previous section, it is reasonable to break data invariance on internal data paths as a means of preserving address invariance in an endian switch.

Supporting data invariance throughout the entire CPU design is a double edged sword. On the one hand, there is some benefit to a consistent treatment of endian mode in any interface that could potentially be exposed to a customer (e.g. CBUS, RAMs for DMA). On the other hand, a substantial set of tools have been developed to support Lexra's simulation environment which access internal state, and these assume big endian formats. While in the long run, these tools might benefit from a consistent treatment of endian mode for the entire environment, they would first have to be changed to support bi-endian operation.

In terms of CPU implementation, the easiest way to provide universal data invariance is to implement the endian switch throughout the entire CPU (data cache, CBUS and outward). This actually entails very little hardware, being limited to logic that decodes low order address bits to control data selection muxes. The muxes themselves already exist to reconcile different access sizes in the current big endian implementation. Changing the endian mode does not add any data inputs to these muxes.

6.7.4. Reverse Endian Support Not Recommend

The MIPS32 architecture permits an optional reverse endian mode, controlled with the CP0 Status register RE bit. When set, this bit reverses the endian mode of user mode references only. Since this only affects user mode, it is not an effective means for managing endian difference between a CPU and system devices.

Allowing endian mode to vary instruction by instruction in response to this flag is burdensome and not worth the effort. Therefore, it is *not recommended* that the LX4580 support reverse endian operation.

6.7.5. Endian Mode and Unaligned Load/Store

As seen in Table 14 in Section 6.3, the processor must zero the two low order address bits of the calculated addresses of unaligned loads and stores, so that the address presented to the system is word-aligned. This truncation depends on the endian mode. Table 14 provides a concise specification of what the individual instructions must do, but it somewhat obscures the real working of the unaligned load and store instructions.

The tables below illustrate unaligned loads and stores from a different perspective. They show an object in memory consisting of bytes represented by the symbols A, B, C, D starting at the (aligned) location 0, and the (unaligned) locations 1, 2 and 3. The left and right flavors of the unaligned load and store instructions are paired in the manner that they would normally be executed. For example, for a big endian LWL with the two calculated low order address bits equal to 0, the corresponding LWR calculates an address with the low order bits equal to 3. From these tables it is clear how these instructions respond the endian mode to access the left and right portions of an unaligned word in memory, and how the portions are merged in the CPU register.

Table 16: Big Endian Unaligned Load/Store Address Adjustments

	LWL/SWL		LWR/SWR		
Memory Contents 0 1 2 3 4 5 6 7	Calculated Addr[3:0]	Register Contents	Calculated Addr[3:0]	Adjusted Addr[3:0]	Register Contents
A B C D - - - -	0	A B C D	3	0	A B C D
- A B C D - - -	1	A B C -	4	4	- - - D
- - - A B C D -	2	A B - -	5	4	- - C D
- - - - A B C D	3	A - - -	6	4	- B C D

Table 17: Little Endian Unaligned Load/Store Address Adjustments

Memory Contents 0 1 2 3 4 5 6 7	LWL/SWL			LWR/SWR	
	Calculated Addr[3:0]	Adjusted Addr[3:0]	Register Contents	Calculated Addr[3:0]	Register Contents
- - - - A B C D	3	0	A B C D	0	A B C D
- - - A B C D -	4	4	A - - -	1	- B C D
- - A B C D - -	5	4	A B - -	2	- - C D
- A B C D - - -	6	4	A B C -	3	- - - D

6.8. CBUS_Y Interface

The CBUS_Y interface is not available for application use. As an alternative to ECI, a simpler version of the CBUS_Y protocol is available with the CBUS_Z interface option. See XREF.

The CBUS_Y provides signalling between the CPU, EJTAG, ICACHE, DCACHE and ECI. Certain requests may require an eviction (RLE, RLME, WLI, and WLE). These evictions are caused by sending an inquiry request on the IBUS interface.

CBUS_Y requests may be single byte, half word, tri-byte, word, or line length. All line requests are zero word first. The CBUS_Y interface runs at the CPU core clock rate. CBUS_Y requests may be throttled if the EC interface is busy. CBUS_Y requests are also throttled while an inquiry is pending on the IBUS interface.

Read reply data from the EC interface is returned over the CBUS_Y reply interface. Up to five CBUS_Y read requests may be pending (one per thread). A transfer on the CBUS_Y reply interface must not be interrupted. Buffering is provided for two lines of reply data to meet this requirement. Transfers on the CBUS_Y reply interface destined for DCACHE may be thottled if the DCACHE is busy.

Table 18: CBUS_Y Request Interface

Signal Name	Direction	Description
CBUS_YREQO	input	CBus Request
CBUS_YADDRO[35:0]	input	Physical Address for Request
CBUS_YDATAO[31:0]	input	Data for Request
CBUS_YCMDO[3:0]	input	CBus Request Command
CBUS_YSZO[1:0]	input	Size of Data for Request
CBUS_YSRCO[1:0]	input	Source of Request
CBUS_YDWAYO[1:0]	input	data cache way L1 duplicate tag update (unused)
CBUS_YLTIDO[1:0]	input	Thread ID of Request
CBUS_YBUSYI	output	EC Interface Busy

Table 19: CBUS Reply Interface

Signal Name	Direction	Description
CBUS_YRDYI	output	Reply Data ready
CBUS_YDESTI[2:0]	output	Destination for Reply Data
CBUS_YLSTEI[2:0]	output	Line State and Transaction Reply Type
CBUS_YRDLTIDI[1:0]	output	Thread ID for Reply
CBUS_YDATAI[127:0]	output	Reply Data
DC_CBDBUSY	input	Data cache Busy

6.8.1. CBUS_Y Endian Mode

Table 20 presents the impact of endian mode on the CBUS_Y. Although the CBUS_Y command is not shown in the table and can be inferred from the size of the valid data in the CBUS_Y data patterns. The effect of endian mode on the EC interface is also shown for convenience.

Table 20: Effect of Endian Mode on CBUS_Y

Inst and addr[2:0]		Big Endian			Little Endian		
		CBUS Addr/Data[a]	EC Data[a]	Reg[b]	CBUS Addr/Data[a]	EC Data[a]	Reg[b]
SB/LB	0	0 / 88------ --------	88------ --------	00000088	0 / -------- ------88	-------- ------88	00000088
	1	1 / --99---- --------	--99---- --------	00000099	1 / -------- ----99--	-------- ----99--	00000099
	2	2 / ----AA-- --------	----AA-- --------	000000AA	2 / -------- --AA----	-------- --AA----	000000AA
	3	3 / ------BB --------	------BB --------	000000BB	3 / -------- BB------	-------- BB------	000000BB
	4	4 / -------- CC------	-------- CC------	000000CC	4 / ------CC --------	------CC --------	000000CC
	5	5 / -------- --DD----	-------- --DD----	000000DD	5 / ----DD-- --------	----DD-- --------	000000DD
	6	6 / -------- ----EE--	-------- ----EE--	000000EE	6 / --EE---- --------	--EE---- --------	000000EE
	7	7 / -------- ------FF	-------- ------FF	000000FF	7 / FF------ --------	FF------ --------	000000FF
SH/LH	0	0 / 8899---- --------	8899---- --------	00008899	0 / -------- ----9988	-------- ----9988	00009988
	2	2 / ----AABB --------	----AABB --------	0000AABB	2 / -------- BBAA----	-------- BBAA----	0000BBAA
	4	4 / -------- CCDD----	-------- CCDD----	0000CCDD	4 / ----DDCC --------	----DDCC --------	0000DDCC
	6	6 / -------- ----EEFF	-------- ----EEFF	0000EEFF	6 / FFEE---- --------	FFEE---- --------	0000FFEE
SW/LW	0	0 / 8899AABB --------	8899AABB --------	8899AABB	0 / -------- BBAA9988	-------- BBAA9988	BBAA9988
	4	4 / -------- CCDDEEFF	-------- CCDDEEFF	CCDDEEFF	4 / FFEEDDCC --------	FFEEDDCC --------	FFEEDDCC
double word[c]		0 / 8899AABB CCDDEEFF	8899AABB CCDDEEFF	n/a	0 / FFEEDDCC BBAA9988	FFEEDDCC BBAA9988	n/a
LWL/SWL	0	0 / 8899AABB --------	8899AABB --------	8899AABB	0 / -------- ------88	-------- ------88	88------
	1	1 / --99AABB --------	--99AABB --------	99AABB--	0 / -------- ----9988	-------- ----9988	9988----
	2	2 / ----AABB --------	----AABB --------	AABB----	0 / -------- --AA9988	-------- --AA9988	AA9988--
	3	3 / ------BB --------	------BB --------	BB------	0 / -------- BBAA9988	-------- BBAA9988	BBAA9988
	4	4 / -------- CCDDEEFF	-------- CCDDEEFF	CCDDEEFF	4 / ------CC --------	------CC --------	CC------
	5	5 / -------- --DDEEFF	-------- --DDEEFF	DDEEFF--	4 / ----DDCC --------	----DDCC --------	DDCC----
	6	6 / -------- ----EEFF	-------- ----EEFF	EEFF----	4 / --EEDDCC --------	--EEDDCC --------	EEDDCC--
	7	7 / -------- ------FF	-------- ------FF	FF------	4 / FFEEDDCC --------	FFEEDDCC --------	FFEEDDCC

Table 20: Effect of Endian Mode on CBUS_Y (Continued)

Inst and addr[2:0]	Big Endian			Little Endian		
	CBUS Addr/Data[a]	EC Data[a]	Reg[b]	CBUS Addr/Data[a]	EC Data[a]	Reg[b]
LWR/SWR 0	0 / 88------ --------	88------ --------	------88	0 / -------- BBAA9988	-------- BBAA9988	BBAA9988
1	0 / 8899---- --------	8899---- --------	----8899	1 / -------- BBAA99--	-------- BBAA99--	--BBAA99
2	0 / 8899AA-- --------	8899AA-- --------	--8899AA	2 / -------- BBAA----	-------- BBAA----	----BBAA
3	0 / 8899AABB --------	8899AABB --------	8899AABB	3 / -------- BB------	-------- BB------	------BB
4	4 / -------- CC------	-------- CC------	------CC	4 / FFEEDDCC --------	FFEEDDCC --------	FFEEDDCC
5	4 / -------- CCDD----	-------- CCDD----	----CCDD	5 / FFEEDD-- --------	FFEEDD-- --------	--FFEEDD
6	4 / -------- CCDDEE--	-------- CCDDEE--	--CCDDEE	6 / FFEE---- --------	FFEE---- --------	----FFEE
7	4 / -------- CCDDEEFF	-------- CCDDEEFF	CCDDEEFF	7 / FF------ --------	FF------ --------	------FF

a. A dash in this column indicates the data is undefined.

b. A dash in this column indicates the register contents are not affected (loads) or are ignored (stores).

c. Double word transfers only occur as part of a line read or write.

6.8.2. CBUS_Y Command Encoding

Table 21: CBUS_Y Commands

Crossbar Request Message	CBUS_YCMDO[3:0]	CBUS_YSZO[1:0]
RL	1001	N/A
RLM	1101	N/A
RLE	1011	N/A
RLME	1111	N/A
RB	1000	00
RH	1000	01
RT	1000	10
RW	1000	11
UM	0101	N/A
WLI	0011	N/A
WLS	0111	N/A
LI	0001	N/A
WB	0000	00
WH	0000	01
WT	0000	10
WW	0000	11
SYNC	0010	N/A

Table 22: CBUS_Y Source Encoding

CBUS_YSRCO[1:0]	Request Source
00	ICACHE
01	DCACHE
10	EJTAG
11	*Reserved*

Table 23: CBUS_Y Destination Encoding

CBUS_YDESTI[2:0]	Description
000	Idle Cycle (no valid data)
100	EJTAG Reply
010	data cache Reply
001	instruction cache Reply

Table 24: CBUS_Y Reply Encoding

CBUS_YLSTEI[2:0]	Description
000	DS
001	DLS (unused)
010	DLE
011	DLM
100	WSA
101	Reserved
110	Reserved
111	UMA (unused)

6.8.3. RLE & RLME Eviction Address

The Eviction address for the RLE and RLME requests is transferred through the CBUS_YDATAO line to the CBUS_Y Request Interface. Since physical addresses are 36-bits and the CBUS_YDATAO bus is 32-bits wide, the entire address cannot be transferred. However, the eviction address is line-aligned, so only 30 bits are required. The format of the address is as follows:

31-30	29-6	5-0
(0)	Address Tag (A35:12)	Address Index (A11:6)

6.9. IBUS Interface

In the ECI application, the IBUS is used to purge the DCACHE eviction buffers. An eviction is caused by sending an IRE inquiry via the IBUS inquiry interface. The DCACHE presents the evicted line on the IBUS reply interface. The IBUS interface runs at the CPU core clock rate.

Table 25: IBUS Request Interface

Signal Name	Direction	Description
IBUS_REQI	output	Inquiry Request
IBUS_CMDI[1:0]	output	Inquiry Request Command
IBUS_CHEI[1:0]	output	Coherency Engine of Inquiry Request (always 2'b00)
IBUS_TIDI[3:0]	output	TID Field of Inquiry Request
IBUS_ADDRI[35:0]	output	Address of Inquiry Request
IBUS_DBUSYO	input	data cache Busy

Table 26: IBUS Reply Interface

Signal Name	Direction	Description
IBUS_RDYO	input	Inquiry Reply Ready
IBUS_STARTO	input	Inquiry Reply Header Data Valid
IBUS_HDRDATAO[63:0]	input	Inquiry Reply Header & Data Bus
IBUS_XBUSYI	output	EC Interface Busy

Table 27: IBUS Commands

Crossbar Request Message	IBUS_CMDI[1:0]	Description
II	00	Line Invalidation (unused)
IIE	01	Evict Line and Invalidate (unused)
IDE	10	Downgrade Line State (unused)
IRE	11	Replacement Eviction

6.9.1. IBUS Header Encoding

The header of an IBUS reply transfer is identical to the crossbar header format. The only IBUS reply supported is IRA.

6.10. ECI Actions on CBUS_Y Commands

Table 28: ECI Actions on CBUS_Y Requests

CBUS_Y Request CBUS_YCMDO	EC Interface Commands	IBUS Request [2]	CBUS_Y Reply CBUS_YLSTEI
RL	!EB_Write EB_Burst EB_Instr (possible)	none	DLE
RLM	!EB_Write EB_Burst	none	DLM
RLE	!EB_Write EB_Burst	IRE	DLE
RLME	!EB_Write EB_Burst	IRE	DLM
RS [1]	!EB_Write !EB_Burst	none	DS
UM	UM	none	UMA
WLI	none	IRE	none
WLE	none	IRE	none
LI	none	none	none
WS [1]	EB_Write !EB_Burst	none	WSA
SYNC [3]	none	none	none

Table 29: ECI Actions on IBUS Replies

IBUS Reply CBUS_YCMDO	EC Interface Commands
IA, IEA	can not occur
IRA	EB_Write EB_Burst

Notes:

[1] On sub-line CBUS_Y requests the DCACHE is not checked or updated.

[2] Further CBUS_Y requests are throttled during an IBUS Request/Reply sequence.

[3] For SYNC, the ECI holds all subsequent CBUS_Y requests until all previous writes have completed. It is not necessary to wait for outstanding reads to complete because a context with an outstanding read is not active, and therefore cannot execute a SYNC instruction.

7.1. Interrupt Overview

The LX4580 provides two classes of cross-context interrupts. Each context includes eight flags to indicate pending cross interrupts. (Two classes from each of the four possible source contexts). The pending cross interrupts are signalled through the IP[2] bit of the destination context's CP0 Status register.

The LX4580 has four active high level sensitive hardware interrupt inputs, HW_INT[4:1]. These interrupts are synchronized within the processor and connected to the IP[6:3] of each context's CP0 Status register.

The LX4580 also has a rising edge sensitive non maskable interrupt input, NMI. This interrupt is synchronized within the processor and converted to a pulse which causes an NMI exception within the CPU.

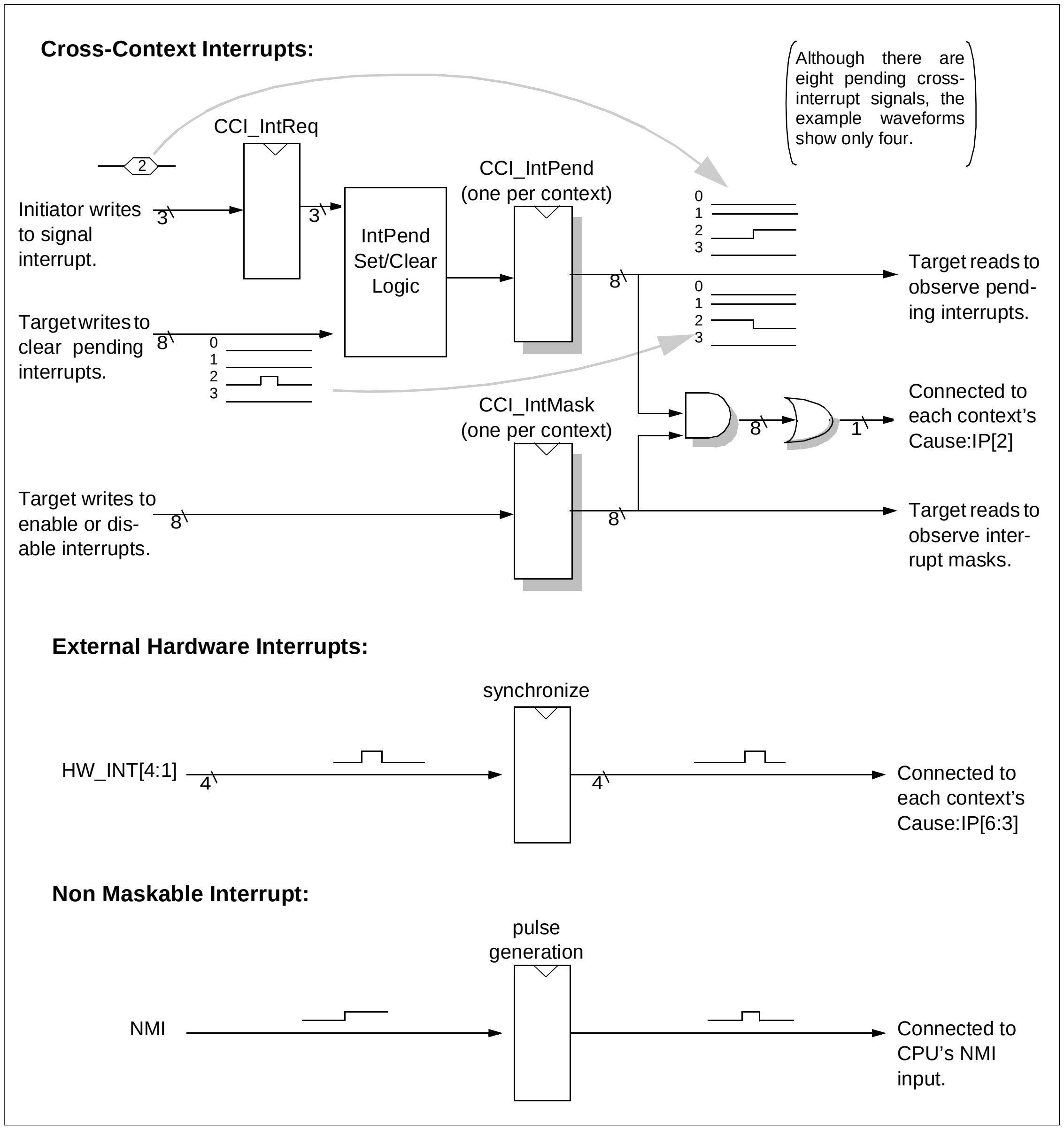
Cross-Context Interrupts:
Although there are eight pending cross-interrupt signals, the example waveforms show only four.
CCI_IntReq
2
Initiator writes to signal interrupt.
3
3
IntPend Set/Clear Logic
CCI_IntPend (one per context)
8
0
1
2
3
Target reads to observe pending interrupts.
Target writes to clear pending interrupts.
8
CCI_IntMask (one per context)
8
1
Connected to each context's Cause:IP[2]
Target writes to enable or disable interrupts.
8
8
Target reads to observe interrupt masks.
External Hardware Interrupts:
synchronize
HW_INT[4:1]
4
4
Connected to each context's Cause:IP[6:3]
Non Maskable Interrupt:
pulse generation
NMI
Connected to CPU's NMI input.

Figure 12: LX4580 Interrupt Logic

Software may send an interrupt to any context on the CPU by executing an uncached store word instruction that specifies the address of the Cross Context Interrupt Request (CCI_IntReq) register. The contents of the word being stored identify the destination context and interrupt class. (The meaning of the two interrupt classes is software defined.) The hardware's interrupt set/clear logic selects the CCI_IntPend register that corresponds to the destination context, and sets the IntPend bit within that register that corresponds to the source context and interrupt class.

The mask bits from each context's CCI_IntMask register are ANDed with cross interrupt pending flags in each context's CCI_IntPend register. The results are reduction ORed and determine the state of the IP[2] bit in the CP0 Status register for each context.

When a cross interrupt event is captured in a CCI_IntPend register, the hardware acknowledges the CPU's write to the CCI_IntReq register. Software that initiates a cross interrupt can use the SYNC instruction to verify that the interrupt is pending at the target (i.e. observable by the target through a read of the target's CCI_IntPend register).

7.1.1. Cross Context Interrupt Request Register (CCI_Req)

Name:	Cross-Context Interrupt Register (CCI_Req)
Size:	32 bits.
Address:	IRR_Base + 0.

31:17	16	15:2	1:0
0	Class	0	Tid

Field	Bits	Description	R/W	Reset
0	31:17	Reserved and must be 0	-	0
Class	16	0 - Set an IntPend0 bit the target context, corresponding to the context ID that is performing the write to CCI_Req. 1 - Set an IntPend1 bit the target context, corresponding to the context ID that is performing the write to CCI_Req.	W	0
0	15:2	Reserved and must be 0	-	0
Tid	1:0	ID of cross-interrupt target context.	W	0

7.1.2. CCI_IntPend Register (One per context)

Name:	CCI_IntPend.
Size:	32 bits.
Address:	AS_Base + 0x0100

31:20	19:16	15:4	3:0
0	IntPend1	0	IntPend0

Field	Bits	Description	R/W	Reset
0	31:20	Reserved and must be 0	R	0
IntPend1	19:16	Bit vector identifying source contexts of class 1 interrupts. 0 - interrupt is not pending. 1 - interrupt is pending.	R/W1C[a]	0
0	15:4	Reserved and must be 0	R	0
IntPend0	3:0	Bit vector identifying source contexts of class 0 interrupts. 0 - interrupt is not pending. 1 - interrupt is pending.	R/W1C	0

One copy of the CCI_IntPend register exists for each context. This register may be read and written only by its associated context. Interrupts for each target context are generated as follows:

INTREQ2_N[target] = | (IntPend & IntMask)

The source context of the interrupt sets IntPend bits by writing to the IRR_CCI register. IntPend bits are cleared by the destination context by writing 1 to the appropriate bit in the IntPend register.

a. W1C = Write 1 to Clear.

7.1.3. CCI_IntMask Register (One per context)

Name:	CCI_IntMask.
Size:	32 bits.
Address:	AS_Base + 0x0104
SW Init:	None.
Restrictions:	None.

31:20	19:16	15:3	3:0
0	IntMask1	0	IntMask0

Field	Bits	Description	R/W	Reset
0	31:20	Reserved and must be 0	R	0
IntMask1	19:16	Mask the IntPend1 bits of the CCI_IndPend register. 0 - pending interrupt does not cause exception. 1 - pending interrupt may cause exception.	R/W	0
0	15:4	Reserved and must be 0	R	0
IntMask0	3:0	Masks the IntPend0 bits of the CCI_IndPend register. 0 - pending interrupt does not cause exception. 1 - pending interrupt may cause exception.	R/W	0

One copy of this register exists for each context, which may be read and written only by its associated context. The IM2 bit in context's CP0 Status must be set for a cross interrupt to cause an exception.

The LX4580 has a fully-featured debug capability which allows full visibility to all LX4580 CPU functions. This capability is based on the MIPS EJTAG Debug Solution 2.0.0 with extra features added to support HMT.

Standard features include:

- Full control of all LX4580 CPUs via the 5 JTAG pins (TCK, TMS, TDI, TDO, RST_N)
- Instruction and Data Breakpoints (number is TBD).
- Hardware single-stepping of any context.
- SDBBP (software debug breakpoint) and DERET (debug exception return) instructions.
- DMA access directly to memory avoiding TLB or cache.
- Instruction jamming to the LX4580 CPU through the EJTAG memory-mapped region.

Features added to support HMT include:

- Choice of which context takes a debug exception when requested by EJTAG.
- Option to disable other contexts when one context takes a debug exception.
- Instruction and data breakpoints match against a particular context, or all contexts.
- Internal PC trace buffers with compression.
- Internal simultaneous PC trace buffering of all contexts.
- Global interrupt of all LX4580 CPUs when a debug exception occurs in one CPU context.
- Optional connection to EJTAG via RS232 UARTE port on LX4580.

LX4580 supplies one set of JTAG pins, through which the TAP controllers for each LX4580 CPU can be daisy-chained together. The TCK and TMS signals are broadcast (so each TAP is always in the same state) and the TDI and TDO are daisy-chained - TDO from CPU0 goes to TDI of CPU1, TDO of CPU1 goes to TDI of CPU2 etc.

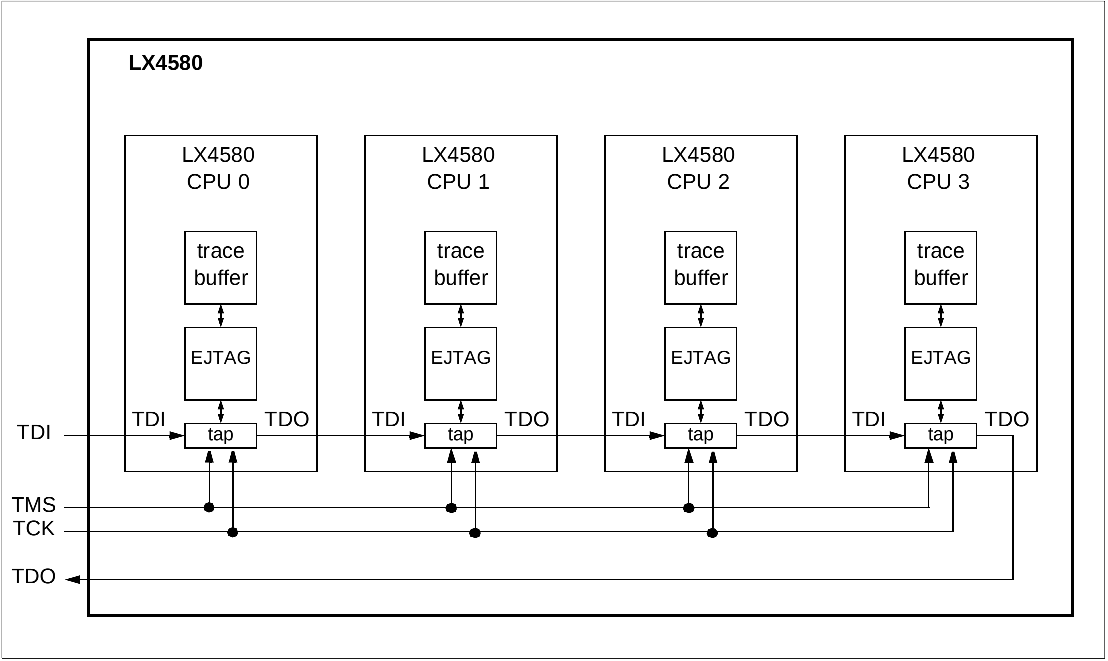

Figure 13: LX4580 ETJAG Organization

THE REMAINDER OF THIS CHAPTER IS FOR INTERNAL LEXRA USE.

8.1. EJTAG Differences from 2.0.0.

The following tables describe implementation options/differences between Lexra's EJTAG solution and the MIPS EJTAG Debug Solution 2.0.0 specification. The O/D column indicates an option or a difference.

8.1.1. EJTAG TAP Registers

Table 30: EJTAG TAP Registers

NAME	OP	Field	O/D	Implementation Specific Information	Reset
Implementation Read-only Register	0x3	0	O	1'b0 Indicates M32	0
		4:1	O	4'b0000 - Obsolete Field	0
		5	O	1'b0 - Instruction Breaks implemented	0
		6	O	1'b0 - Data Breaks implemented	0
		7	O	1'b1 - Processor Breaks not implemented	1
		10:8	O	3'b000 - no external PC trace	0
		13:11	O	3'b000 - no external PC trace	0
		16	O	1'b0 - M16 not supported	0
		17	O	1'b0 - ICache does not keep DMA coherent	0
		18	O	1'b0 - DCache does not keep DMA coherent	0
		19	O	1'b1 - EJTAG_ADDR > 32 bits wide	1
		20	O	1'b0 - Complex Breaks not supported	0
		22:21	O	2'b10 - 8-bit ASID field in implementation	2'b10
		23	O	1'b1 - sdbbp is Special2 Opcode	1
		25:24	O	2'b00- No profiling support	0
		29	D	1'b1 - Lexra Internal Trace Buffer implemented	1
Address	0x8	35:0		36-bit address register - note: Although DMA accesses use 36-bit addresses, CPU accesses use 32-bit addresses which will appear right-justified in this register.	0
Data	0x9	31:0		32-bit data register	0

Table 30: EJTAG TAP Registers (Continued)

NAME	OP	Field	O/D	Implementation Specific Information	Reset
Control	0xA	0	D	PCBufTAC (PC Trace All Contexts) (R/W) 0 - single context traced (RST value) 1 - all contexts traced	0
		5	O	1’b0 DLock not supported (R)	0
		6	D	DOC (Disable other contexts when in DM) 0 - other contexts not disabled when in DM 1 - other contexts disabled when in DM	0
		10	O	1’b0 DMA Error is not supported (R)	0
		13	O	1’b0 DMA Abort is not supported (R)	0
		14	D	SetDev 1 - Debug XCPN vector = 0xBFC00480	0
		15	D	ProbeEn 0 - Debug XCPN vector = 0xBFC00480	0
		19		PrAcc Write not Read. Name incorrect in 2.0.0	0
		20	O	1’b0 PerRst is not supported (R)	0
		23	D	PCBufEn (PC Trace Enable) (W1/R) 0 - Tracing stopped 1 - Start tracing	0
		25:24	D	PCBufMode (PC Trace Mode) (R/W) 2’b00 - Continuous trace mode 2’b01 - Trigger Stops Trace mode 2’b10 - Trigger Starts Trace mode 2’b11 - Reserved	0
		26	O	1’b0 External PC trace not supported	0
		28:27	D	CDM (Context in DM) (R) Displays the context currently in debug mode. Only valid when BrkStatus (bit 3) is set.	0
		30:29	D	CXS (Context Select) (R/W Context to be sent debug exception when Jtag-Brk (bit 12) is set. Only valid when JtagBrk is set.	0
		31	D	WasRst (CPU was reset) (R/W) RST value 1’b0 Reset on a CPU reset. Probe can set this bit to 1 and if it is ever cleared a CPU reset has occurred.	0
All	0xB	99:0		100-bit register containing concatenation of Address, Data and Control registers	
InternalTrace	0xC	-	D	Data from internal trace buffers. Function described below.	1

8.1.2. EJTAG Registers in FF3 (DRSeg)

Below is a table of the options/differences in DRSeg registers with respect to EJTAG 2.0.0. DRSeg starts at logical address 0xFF300000, from which the offsets below are shown.

Table 31: EJTAG DRSeg Registers

NAME	Offset	Field	O/D	Implementation Specific Information	Reset
Debug Control	0	0	O	Trace Mode not supported	0
		1	O	Mask Soft Reset not supported	0
		2	O	Memory Protection not supported	0
		3	O	Mask NMI in non DM not supported	0
		29	O	1’b1 - Endianness (Big)	1
IBS	4	30	O	1’b1 - ASID supported in breaks	1
DBS	8	28	O	1’b1 - Data Break Enhancements	1
		30	O	1’b1 - ASID supported in breaks	1
IBAn	0x100 + 0x10n	1	O	1’b0 - No MIPS16 support	0
IBCn	0x104 + 0x10n	1	O	1’b0 - Complex break no supported	0
		21:20	D	Context Value to match. Causes match for specific context only when CNTXuse enabled.	0
		22	D	CNTXuse (context match use) 0 - Match on any context 1 - Match on context given in Context Value	0
IBMn	0x108 + 0x10n	1	O	1’b0 - No MIPS16 support	0
DBAn	0x200 +0x10n	31:2		Address to match	0
DBCn	0x204 +0x10n	1	O	1’b0 - Complex break not supported	0
		12	O	No Load Breaks supported 0 - data breaks enabled on loads 1 - data breaks disabled on loads	0
		13	O	No Store Breaks supported 0 - data breaks enabled on stores 1 - data breaks disabled on stores	0
		21:20	D	Context Value to match. Causes match for specific context only when CNTXuse enabled.	0
		22	D	CNTXuse (context match use) 0 - Match on any context 1 - Match on context given in Context Value	0
DBMn	0x204 + 0x10n	31:2		Address Mask - 0 address is not masked 1 - address is masked	0

Table 31: EJTAG DRSeg Registers (Continued)

NAME	Offset	Field	O/D	Implementation Specific Information	Reset
DBVn	0x20c +0x10n	31:0	D	Data Value to Match. Only matched on stores. Masked on loads.	0
PB*n	0x300*	31:0	O	Processor Breaks not supported	0

Table 32: COP0 EJTAG registers

NAME	Addr/ Sel	Field	O/D	Implementation Specific Information	Reset
Debug	23/0	6	O	Debug Complex Break Status no supported	0
		10	O	Bus Error not supported	0
		11	O	TLB Exception not supported	0
		13	O	UTLB Miss not supported	0
		14	O	NMI Status not supported	0
		28	O	LSNM not supported	0
DEPC	24/0	31:0		As 2.0.0	0
DESAVE	31/0	31:0		As 2.0.0	0

8.2. Description of LX4580 CPU Specific EJTAG features

8.2.1. Disable Other Contexts (DOC) EJTAG Control Register bit 6

This bit affects the behavior of the CPU only when a context is in DM. When this bit is set it causes other contexts not in debug mode to be disabled no matter what the value of the Disable Context bits in the CP0 LX_CTRL registers. When there are no contexts in debug mode the running state of contexts is determined by their Disable Contexts bits.

Note: there is a skid associated with DOC. Existing instructions in the pipeline complete before other contexts are disabled.

When cleared the DOC bit has no affect.

8.2.2. Context Select (CXS) EJTAG Control Register Bits 30:29

The CXS bits allow selection of which context takes a debug exception on JtagBrk (EJC bit 12) being asserted. Hence, the CXS bits are only valid when JtagBrk is asserted.

8.2.3. Context in Debug Mode (CDM) EJC Bits 28:27

The CDM bits report which context is in debug mode. These bits are only valid when BrkStatus (EJC bit 3) is asserted.

8.2.4. CNTXUse & CNTX in Breakpoint Control Registers

The CNTXUse and CNTX bits in both Imatch and Dmatch control registers allow matches against a specific context, or against all contexts.

8.2.5. Precise Data Breaks

Data Breaks are precise. The load or store that matches the data breakpoint will be squashed.

8.2.6. Data Value Break Loads

Data value breaks on loads are not supported.

8.2.7. EJTAG_ADDR (36-bit)

As the LX4580 has a 36-bit physical address space and a 32-bit logical address space the EJTAG_ADDR register is 36-bits wide to accommodate the physical address.

EJTAG_ADDR is used for 2 functions determined by the DMA Acc bit in the EJC:

DMAAcc = 1 EJTAG_ADDR is read/write and contains the physical address for a DMA transfer.

DMAAcc = 0 EJTAG_ADDR is read-only and contains the logical address of a processor access (only valid with PrAcc is set).

So when DMAAcc = 0 EJTAG_ADDR contains a 32-bit logical address in a 36-bit register. It is padded as follows: {4'b0000, Logical Address}. This does not require a change in behavior, however. As this register is read-only when DMAAcc=0 reading out EJTAG_ADDR only requires 32 shifts, as before, because the logical address is right-justified.

8.2.8. PC Trace Buffer & TAC

8.2.8.1. Overview

The PC trace buffer provides real-time PC trace solution which does not restrict the speed of the CPU and reduces the pin count which is prohibitive for normal PC trace with multiprocessor systems. It employs an on-chip RAM to store compressed PC trace information for retrieval after-the-fact by the EJTAG probe. Stored in the RAM is all the information needed to fully reconstruct the program flow.

If tracing is enabled, a buffer entry is written on every PC discontinuity. The buffer entry contains the target of the discontinuity, the ASID, the number of sequential instructions executed since the last buffer entry was written, and a trace-point indicator set if a trace point occurred since the last entry was written.

8.2.8.2. CPU EJTAG Block Diagram

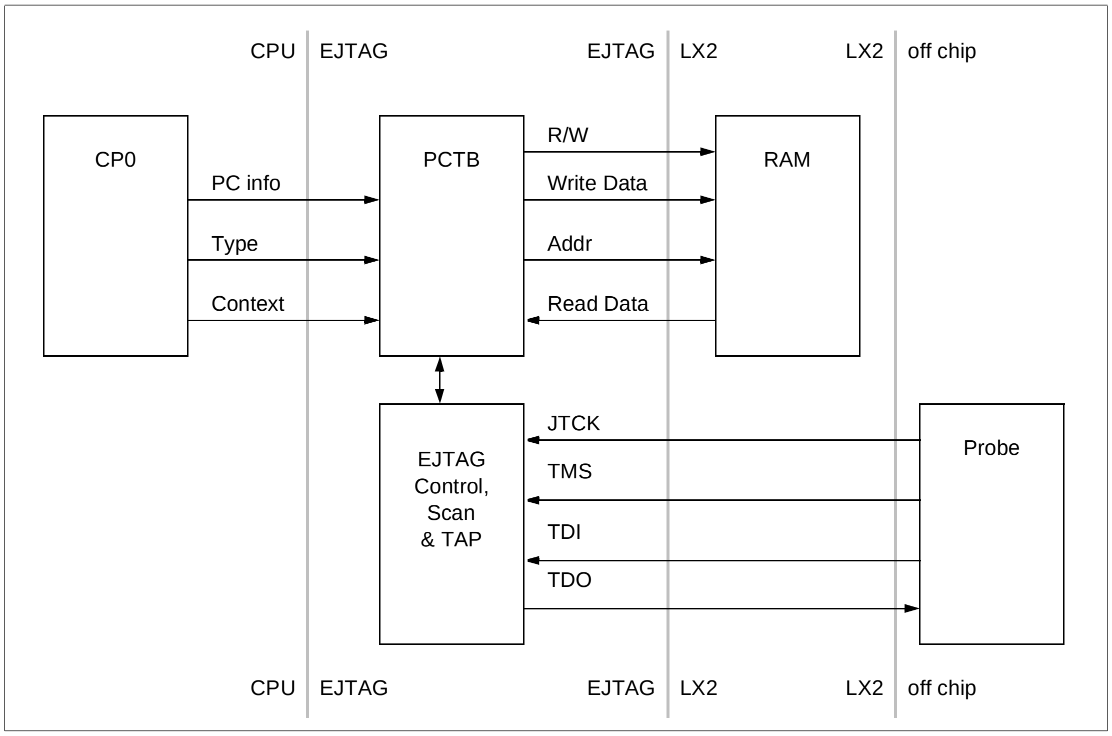

Figure 14: CPU EJTAG Block Diagram

The diagram shows the structure of the PC trace buffer block (PCTB). The PCTB receives pipe-flow information from COP0 every clock cycle. It reads and writes entries into the PCTB RAM.

The control of the PCTB comes from the EJTAG probe which can scan in and scan out control and data via the TAP to initiate PCTB functions.

8.2.8.3. Block Descriptions

COP0	This block sends W-stage signals to the PCTB providing all the pipeline information needed to write buffer entries. Thisinformation comes in the form of the W-stage PC, ASID, contextand instruction-type code.
PCTB	This block is the heart of the PC trace buffer. It receives control from the probe via retimed registers in EJTAG control. When tracing it tests the data from COP0 and decides when to write a buffer entry. It also keeps a count of the sequential instructions executed since the last buffer entry. It handles all the RAM accounting keeping track of the Address, RAM Address wrapping conditions and start/stop tracing conditions.
RAM	This block contains the RAM used by the PCTB. It is about 50 bits wide, and 128 entries deep.
EJTAG Control	This block contains all the retiming registers for control information scanned in by the probe. It also containsthe registers read/written by the probe to control the PCTB.
Probe	This is the EJTAG probe. It is external to the chip. It controls EJTAG by scanning data in and out of registers in EJTAG.

8.2.8.4. RAM Format

Here is the description of a buffer entry (assuming SEQ field width = 8):

Bit	Name	Description
0	VAL	1'b0 marker at the beginning of each entry to indicate that the memory entry is ready to be scanned (through the asynchronous interface)
1	TRIG	a trigger occurred between this and the previous entry
9:2	SEQ	number of sequential instructions since last entry (saturates)
17:10	ASID	ASID
48:17	PC	PC (Logical Address)

Here is the description of a header entry:

Bit	Name	Description
0	VAL	1'b0 marker at the beginning of each entry to indicate that the memory entry is ready to be scanned (through the asynchronous interface)
4:1	SEQW	Width of SEQ field in buffer
10:5	RSVD	Reserved = 6'b000000
25:18	ASIDW	Width of ASID field. Always 4'b0100
14:30	BUFE	Number of valid buffer entries.

8.2.8.5. Mode of Operation

PCTB function is controlled via bits in the EJTAG control register. These bits are bit 23 - PCBufEn (previously "Sync") bits 24-25 - PCBufMode (previously PClen) and bit 27 - PCBufTAC (trace all contexts). Bit 29 in the Implementation Register informs the probe of the presence of the buffer and enables the secondary definition of the four mode bits.

PCBufEn[a] bit 23	**PCBufMode bits 25:24**	**Buffer Mode[b]**
1	00	Continuous wrap (reset-state)
1	01	Any trigger stops trace ("trigger-stop")
1	10	Any trigger starts trace ("trigger-start"), trace stops when buffer full[c]
1	11	Reserved
0	xx	Buffer disabled/reset (reset-state)

a. PCBufEn is set by writing a 1 to it only during debug mode. A 0->1 transition resets the buffer.
b. The buffer mode can be scanned in at any time, even when not in debug mode. If the mode changes during tracing, PCBufEn will be cleared, stopping trace.
c. Buffer full means that it wrapped around to the first entry

PCBufTAC bit 27	**Contexts Traced**
1	Trace all contexts
0	Trace context currently in debug mode.

PCBufEn cannot be cleared directly by the probe. The hardware clears the bit in a number of cases:

1. Buffer full after a trigger (for trigger start and stop modes).

2. By the probe scanning 0x0c into JTAG Instruction register. (to read out buffer entries).

3. Changing the PCBufMODE bits.

PCTB for a Single Context

When PCBufEn=1 the trace buffer continues to fill until a stop condition occurs or debug mode is entered. When debug mode enters, the trace buffer records an entry for the debug exception vector address (usually 0xff20_0200). On exiting debug mode, the trace buffer records the DEPC address.

In trigger-start mode, when a trigger breakpoint occurs, the trace buffer marks its most recent entry as the beginning of the buffer. The trace buffer will continue to record entries until it wraps around to this start and then will stop recording.

In trigger-stop mode, the trace buffer continues to record entries until a trigger breakpoint occurs. The trace buffer will then record one more entry.

When PC Trace stops, the PCBufEn will be cleared. The probe will then scan the data out of the buffer. To restart the trace PCBufEn must be set. This resets the buffer.

To read the buffer, the probe reads each entry sequentially. A new TAP Instruction Register opcode selects the buffer to be read. This opcode is 0x0C. The first time the probe scans the data register (DR) after scanning in a 0x0C opcode, it gets the header that describes the widths of the entry fields and the number of entries.

When the TAP controller reaches the “update” state, the hardware reads the first entry into a scan buffer (through an asynchronous interface since the memory will be operating using SYSCLK). By the time the TAP comes back around to the first entry, the scan buffer will have the data from the first entry. To indicate that data is ready, the first bit to be scanned is 1’b0. If for some reason the TAP controller is faster than the asynchronous interface and data is not yet ready in the scan buffer, the hardware will scan out 1’b1 until data is ready.

After the first entry has been scanned out, the TAP will leave the scan state. As it passes through the update state, the next sequential entry is loaded into the scan register.

Note: that it is more traditional that the scan buffer be loaded in the “capture” state. However, since the asynchronous interface is going to take several cycles, using the previous scan’s update state will get a head-start on fetching the entry.

If the probe tries to scan out data past the number of entries given in the header, the data read back from 0x0c will be 1’b1 indicating data not valid.

If the probe changes the JTAG instruction register from 0x0c, the buffer data will be lost. It cannot return to 0x0c and read out more data. If it does the data will again be 1’b1 - not valid.

PCTB for All Contexts

One extra mode of operation is added to support HMT - tracing of all contexts simultaneously. In this mode data in the buffer is uncompressed and the SEQ field is used as a CONTEXT field to indicate to which context the data corresponds. This information shows the interaction between threads.

Access to the buffer is the same as when a single context is being debugged.

When the probe starts PC trace all pointers and counters are reset such that any data in the buffer from the previous trace will be lost.

8.2.9. Instruction Replay

One artifact of HMT is that some instructions (loads and stores) can be speculatively fetched or replayed. When instructions are jammed there is a possibility that a particular load or store may have to be jammed a number of times before it is executed. This condition is easily detected as the address reported in EJTAG_ADDR will not increment when a REPLAY has occurred.

8.2.10. DMwait

The HMT CPU implements a DM Wait feature which prevents more than one context from executing in Debug mode at any given time. As there is only one instance of various CP0 registers (such as DESAVE) used to support Debug mode. Furthermore, the EJTAG probe software is unlikely to support intermixed accesses to the Dmseg and Drseg regions. Therefore, after one context begins executing in Debug Mode (due to an EJTAG exception) any other context which takes an EJTAG exception is placed in the DM Wait queue.

While in the DM Wait state, the context does not issue any instructions. When the first context leaves Debug Mode (by executing a DERET instruction), the next context in the DM Wait queue resumes execution (in the EJTAG exception handler).

8.2.11. Debug Mode Overrides Disable Context

If a debug exception occurs on a context which is disabled via the Disable Context bits in the COP0 LX_CTRL, the context will become enabled whilst in debug mode. This will allow the debug exception to be taken. On exit from debug mode the context will disabled as per its Disable Context bit.

8.2.12. EJTAG BOOT

Via EJTAG BOOT any LX4580 can start execution from probe space after reset. In this mode the reset vector is changed to 0xFF200200. Thus the first instruction fetched by the LX4580 is from the probe.

This mode is controlled by the value of EJC.ProbeEn at reset. In order for the ProbeEn bit not to be reset itself it has its own dedicated reset pin on lx2 - PRBENRST_D1_R_N. This pin should only be asserted on a cold reset, and should not be asserted when JTAG_RST_N is asserted.

To enter EJTAG BOOT mode the following steps should be taken:

Power-up LX4580 normally and assert/de-assert cold reset (CRESET_N)

Connect the EJTAG probe to the LX4580 and set the ProbeEn bit in the EJTAG Control Register.

Assert/de-assert JTAG_RST_N

The LX4580 will now be fetching from 0xFF200200. The COP0 DEPC register will be 0x00000000.

8.2.13. The Lexra Probe

The Lexra Probe is internal logic which allows a host PC via an RS232 connection to control EJTAG in the LX4580. A null modem cable is connected from the host PC to 2 pins on the LX4580 - EJTAG_RX and EJTAG_TX pins. With this scheme no external probe is required as any PC can be connected to the LX4580 with great ease via a serial connection.

The Lexra Probe logic resides inside the EJTAG block.

EJTAG commands are sent to the Lexra Probe in ASCII over a RS232 UART connection. These commands are translated into JTAG commands sent via the JTAG pins to the EJTAG logic. Responses to the commands are returned in ASCII form from the Lexra Probe to the host.

Lexra provides software interfaces between the Lexra Probe and mainstream debugger tools such as GHS and X-Ray.

8.2.14. Access to EJTAG Memory Space

The following table describes the target of different memory accesses as implemented by Lexra.

M- Main Memory

P- EJTAG Probe (DMSEG)

Reg - EJTAG Registers (DRSEG)

```
            Access Address
DBM ProbeEn FF2 FF3
0   0       M   M
0   1       M   M
1   0       P   Reg (differs from EJTAG 2.0.0*)
1   1       P   Reg
```

*For completeness, the EJTAG 2.0.0 specification has the following difference:

```
DBM ProbeEn FF2 FF3
1   0       M   Reg
```

9.1. Interfaces

Table 33: Interface Summary

Name	Pins	Direction	Description
Clocks and Reset (5 pins)			
CPUCLK	1	input	CPU core clock.
CPUCLK_N	1	input	CPU core clock differential input.
CRESET_N	1	input	Cold Reset.
JTAG_RST_N	1	input	Connection from the EJTAG probe.
CFG_BIGENDIAN	1	input	Static signal designating power-up endian mode. 1 - big endian. 0 - little endian.
Interrupts (4 pins)			
HW_INT[4:1]	1	input	External hardware interrupt lines. Active high.
Chip Test (4 pins)			
JTAG_TDI	1	input	JTAG test data in.
JTAG_TDO	1	output	JTAG test data out.
JTAG_TMS	1	input	JTAG test mode select.
JTAD_TCK	1	input	JTAG clock.
EJTAG Software Debug (4 pins)			
EJTAG_DTR	1	output	EGJAG data terminal ready.
EJTAG_DSR	1	input	EJTAG data set ready.
EJTAG_RXD	1	input	EJTAG receive data.
ETJAG_TXD	1	output	EJTAG transmit data.
EC Interface (186 pins) *See MIPS “EC^{tm} Interface Specification”, Revision 1.05*			
EB_A[35:2]	34	output	Address bus.
EB_ARDY	1	input	System address ready.
EB_AVALID	1	output	CPU address valid.
EB_BE[7:0]	8	output	Byte enables.
EB_BFIRST	1	output	First address cycle.
EB_BLAST	1	output	Last address cycle.
EB_BLEN[1:0]	2	output	Burst length (encoded).
EB_BURST	1	output	Burst transaction.
EB_EWBE	1	input	Write buffer empty.

Table 33: Interface Summary

Name	Pins	Direction	Description
EB_INSTR	1	output	Instruction transfer.
EB_RBERR	1	input	Bus read error.
EB_RDATA[63:0]	64	input	Read data.
EB_RDVAL	1	input	Read data valid.
EB_SBLOCK	1	input	Static configuration for sub-block ordering.
EB_WBERR	1	input	Bus write error.
EB_WDATA[63:0]	64	output	Write data.
EB_WDRDY	1	input	Write data system ready.
EB_WRITE	1	output	Write transfer.
WB_WWBE	1	output	Wait for write buffer empty.
CBUS_Z Interface (182 pins)			
CBUS_ZREQO	1	output	0 - no request, 1 - processor is initiating a request
CBUS_ZBUSYI	1	input	1 - External logic cannot accept request. The current CBUS_Z request, if any, is ignored by external logic. 0 - External logic is ready to accept a request.
CBUS_ZADDRO[35:0]	36	output	Address
CBUS_ZREADO	1	output	1=Read, 0=Write
CBUS_ZSYNCO	1	output	1=Sync request, 0=Normal Request (CBUS_ZREAD will indicate write on sync cycles)
CBUS_ZSZO[1:0]	2	output	Transfer size 2'b00 - 1 byte 2'b01 - 2 bytes 2'b10 - 3 bytes 2'b11 - 1 word This signal is don't care when CBUS_ZLINEO is asserted.
CBUS_ZLINEO	1	output	1 - line access, 0 - single access
CBUS_ZDATAO[63:0]	64	output	Write Data
CBUS_ZLTIDO[1:0]	2	output	Local thread ID
CBUS_ZUCO	1	output	1 - uncached access, 0 - cached access
CBUS_ZSRCO[1:0]	2	output	transaction source (within LX4580): 2'b00 Instruction Cache 2'b01 Data Cache 2'b10 EJTAG 2'b11 reserved
CBUS_ZRDYI	1	input	Read data is available

Table 33: Interface Summary

Name	Pins	Direction	Description
CBUS_ZDBUSYO	1	output	1 - LX4580 is not ready to receive Data. External logic must hold CBUS_ZDATAI, CBUS_ZLTIDI, and CBUS_ZVALTYPEI until CBUS_ZDBUSYO is de-asserted. 0 - LX4580 is ready to receive Data.
CBUS_ZDATAI[63:0]	64	input	Read Data
CBUS_ZLTIDI[1:0]	2	input	Thread associated with Read Data
CBUS_ZVALTYPEI[1:0]	2	input	Indicates read data type: 2’b00 Instruction Cache 2’b01 Data Cache 2’b10 EJTAG 2’b11 reserved