Stream Processor (SP-1)
Data Sheet

Revision 2.1

August 1, 2002

Lexra, Inc.

Proprietary and Confidential

DO NOT COPY

COPY NUMBER ____

Stream Processor Data Sheet Revision 2.1 August 1, 2002

.

Table of Contents

List of Figures

List of Tables

1.1. Introduction

The Stream Processor (SP-1) IC provides the ultimate in both performance and flexibility required to execute demanding network communication applications on 1 Gigabit per second packet streams. The SP-1 includes four (4) of Lexra's LX4580 RISC CPUs, with independent L1 caches, that operate in parallel on independent packets or on independent packet flows. The LX4580 CPU implements the MIPS32™ ISA, with additional specialized instructions for optimized packet processing. Peak processor performance is 2800 Dhrystone 2.1 MIPS. The CPUs also incorporate 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 CPUs are interconnected to shared resources through a high bandwidth full-duplex Crossbar. The Crossbar provides a peak bandwidth of 384 Gbps and can sustain 50 Gigabits per second full duplex data bandwidth to shared SDRAM. Shared resources include a Memory Subsystem with a unified Level 2 cache and cache Coherency Engine. The Memory Subsystems includes two (2) SDRAM controllers with 64-bit data interface to external DDR SDRAM. The Coherency Engine provides hardware support for cache coherency among all CPUs.

Typically, packets enter and exit the SP-1 through the Ethernet MAC interfaces. These interfaces are full duplex and operate at up to 125 MHz with 8-bit data buses to support 1 Gbps traffic. The intelligent DMA Controller associated with each MAC interface allows packets to be queued in off-chip SDRAM without interrupting the application program. Similar DMA Controllers are associated with other SP-1 interfaces. Three (3) independent MAC interfaces are provided so that the SP-1 can be used in applications requiring moderate connectivity without the need for extra board-level components.

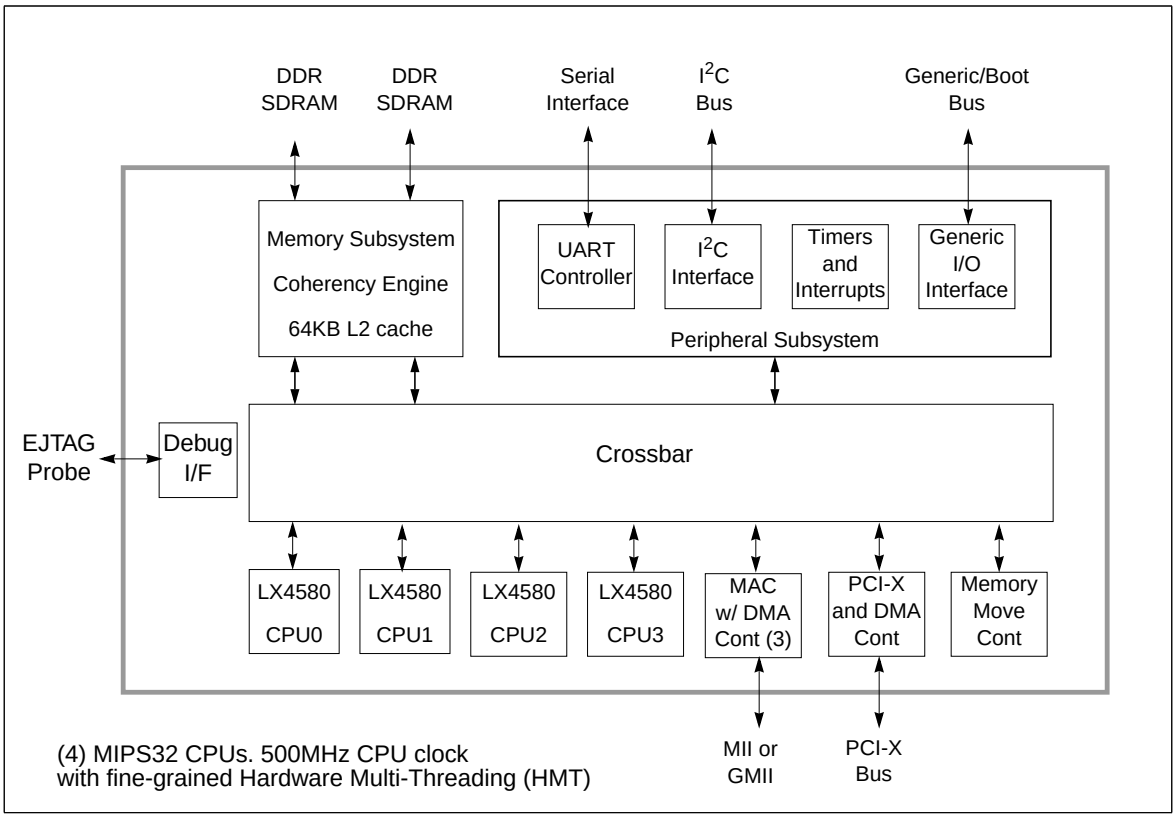

Figure 1: IC Diagram

The Memory Move Controller transfers data from one region of SDRAM to another. This is useful, for example, for transferring incoming packets between temporary buffers and specialized micro-flow queues

for later processing by the application program. Again, the Memory Move Controller performs its transfers without program intervention.

The SP-1 incorporates high performance industry standard I/O interfaces to permit the SP-1 to be easily designed into a wide variety of systems. In addition to Ethernet MACs, a PCI-X interface is included for communication with coprocessors. The generic I/O bus permits simple, low-cost connections to boot EPROMs, and micro-control components. The I^2C interface allows an SP-1 application to configure and monitor other components in the system. The UART provides an interface to a debug console or remote access port. The EJTAG interface enables industry-standard debug software.

Target applications for the SP-1 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 SP-1 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 SP-1 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 SP-1 provides TCP termination, iSCSI protocol conversion and security services.

1.2. Key Features

- (4) MIPS32™ RISC CPUs with Lexra ISA extensions and fine-grained Hardware Multi-Threading (HMT). Each CPU includes a 64KB L1 instruction cache and a 16KB L1 data cache.
- Crossbar Interconnect. 64-bit data ports. Full-duplex. Peak data bandwidth, 32 Gbps/port.
- 64-bit Memory Subsystem. Level 2 cache memory, coherency engine and controller for 2 external SDRAM interfaces. Total unified Level 2 cache memory, 64 KB. Total external DDR SDRAM up to 8 GB, external SDRAM bandwidth up to 50 Gbps.
- DMA Controllers and Memory-to-Memory Move Controller. Background data transfer between I/O interfaces and SDRAM or between two SDRAM locations.
- (3) 10/100/1000 Ethernet MAC Interfaces (MII or GMII).
- (1) PCI-X Interface, 32 bits at 133 MHz.
- (1) 32-bit Generic/Boot bus.
- (1) I^2C Interface.
- (1) UART Interface.
- (2) System timers.
- Support for leading operating systems Linux®, VxWorks®.
- Complete SP-1 Development Platform.

1.3. Specifications

- Technology: 0.13 µm CMOS.
- 500 MHz processor clock.
- 2800 Dhrystone 2.1 MIPS at 1.4 DMIPS/MHz/CPU.
- Power dissipation: 5 W (worst-case)
- Voltage: 1.2 V
- Operating temperature: 0°C to +70°C (commercial).
- Package: 676-pin PBGA.

1.4. SP-1 Architecture

1.4.1. LX4580 CPU

The SP-1 incorporates four (4) LX4580 CPUs, illustrated in Figure 2. The LX4580 is a complete RISC processor subsystem, optimized for high-performance packet processing.

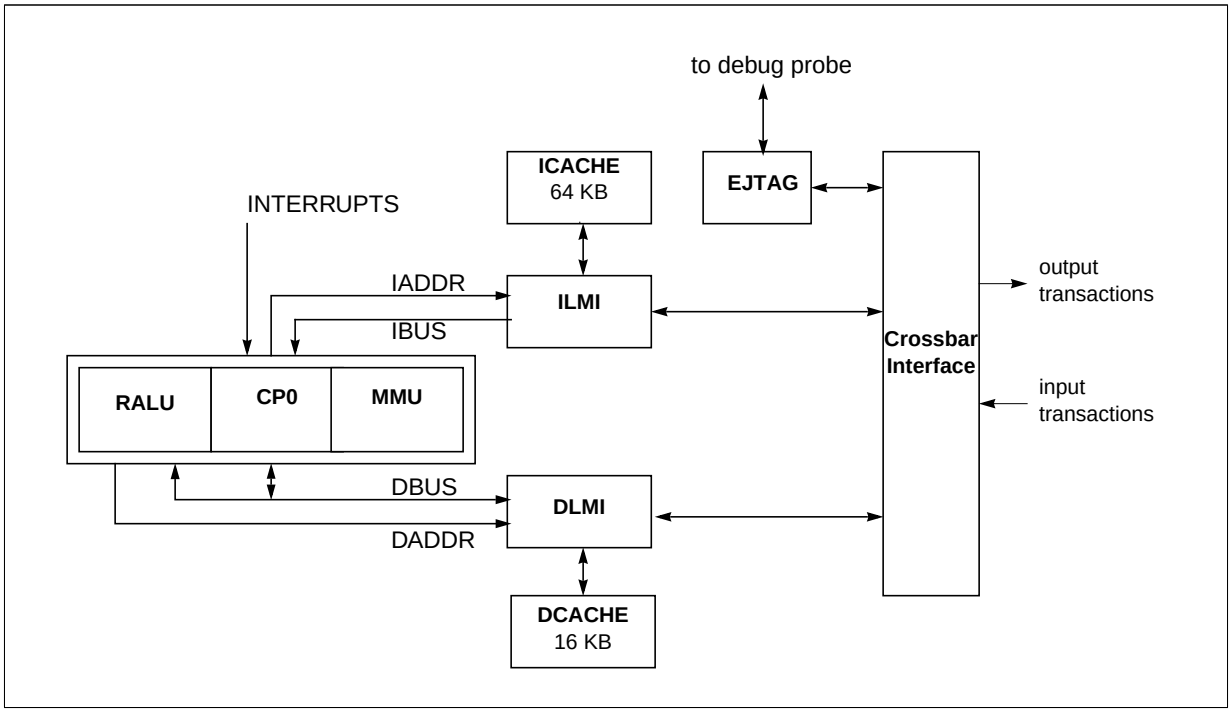

Figure 2: LX4580CPU Diagram

The major blocks are the Register file and ALU (RALU), Coprocessor 0 (CP0), the Local Memory Interfaces (LMI) to 64KB instruction cache and to 16KB data cache. The Crossbar interface unit includes a write buffer and provides a 64-bit bi-directional split transaction interface to the Crossbar.

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 address-register-to-data-output-register cycle for both L1 instruction cache read and L1 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 SP-1 customer that deserve attention in this overview.

In Lexra's implementation, instructions are issued round-robin to four alternate pipeline flows. Each pipeline flow 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 pipeline flows support four independent execution threads.

In typical network processing programs L1 data cache miss rates are high and a single-threaded processor is idle much of the time. The problem can be mitigated somewhat with an on-chip Level 2 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 L1 cache miss, and 50% of the L1 cache misses result in an L2 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 SP-1 target applications.

1.4.3. Crossbar Interconnect

The Crossbar provides a high bandwidth full-duplex interconnect between the CPUs, DMA Controllers, main memory, on-chip Level 2 cache and external I/O interfaces. The Crossbar operates at the SP-1's system clock rate (1/2 the CPU clock). The Crossbar's internal data paths are 64-bits wide. Distributed queuing is used, internal to the Crossbar, to eliminate head-of-line blocking. Ports can transfer 64-bits per cycle in each direction and achieve peak data bandwidth of 32 Gbps/port.

The Crossbar can transfer words or sub-words. It also supports processor L1 cache line read and write transfers (64 bytes). The Coherency Engine in the Memory Subsystem communicates with the CPUs using Crossbar coherency signalling to insure that cache coherency is maintained in processor memory transactions, without software intervention.

The Crossbar provides a point-to-point messaging protocol. Resources connected to the crossbar may operate as an *initiator*, as a *target*, or both. Messages of a similar type between the same initiator-target pair are

always delivered in order. Error messages are used to report memory system errors. Interrupt messages are used for processor-to-processor and device-to-processor signalling.

1.4.4. DMA Controllers

The Stream Processor provides multiple DMA controllers for high-speed data transfer between the main memory and external interfaces, and from memory to memory. Each Ethernet MAC and PCI-X interface includes a dedicated DMA Controller. The Memory Move Controller is a specialized DMA controller, used for transfers between source and destination buffers located in the Stream Processor's main memory. The Ethernet MAC and PCI-X DMA controllers provide sustained full duplex data bandwidth of 1 Gbps in each direction. The Memory Move DMA controller provides a data bandwidth of up to 10 Gbps.

Software can atomically enqueue new DMA requests and dequeue completed requests using simple stores and loads to memory mapped registers within the DMA Controller, without requiring an interrupt for each completed transfer. The controller provides atomicity for these operations so that multiple CPUs may access the queues without prior synchronization or resource locking. Main memory data buffers are organized in linked lists of buffer elements. The size and number of buffers in the list is software-defined.

1.5. Interfaces

Table 1 summarizes the interfaces provided by the Stream Processor.

Table 1: Summary of Stream Processor Interfaces

Name	Qty	Performance	Function
SDRAM	2	64-bit data. 133 MHz clock up to 26 Gbps/interface	Incoming/outgoing packet queues, micro-flow queues, state tables.
MII/GMII	3	Up to 125MHz. clock. Full -duplex. 8-bit Rx data, 8-bit Tx data.	Point-to-point full duplex packet interface. Glueless connection to a wide variety of standard 10/100/1000 PHYs and switches.
PCI-X	1	133 MHz clock. 32-bit data. Total bandwidth is 4.2 Gbps	The leading high-performance embedded system bus standard.
Generic/Boot	1	32-bit asynchronous I/O bus. 32-bit A/D or 24-bit A/8-bit D modes.	Glueless address/data interface to devices such EPROMs, microcontrollers. Programmable timing and control.
I^2C	1	Shared serial bus interface. Up to 3.4 Mbps. Full-duplex.	Low-cost, inter-chip control. Conforms to I^2C Version 2.1, Phillips Semiconductor.
Serial	1	Serial I/O, up to 614,400 baud.	Simple 4-wire interface to applicaition-specific debug terminal.
EJTAG	1	40 MHz clock.	Scan chain debug. Conforms to EJTAG 2.0. Provides PC-trace. Multi-processor support.

1.6. Software Support

1.6.1. Operating Systems

Two operating systems are provided for the Stream Processor:

- 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

The following development tools are available:

- The MontaVista™ toolchain is available on more than 12 development platforms including RedHat and Solaris.
- The GNU toolchain (compiler version 3.0.3 or higher; binutils version 2.11.2 or higher) is available and is supported by Lexra for cross-compilation from Linux, Solaris, and Windows/Cygwin hosts.
- The Tornado® toolchain is available from Wind River® on Solaris and Windows for VxWorks® development.
- An EJTAG debugging solution is available from Embedded Performance, Inc.
- SP-1 Development Board. All of the above software tools and operating systems are supported on Lexra's SP-1 Development Board. The Board provides PHY and Link Layer components to route packets to and from the SP-1. The SP-1 packet and control interfaces can be connected to customer-defined boards to provide a complete, realistic development environment.

1.6.3. Sample Applications

Sample applications and device drivers showing how to use the interfaces of the processors are available from Lexra in source form.

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 used in the Stream Processor. 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)
Write by another entity outside this CPU will invalidate the line
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, 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 GTId 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	1 (always BigEndian)
				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 Icache size, 4 ways
				DS,DL,DA	64-byte linesize,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		
CacheErr	27		1		TBD format
TagLo	28	0	1		TBD format
DataLo	28	1			not implemented
TagHi	29	0			not implemented
DataHi	29	1			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 interrupts (See Section 4.2.3)
IP3	Hardware 1	Device interrupts (See Section 4.2.2)
IP4	Hardware 2	Hardware error interrupt from Interrupt Reflector (See Section 4.2.4)
IP5	Hardware 3	0 (reserved) TBD (for off-chip real time interrupt?)
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		TBD (for errors)
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	
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	
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 15. 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 is always selected. 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 15.2.8.

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.

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
000110	Dcache load
000111	Dcache store
001000	Dcache load miss
001001	Dcache store miss
001010	Dcache load or store
001011	Dcache load or store miss

Table 7: Event Field of PerfCnt Control Registers

Value (bits 10:5)	Event Counted
001100	Uncached load or store
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)

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.

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.

3.1. Reset Overview

A comprehensive reset strategy provides reliable initialization of the Stream Processor.

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

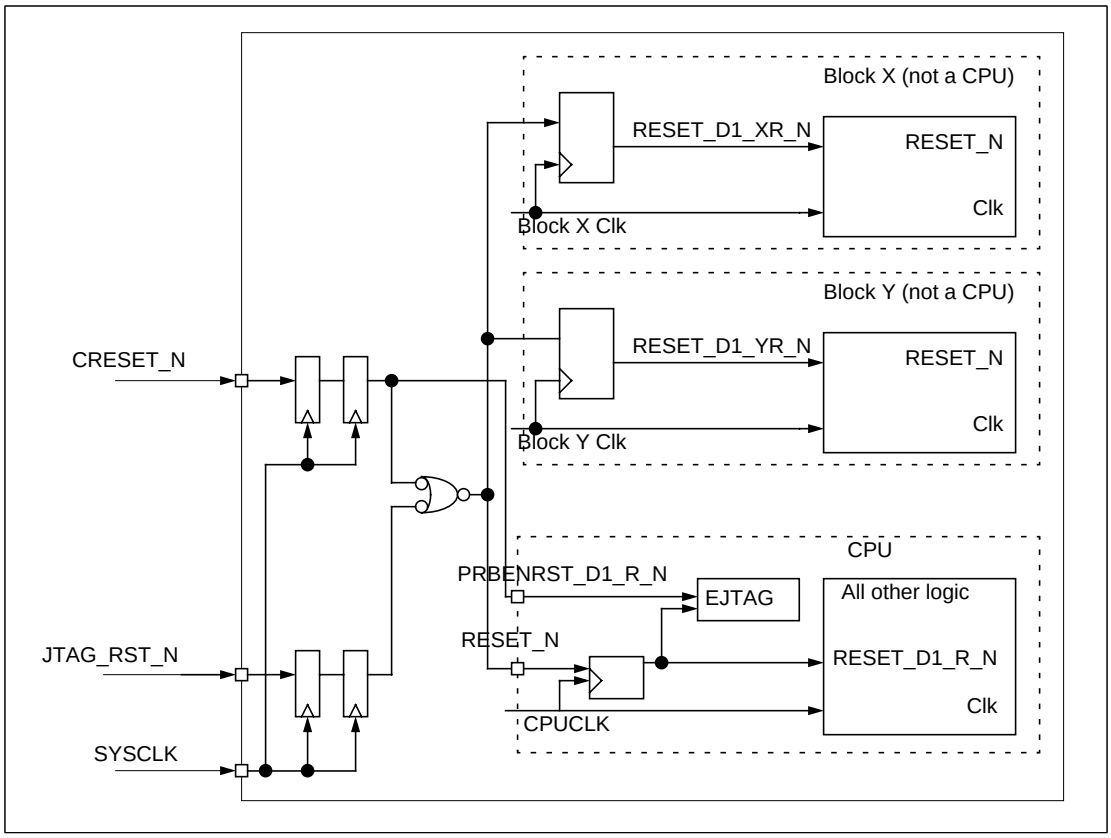

Figure 3: Reset Overview

The reset strategy ensures the following:

- Complete initialization of the Stream Processor by the assertion of one external pin.
- Reset debug by EJTAG. All CPUs 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.
- There is a software register (Test and Set register) in uncacheable space which is used to determine which processor will be the boot master.

3.2. Reset Distribution

The reset system is distributed across the design. Each block or CPU will contain 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 typical multiple CPU cold-boot sequence would be as follows:

1. CRESET_N is asserted.
2. CRESET_N is de-asserted.
3. All CPUs start executing identical software from the reset vector.
4. All CPUs read the Test and Set Register.
5. Due to the nature of the Test and Set Register only one CPU will read it as zero. This CPU will become the boot master.
6. The boot master will initialize the multiple CPU environment and the Stream Processor peripherals. The other CPUs that did not read a zero from the Reset Register will be executing software loops.
7. When the boot master has finished the initialization of the Stream Processor it sends inter-processor interrupts to each of the other CPUs. The other CPUs will start to execute the appropriate interrupt handler after which they operate normally.

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 each processor in turn setting the ProbeEn bit in each CPU's EJTAG Control Register.
4. The Probe then asserts JTAG_RST_N. This resets everything in the Stream Processor apart from the EJTAG ProbeEn flop.
5. JTAG_RST_N is de-asserted and all CPUs jump to the debug exception vector at 0xFF200200 from where the system is under EJTAG probe control.

3.4. Reset Registers

3.4.1. TestAndSet Register (TAS)

Name: TestAndSet Register (TAS)
Size: 32 bits.
Address: TBD - uncached
SW Init: None.
Restrictions: None.

31	0
TAS	Reserved

Field	Bits	Description	R/W	Reset
TAS	31	After a read this field changes to 1. Writes load the register with value written.	R/W	0

3.5. Reset External SP-1 Interfaces

Table 9: Reset External Interface

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

INTERNAL USE. The TAS register is implemented in the System Control logic. The register description should be moved to that section, and a cross-reference included here.

4.1. Interrupt Overview

The Stream Processor provides a variety of interrupt mechanisms to allow CPU contexts to be efficiently signalling by external sources, integrated devices and other CPU contexts.

The following types of interrupts are supported:

- (6) Edge sensitive interrupts from integrated devices.
- (16) Edge sensitive cross interrupts from CPU context to CPU context.
- (4) Level sensitive external interrupts.
- (1) Level sensitive hardware error interrupt.

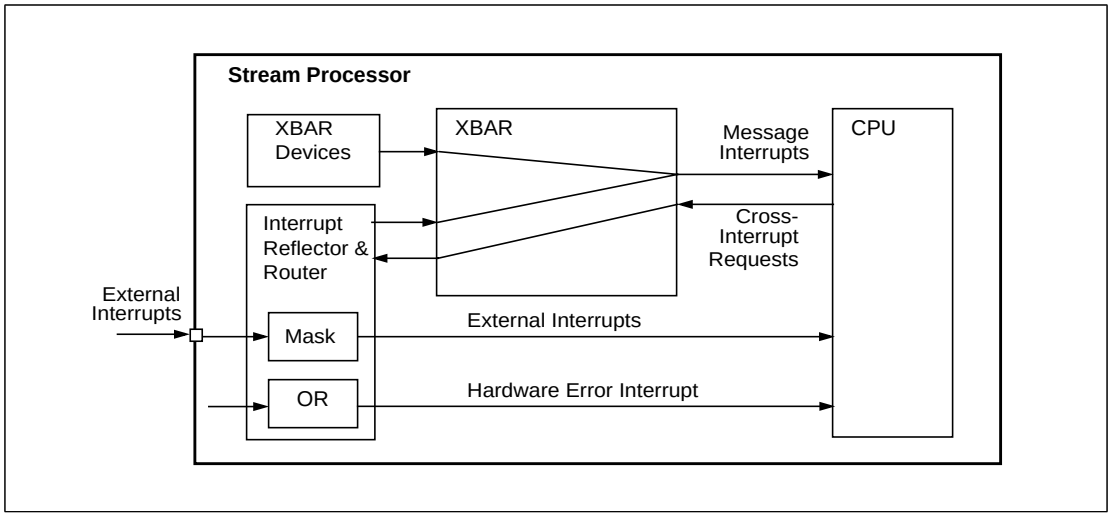

Figure 4: System View of Interrupt

A CPU context can be directly interrupted by crossbar message from an integrated device, or via crossbar message or interrupt line from the Interrupt Reflector and Router (IRR).

The edge sensitive device and cross interrupts use a crossbar message to route an interrupt request from the interrupt source to a single CPU context. The contents of the CPU's XBI and CP0 registers determine how the interrupts are honored by the target.

The level sensitive external and error interrupts are globally distributed throughout the Stream Processor and can be observed by any number of CPU contexts according to the contents of the CP0 registers.

The external interrupts are globally maskable via a register within the IRR.

Software can observe pending interrupts by polling registers within the LX4580 CPU and throughout the Stream Processor, even when the interrupt is disabled through the corresponding mask register.

4.2. Interrupt Architecture

The interrupt architecture details are shown in Figure 5.

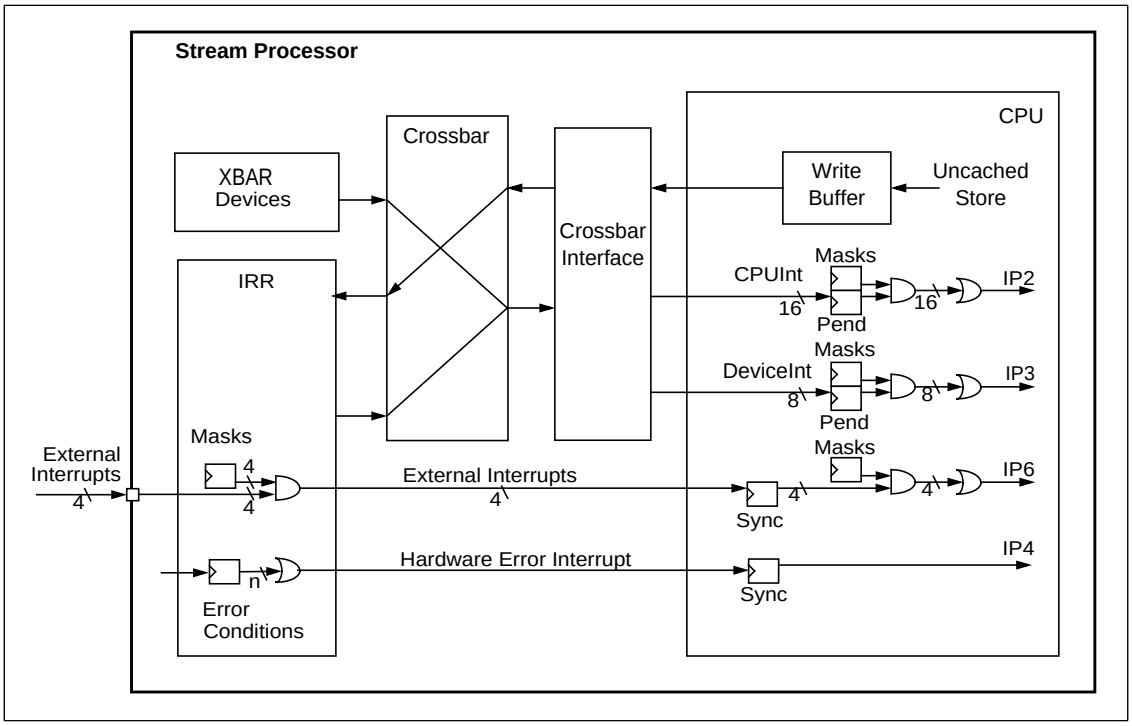

Figure 5: Interrupt Architecture

4.2.1. External Interrupts

External interrupts are passed through the IRR where they can be globally masked. The result is then passed to all CPUs. The interrupts are synchronized at the CPU boundary. The output of the synchronizer is the source of the read-only level-sensitive interrupt pending flags in the XBI EXT_IntPend register. (See Section 8.7.12.) The mask from the XBI EXT_IntMask register is ANDed with the interrupt pending flags. (See Section 8.7.13.) The result is reduction ORed and determines the state of the IP6 bit in the CP0 Status register.

4.2.2. Device Interrupt Messages

Devices interrupt a CPU context via the Interrupt request (IN) crossbar message. At the CPU's crossbar interface the message is converted to one of six edge sensitive device interrupt events. The device interrupt events are captured in the XBI DEV_IntPend register. (See Section 8.7.10.) The mask bits from the XBI DEV_IntMask register are ANDed with the device interrupt flags in DEV_IntPend. (See Section 8.7.11.) The result is reduction ORed and determines the state of the IP3 bit in the CP0 Status register.

When a device interrupt event is captured in the DEV_IntPend register, the CPU sends an Interrupt Acknowledge reply (INA) crossbar message to the device that requested the interrupt.

4.2.3. CPU Cross Interrupt Messages

Software may send an interrupt to any context on any CPU by executing an uncached store word instruction that specifies the address of the Interrupt Reflector (IRR). The contents of the word being stored identify the

CPU context which is the target of the interrupt. The store word instruction generates a Write Word (WW) crossbar message to the IRR.

The IRR in turn interrupts the target CPU context by sending a crossbar Interrupt Request (IN) message to the CPU. At the target CPU's crossbar interface, the message is converted to one of sixteen edge sensitive cross interrupt events. The interrupt events are captured in the XBI CPUX_IntPend register. (See Section 8.7.8.) The mask bits from the XBI CPUX_IntMask register are ANDed with the device interrupt flags in CPUX_IntPend. (See Section 8.7.9.) The result is reduction ORed and determines the state of the IP2 bit in the CP0 Status register.

When a cross interrupt event is captured in the CPUX_IntPend register, the target CPU sends an Interrupt Acknowledgement reply (INA) crossbar message to the Interrupt Reflector, which in turn sends a Write Sub-Line Acknowledgement reply (WSA) crossbar message to the CPU that originally requested the interrupt. Therefore, software that sends a cross interrupt can use the SYNC instruction to verify that the interrupt is pending at the target.

4.2.4. Hardware Error Interrupt

Hardware errors, such as an SRAM parity error, are detected throughout the Stream Processor as described in Chapter 17. Each module supplies an error interrupt signal that is asserted when an error condition has been detected within the module and the generation of an interrupt for that error is enabled. When one or more of the module-specific error signals transitions high, a priority encoded value representing the error is captured in the IRR_ModuleErrorCapture register, and an error flag in this register is also set. This flag determines the state of the IP4 in the CP0 Status registers of all CPUs. See Section 4.3.3 for more details on the IRR_ModuleErrorCapture register.

4.3. Interrupt Registers

4.3.1. IRR External Interrupt Master Mask Register (IRR_EIMM\)

Name: External Interrupt Master Mask Register (IRR_EIMM)
Size: 4 bits.
Address: IRR_Base + <TBD>
Restrictions: None.

31:4	3:0
Reserved	XIMM

Field	Bits	Description	R/W	Reset
XMIM	3-0	Interrupt line enabled if set. Bit 0 masks INT0 etc.	R/W	0

4.3.2. IRR CPU Cross Interrupt Register (IRR_CCI)

Name: Inter-CPU Interrupt Register (IRR_CCI)
Size: 32 bits.
Address: IRR_Base + <TBD>
Restrictions: None.

31:4	3:0
Reserved	TGTid

Field	Bits	Description	R/W	Reset
TGTid	3:0	CPU GTid of interrupt target	W	0

4.3.3. Module Error Capture

Name: IRR_ModuleErrorCapture
Size: 32 bits.
Address: IRR_Base + <TBD>
SW Init: None.
Restrictions: None.

31	30	29:4	3:0
Err	MultiErr	0	ErrModule

Field	Bits	Description	R/W	Reset
Err	31	Indicates an error condition has been captured. 0 - No error condition has been captured. 1 - An error condition has been captured in ErrModule. To clear the contents of this register, software writes a zero to this register field. This should be done after each individual module's error registers have been interrogated and cleared. If an error condition remains pending when zero is written to this field, the error condition is re-sampled and captured in this register.	R/W	0
MultiErr	30	Indicates multiple error conditions are present. 0 - Multiple error conditions have not been detected. 1 - Multiple error conditions have been detected. The module that reported the first error condition is indicated in the ErrModule field.	R	0
ErrModule	3:0	Encoded value representing the first module that reports an error via its ERR_INT output. Valid only if the Err field is 1. 0 - Memory Subsystem 0 1 - (not used) 2 - Memory Subsystem 1 3 - (not used) 4 - CPU 0 5 - CPU 1 6 - CPU 2 7 - CPU 3 8 - device 0 9 - device 1 10 - device 2 11 - device 3 12 - device 4 13 - device 5 14 - device 6 15 - device 7 If more than one module signals an error in the same system clock cycle, the lowest numbered module that signals an error is reported.	R	0

4.4. Interrupt External SP-1 Interfaces

Table 10: Interrupt External Interface

Signal Name	Direction	Description
INT0_N	input	External Interrupt Line 0. Active low.
INT1_N	input	External Interrupt Line 1. Active low.
INT2_N	input	External Interrupt Line 2. Active low.
INT3_N	input	External Interrupt Line 3. Active low.

5.1. Address Space Overview

The Stream Processor's address space defines the location of SDRAM, controllers and external devices within the Stream Processor's 64 GByte (36-bit address) physical address range. Applications can limit the physical address space to 4 GBytes for hardware and software that employ 32-bit physical addressing. All devices and memory that are attached to the crossbar are globally accessible from all processors.

Two spaces are defined to support boot (such as ROM) and control accesses (such as EJTAG, DMA controllers and configuration registers). These spaces have a fixed size and location.

Additional spaces may also be defined for access to external devices that are attached to the Stream Processor through its PCI-X and Generic I/O (GIO) interfaces.

Figure 6 illustrates the domains of logical and physical addressing within the Stream Processor.

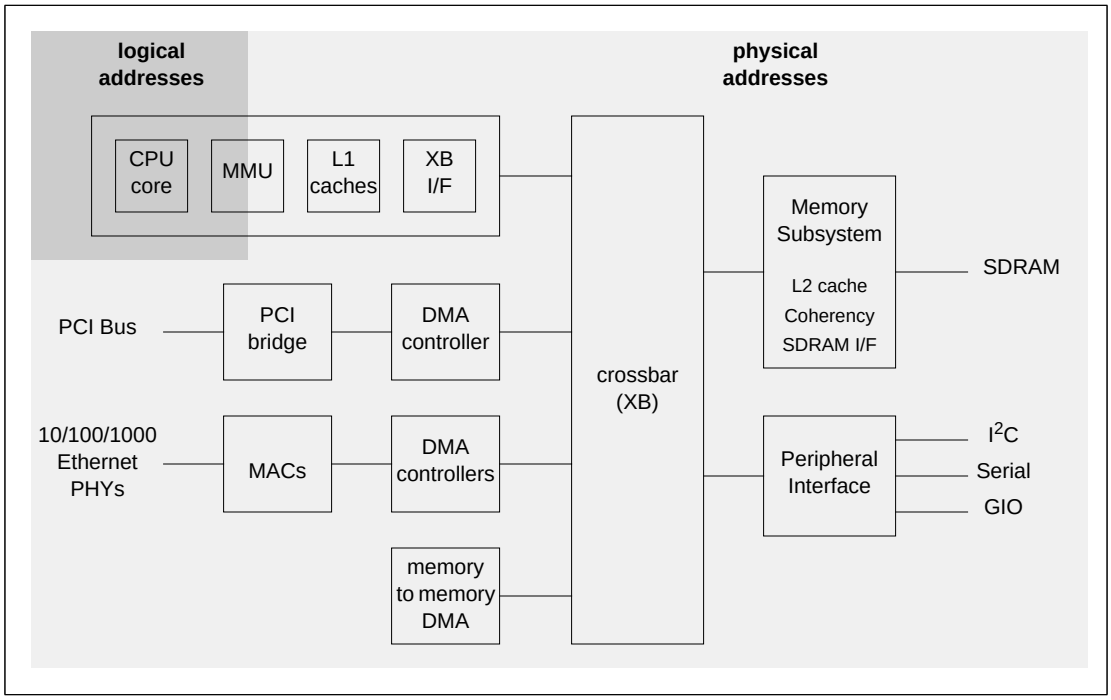

Figure 6: Address Space Overview

The Stream Processor's address space provides the following characteristics:

- 36-bit physical addresses for a total of 64 GBytes of addressable resources.
- Optionally constrained to a 32-bit physical address subset.
- SDRAM address space.
- Kseg1 boot space.
- Kseg0/kseg1 control space for integrated controllers, configuration and status.
- EJTAG debug mode space (dmseg, drseg).
- External address spaces for access to resources through PCI-X and GIO.
- Error detection for access outside of defined address spaces.

5.2. Address Space Size

The Stream Processor supports a 36-bit physical address space (64 GBytes). Within a Stream Processor CPU, the MMU translates 32-bit logical addresses into 36-bit physical addresses prior to referencing the CPU's internal caches or resources outside of the CPU.

Applications need not take advantage of the 36-bit physical address capability. All of the Stream Processor internal resources, such as configuration registers and integrated devices, are located at fixed addresses that fall within the low 4 GBytes of the address space. Application-specific resources such as SDRAM and devices attached to the Stream Processor's PCI-X interface can also be located within the low 4 GBytes of the address space.

5.3. Physical Address Space Decoding

Figure 7 shows how the Stream Processor's physical address space is decoded. Certain spaces are fixed, and have the highest priority in the decode logic. Other spaces are configurable and are decoded in the priority order shown. Address spaces may overlap. A space with higher priority of decode may create a "hole" in lower priority decoded spaces.

See Section 8.7 for a description of the address space configuration registers which apply to steps 2-5 of the diagram.

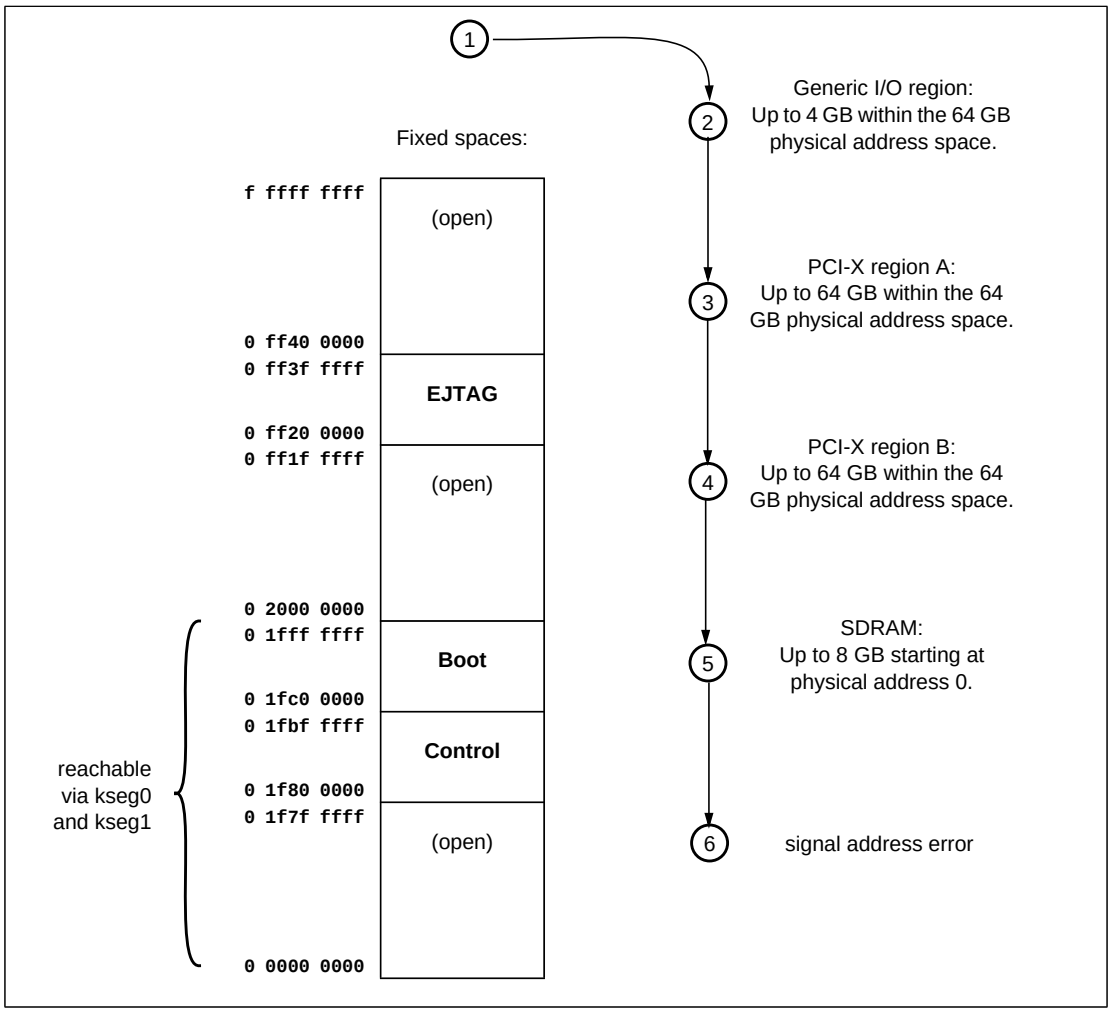

Figure 7: Address Space Decoding

5.4. Boot Space

The boot space is located at 16#0_1fc0_0000 through 16#0_1fff_ffff. The BEV flag of the CPU's Status register is set to 1 by a reset, which causes all CPU exception vectors except EJTAG to be mapped within the boot space.

Generic I/O user device 0 is dedicated to the boot space, and is automatically configured by reset to communicate with a ROM attached to the Stream Processor's GIO interface. (See Section 14.3.) When a CPU is brought out of reset, it begins fetching instructions from the logical kseg1 address 16#bfc0_0000, which corresponds to physical address 16#0_1fc0_0000.

The operating system startup software that runs from the boot space typically configures the SDRAM memory space. After the operating system has copied the appropriate program code to low memory, the operating system may clear the BEV flag in the CP0 Status register. This causes some of the exception vectors to be mapped to locations starting at logical address 16#8000_0200, which corresponds to physical address 16#0_0000_0200.

5.5. Control Space

The control space is located at 16#0_1f80_0000 through 16#0_1fbf_ffff and therefore is accessible by a CPU via the unmapped and uncached kseg1 segment. Software accesses integrated devices through this space to configure the device and to directly interact with the device.

Most of the Stream Processor's integrated devices are assigned a 64 KByte sub-space within control space, as defined in Table 11. Some devices with smaller address space requirements share a 64 KByte control space. The definition of resources within the 64 KByte space is specific to the device.

The operating system may employ the MMU to enforce device access privileges to user mode application code. With an MMU page size of 4 KBytes, access can be given to 16 distinct subsets of a given device's functions. For example, the head and tail window registers of DMA controller's queues are located in a separate 4 KByte page from other DMA control registers. All such MMU pages should specify the uncached access type.

Table 11: Control Space Organization

Control Space	Symbol	Address Range	Description	See Section
MAC0	MAC0_Base	16#0_1f80_0000 to 16#0_1f80_FFFF	Ethernet MAC interface 0 configuration, status and DMA registers.	10.15, 11.4
MAC1	MAC1_Base	16#0_1f81_0000 to 16#0_1f81_FFFF	Ethernet MAC interface 1 configuration, status and DMA registers.	10.15, 11.4
MAC2	MAC2_Base	16#0_1f82_0000 to 16#0_1f82_FFFF	Ethernet MAC interface 0 configuration, status and DMA registers.	10.15, 11.4
PCI	PCI_Base	16#0_1f83_0000 to 16#0_1f83_FFFF	PCI-X bridge configuration, status and DMA registers.	10.15, 12.6
MM	MM_Base	16#0_1f84_0000 to 16#0_1f84_FFFF	Memory to memory DMA	10.15
XI	XI_Base	16#0_1f85_0000 to 16#0_1f85_FFFF	Cross Interrupt functions.	13.2
ST0	ST0_Base	16#0_1f85_2000 to 16#0_1f85_3FFF	System timer 0.	13.3

Table 11: Control Space Organization (Continued)

Control Space	Symbol	Address Range	Description	See Section
ST1	ST1_Base	16#0_1f85_4000 to 16#0_1f85_5FFF	System timer 1.	13.3
I2C	I2C_Base	16#0_1f85_6000 to 16#0_1f85_7FFF	I^2C controller.	13.4
UART	UART_Base	16#0_1f85_8000 to 16#0_1f85_9FFF	Serial port controller.	13.6
GIOC	GIOC_Base	16#0_1f85_A000 to 16#0_1f85_BFFF	Generic I/O configuration registers.	14.7
AS	AS_Base	16#0_1f88_0000 to 16#0_1f88_FFFF	Address space configuration.	8.7
MS	MS_Base	16#0_1f89_0000 to 16#0_1f89_FFFF	Memory subsystem configuration and status.	9.13, 9.14, 9.15

5.6. EJTAG Space

In debug mode, the EJTAG space is reserved for EJTAG. When the EJTAG probe is enabled the CPU detects any accesses to EJTAG space and redirects the transaction to the EJTAG hardware. When the probe is disabled, the decoding of EJTAG reserved space is disabled. Any transaction that would have targeted the EJTAG space in this situation will be directed to another portion of the address space according to the decode priority shown in Figure 7.

It is strongly recommended that applications not use the EJTAG space for any reason, even when a debug probe is disabled or is not present.

5.7. Generic I/O Space

One Generic I/O (GIO) space may be configured through the XBI control registers. (See Section 8.7.) All accesses within this space are directed to the SP-1's GIO interface. Further decoding within the interface can differentiate addresses for up to four separate devices. (See Chapter 14.)

5.8. PCI-X Space

Two PCI-X spaces may be configured through the XBI control registers. (See Section 8.7.) Typically one of the spaces is used to access uncacheable control registers within the PCI-X Bridge or on the PCI-X bus, and the other space is used to access large cacheable memories. (See Chapter 12.)

5.9. SDRAM Space

The top of SDRAM space is configured through the XBI control registers. (See Section 8.7.) The SDRAM space is accessible from all CPU contexts, starting at the physical address 0 and ending at the configured top of SDRAM. Depending on the size of SDRAM, a context may access all or part of SDRAM through the kseg0 and kseg1 logical address ranges.

The special purpose GIO, control and EJTAG spaces may overlap with the SDRAM space. In this case, the special spaces take priority, and SDRAM located at these addresses is not accessible.

5.10. Address Space Configuration Registers

The address space configuration registers are implemented within each CPU crossbar interface and are described in Section 8.7.

5.11. Error Detection

The Stream Processor fully decodes the address of all access made by the CPUs and DMA controllers. All illegal addresses result in a bus error condition being signaled to the originator. See Chapter 17, "Error Detection and Reporting".

6.1. Crossbar Overview

The crossbar (XB) provides the on-chip interconnect that allows the Stream Processor's LX4580 CPUs, DMA controllers, memory and I/O devices to communicate with each other. Crossbar transfers are initiated as a result of events such as a cache miss or the execution of an uncached load or store instruction.

Since Stream Processor software does not interact directly with the crossbar, the material in this chapter is provided to allow users to optionally gain a greater understanding of how the Stream Processor functions.

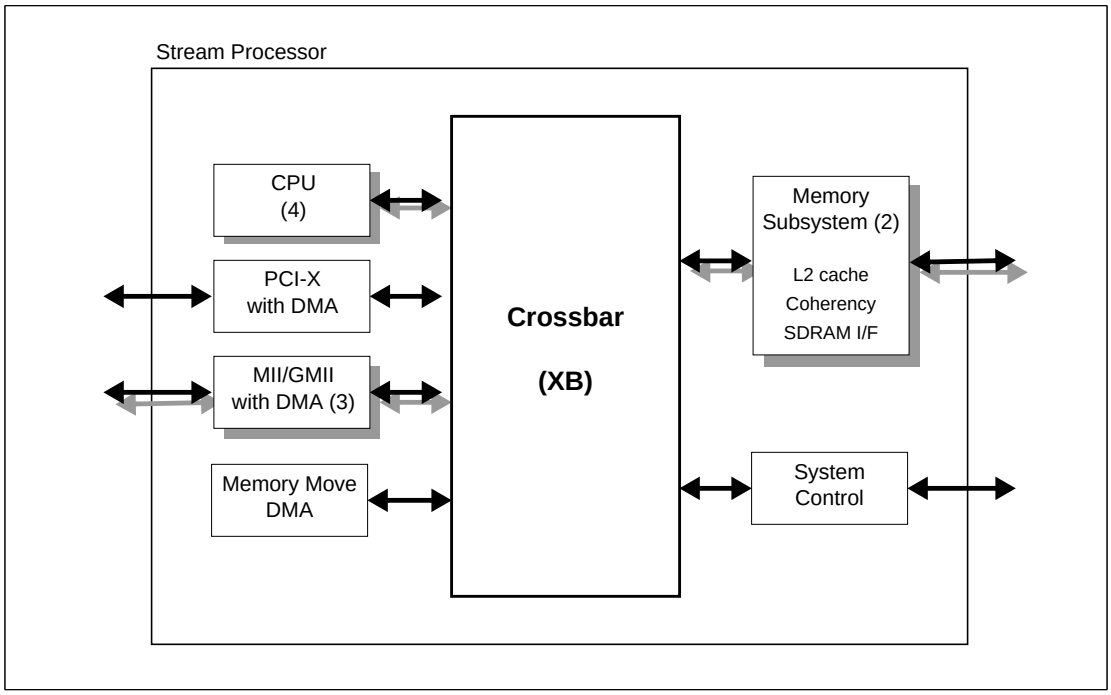

Figure 8: Overview of Crossbar

The crossbar supports the following types of operations:

- Operates at the SP-1's system clock speed (1/2 the CPU clock speed).
- High bandwidth interfaces transfer 64 bits per cycle, full duplex.
- Distributed queuing to reduce head-of-line blocking.
- Word and sub-word read and write transfers (1, 2, 3 or 4-byte).
- Line read and write transfers (64-byte).
- Coherency signalling.
- Error signalling.
- Interrupt messages for processor-to-processor and device-to-processor signalling.

6.2. Crossbar Architecture

The organization of the crossbar is shown in Figure 9. Each crossbar sub-block includes muxes, arbitration logic and queues. The details of these structures are discussed in Section 6.4.

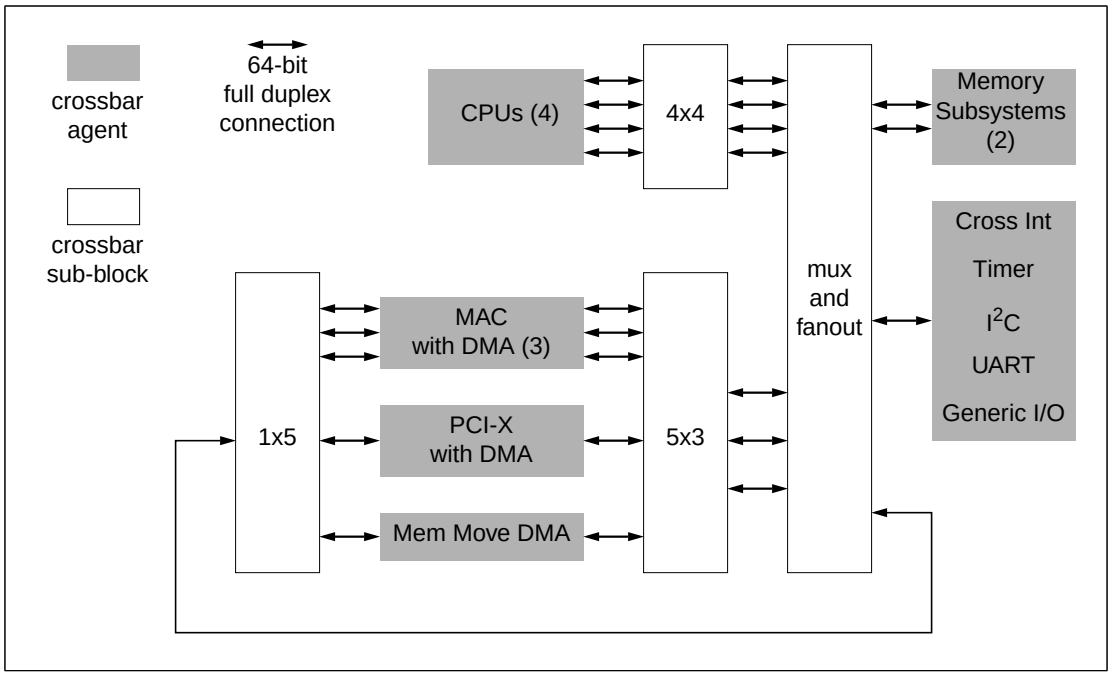

Figure 9: Crossbar Architecture

Table 12 lists the agents that are attached to the crossbar.

Table 12: Crossbar Agents

Agent	Description
p0-3	(4) Processors with L1 caches.
d0-2	(3) 10/100/1000 Ethernet MAC and DMA engine.
d3	(1) PCI-X Master/Slave Bridge and DMA engine.
d4	(1) Memory to memory DMA engine.
d5	(1) Cross-interrupt reflector.
	(1) Timers.
	(1) I^2C interface.
	(1) UART.
	(1) Generic Input/Output (GIO) configuration registers.
d6	(1) Generic Input/Output (GIO) interface.
d7	(1) PCI-X external bus target.
m0-1	(2) Memory Subsystem (MS) with coherency engine, L2 cache and SDRAM controllers.

6.3. Crossbar Messages

The crossbar provides a point-to-point messaging protocol. An *agent* connected to the crossbar may operate as an *initiator*, as a *target*, or both. An initiator sends a message to a specific target, which in turn may act as an initiator by sending one or more new messages, as shown in Figure 10.

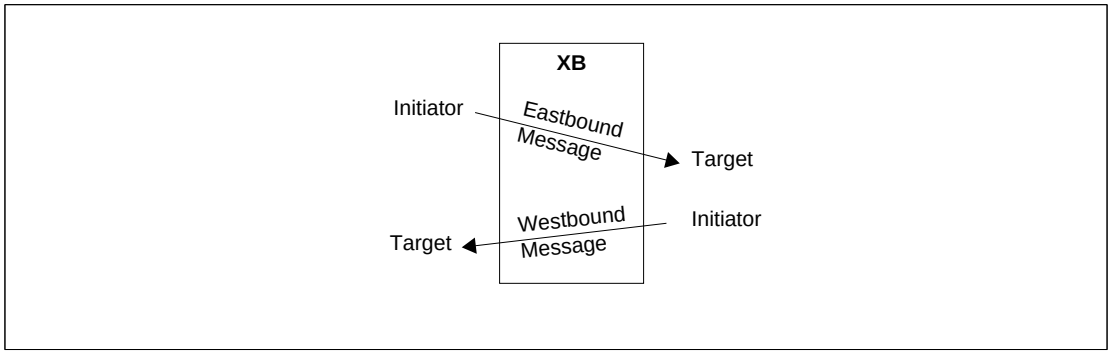

Figure 10: Crossbar Messages

Tables 13 through 16 summarize the crossbar messages types. A complete transaction requires one or more crossbar messages. The originator of the transaction sends the first message of a transaction, typically a *request* message. The target sends a *request reply* to the originator to complete the transaction. The memory subsystem's coherency engine may send one or more *inquiries* to CPUs before completing before a transaction. In this case the CPUs send an *inquiry reply* back to the coherency engine.

Table 13: Eastbound Request Messages

Type	Encoding	Source/Destination	Possible Replies	Message Length	Meaning
CPU Initiated Line Reads					
RL	00 0000	CPU / MS	DLS, DLE	1	Read line
		CPU / GIO	DLE		
RLE	01 0000	CPU / MS	DLS, DLE	2	Read line with eviction
RLM	00 0001	CPU / MS	DLM	1	Read line with intent to modify
		CPU / GIO	DLM		
		CPU / PCI	DLM		
RLME	01 0001	CPU / MS	DLM	2	Read line with intent to modify and eviction
UM	00 0010	CPU / MS	UMA, DLM	1	Upgrade line to Modified
		CPU / GIO	UMA		
		CPU / PCI	UMA		
VE	00 0011	CPU / MS	none	Beat 2 of RLE or RLME	Eviction address beat

Table 13: Eastbound Request Messages (Continued)

Type	Encoding	Source/Destination	Possible Replies	Message Length	Meaning
CPU Initiated Line Writes (including CPU CACHE instruction)					
WLI	00 0100	CPU / MS, GIO, PCI	none	1	Write line, mark Invalid
WLS	00 0101	CPU / MS, GIO, PCI	none	1	Write line, mark Shared
LI	00 0110	CPU / MS, GIO, PCI	none	1	Invalidate line
DMA and PCI Initiated Line Reads and Writes					
RLN	00 0111	DMA, PCI / MS	DL	1	Read line with no L2 allocation
WLN	10 0111	DMA, PCI / MS	WLA	9	Write line with no L2 allocation
CPU, DMA and PCI Initiated Sub-Line Reads and Writes					
RB	00 1000	CPU, DMA, PCI / MS	DS	1	Read byte
		CPU / d0-6, PCI	DS		
		PCI / d5-6	DS		
RH	00 1001	CPU, DMA, PCI / MS	DS	1	Read half-word
		CPU / d0-6, PCI	DS		
		PCI / d5-6	DS		
RT	00 1010	CPU / MS, PCI	DS	1	Read tri-byte
RW	00 1011	CPU, DMA, PCI / MS	DS	1	Read word
		CPU / d0-6, PCI	DS		
		PCI / d5-6	DS		
WB	01 1000	CPU, DMA, PCI / MS	WSA	2	Write byte
		CPU / d0-6, PCI	WSA		
		PCI / d5-6	WSA		
WH	01 1001	CPU, DMA, PCI / MS	WSA	2	Write half-word
		CPU / d0-6, P CI	WSA		
		PCI / d5-6	WSA		
WT	01 1010	CPU / MS, PCI	WSA	2	Write tri-byte
WW	01 1011	CPU, DMA, PCI / MS	WSA	2	Write word
		CPU / d0-6, PCI	WSA		
		PCI / d5-6	WSA		

Table 14: Westbound ReqReply Messages

Type	Encoding	Source/Destination	Message Length	Meaning
DL	10 0000	MS / DMA, PCI	9	Read line data, no cache
DLS	10 0001	MS / CPU	9	Read line data, install S
DLE	10 0010	MS, GIO / CPU	9	Read line data, install E
DLM	10 0011	MS / CPU	9	Read line data, install M
DS	01 0000	MS / CPU	2	Read sub-line data
UMA	00 0000	MS, GIO / CPU	1	Upgrade ack
WLA	00 0001	MS, GIO / DMA, PCI	1	Line write ack
WSA	00 0010	MS / CPU, DMA	1	Sub-line write ack
		d0-6 / CPU		
		d5-6 / PCI		
BE	00 0011		1	Bus error (bad address or size)

Table 15: Westbound Inquiry Messages

Type	Encoding	Source/ Destination	Possible Replies	Message Length	CPU Actions
II	11 0000	MS / CPU	IA	1	Invalidate line; no eviction.
IIE	11 0001	MS / CPU	IA IEA	1	Invalidate line; If M evict
IDE	11 0010	MS / CPU	IA IEA	1	If E downgrade to S; If M downgrade to I and evict
IRE	11 0011	MS / CPU	IRA	1	If due to WLS downgrade to S, else downgrade to I; purge from evict buffer
IN	11 0100	d0-5 / CPU	INA	1	Request interrupt

Table 16: Eastbound InqReply Messages

Type	Encoding	Source/ Destination	Message Length	Meaning
IA	110 000	CPU / MS	1	Inquiry ack
IEA	111 000	CPU / MS	9	Inquiry w/eviction ack
IRA	111 001	CPU / MS	9	Inquiry w/replacement eviction ack
INA	110 001	CPU / d0-5	1	Request interrupt ack

6.3.1. Single Beat Message Format

Message Header

6.3.2. RLE, RLME Request Message Format

Message Header
Eviction Address

6.3.3. DS, WB, WH, WT, WW Message Format

Data must be presented in its natural byte lane.

Message Header
Data

6.3.4. DL*, WLN, IEA, IRA Message Format

	Message Header
Header address[a]	Data Beat 0
Header address + 0x8	Data Beat 1
Header address + 0x10	Data Beat 2
Header address + 0x18	Data Beat 3
Header address + 0x20	Data Beat 4
Header address + 0x28	Data Beat 5
Header address + 0x30	Data Beat 6
Header address + 0x38	Data Beat 7

a. For these messages, the low order 6 bits of the Header address must be zero.

6.3.5. Message Header, Eviction Address Beat Format

6 63-58	2 57-56	36 55-20	10 19-10	4 9-6	6 5-0
unused	Way	Addr	WestAgent	EastAgent	Type

Type	Message Type; see Section 6.3 (includes class and size) Eastbound: 00 xxxx 1 beat Request message 01 xxxx 2 beat Request message 10 xxxx 9 beat Request message 110 xxx 1 beat InqReply message 111 xxx 9 beat InqReply message Westbound: 00 xxxx 1 beat ReqReply message 01 xxxx 2 beat ReqReply message 10 xxxx 9 beat ReqReply message 11 xxxx 1 beat Inquiry message
EastAgent	East Side Message Agent 0000 Memory Subsystem 0 controller 0001 Memory Subsystem 0 SDRAM 0010 Memory Subsystem 1 controller 0011 Memory Subsystem 1 SDRAM 01xx (reserved) 1000 device 0 1001 device 1 1010 device 2 1011 device 3 1100 device 4 1101 device 5 1110 device 6 1111 device 7
WestAgent	West Side Message Agent 00xxxxxxx CPUI (I cache) 01xxxxxxx CPUD (D cache) 10xxxxxxx CPUE (EJTAG) 11xxxxxxx DMA data path WAgent[7:4] = CPU/DMA number WAgent[3:0] = context number
Addr	Memory address for Cmd. For line operations (DL*, WLN, IEA and IRA), the low order six bits of the address must be zero. For interrupt requests (IN) that are sourced by the interrupt reflector, bit 24 of the message header is 1, and bits 23:20 of the header indicate which bit to set in the west agent's CP0 ExtendedIP1 register. For interrupt requests from any other source, bits 24:20 of the message header are 0. See Section 4.2.3.
Way	Encoded way of L1 tag operation for use in duplicate L1 tag update. Required with CPU RL, RLE, RLM, RLME, UM, and LI requests.

6.3.6. Data Beat Format

63-56	55-48	47-40	39-32	31-24	23-16	15-8	7-0
byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7

6.3.7. Error Detection and Reporting

Initiators are responsible for detecting address errors that arise if a 36-bit system address does not decode to an XB target. The method of signaling this error within the initiator is initiator-defined.

Targets may also detect address-related errors as a result of fine-grained decoding of the 36-bit system address performed by the target. These errors are reported to the original initiator with an error message. The target of such a message (i.e. the initiator that caused the error) processes the error using a target-specific mechanism.

6.4. Crossbar Operation

6.4.1. Clocking

The XB operates in the system clock domain. Agents that operate at other speeds are responsible for frequency matching.

6.4.2. Initiator-Target Relationships

The internal structure of the crossbar is optimized based on the characteristics of the attached agents and the types of messages that must be supported between agent pairs. Table 17 summarizes the basic message types possible for all valid initiator-target pairs. In reference to Figure 9, an *eastbound* message is one that is sourced from the right side of an initiator. A *westbound* message is one that is sourced from the left side of an initiator.

Table 17: Initiator-Target Relationships

Initiator	Target	Direction	Message Sizes (bytes)
p0-3, d0-4, d7	m0-1	eastbound	8, 16, 72
p0-3	d0-5	eastbound	8, 16
p0-3	d6-7	eastbound	8, 16, 72
d7	d5-6	eastbound	8, 16, 72
m0-1	p0-3, d0-4, d7	westbound	8, 16, 72
d0-5	p0-3	westbound	8, 16
d6-7	p0-3	westbound	8, 16, 72
d5-6	d7	westbound	8, 16, 72

6.4.3. Crossbar Transfer Networks

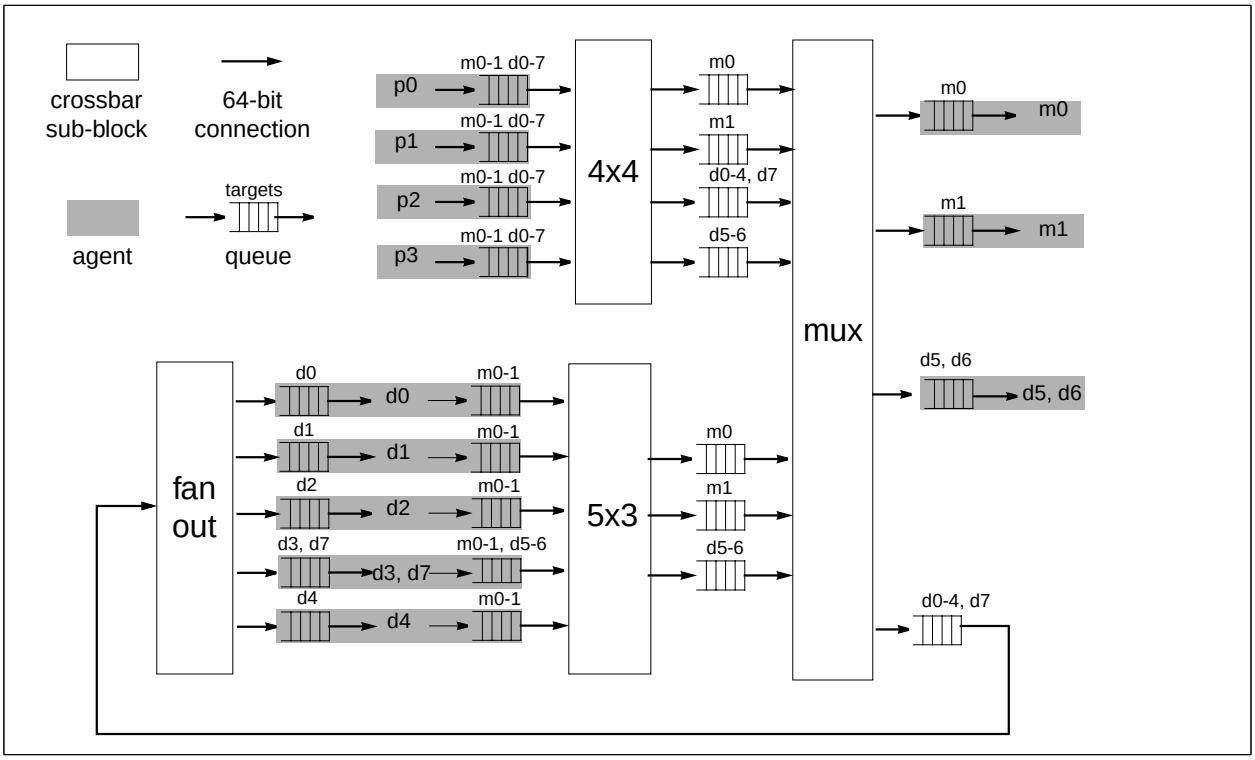

Figure 11: Eastbound Crossbar Network

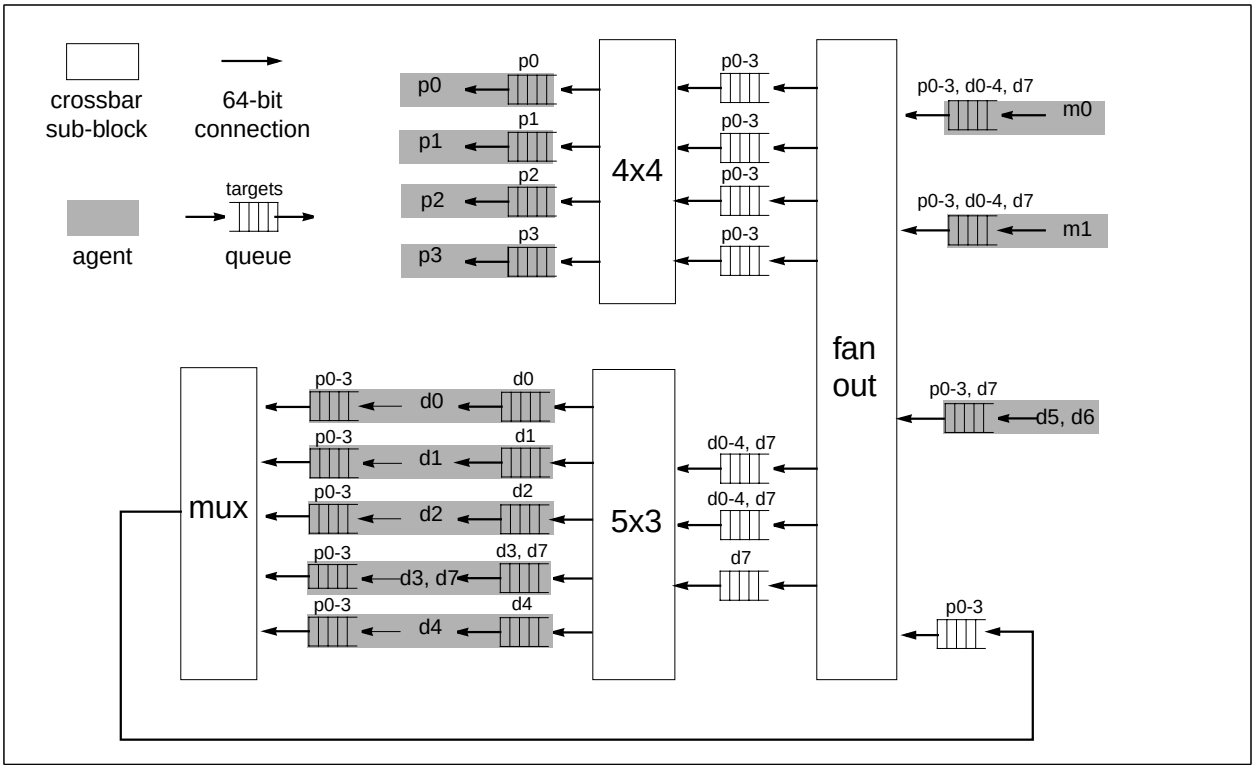

Figure 12: Westbound Crossbar Network

6.5. Crossbar Internal SP-1 Interfaces

6.5.1. Initiator and Target Message Interfaces

Initiators connect to the XB through separate initiator and target message interfaces.

Table 18: Initiator Message Interface

Signal	Direction	Description
ImsgVal	from initiator	Asserted if message beat present on ImsgData.
ImsgStart	from initiator	Asserted first message beat is present on ImsgData. Not valid if ImsgVal is not asserted.
ImsgRdy	to initiator	Asserted if XB accepts the current message beat.
ImsgData[63:0]	from initiator	Message beat contents.

Table 19: Target Message Interface

Signal	Direction	Description
TmsgVal	to target	Asserted if message beat present on TmsgData.
TmsgStart	to target	Asserted if first message beat present on TmsgData. Not valid if TmsgVal is not asserted.
TmsgRdy	from target	Asserted if target accepts the current message beat.
TmsgData[63:0]	to target	Message beat contents.

6.5.2. Initiator and Target Protocols

The protocol is described in terms of an initiator interface. However the same statements hold true for a target interface, with the agent and crossbar relationships exchanged. The general characteristics of a message transfer are illustrated in Figure 13.

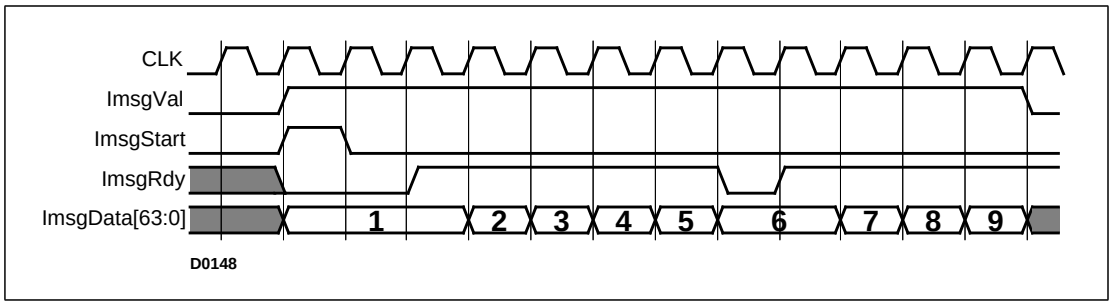

Figure 13: Message Transfer Protocol

Before starting to transmit a message the initiator should be able to source message beats without any delay of consecutive data beats. The initiator must be able to source all message beats without any dependency on the completion of other currently pending internal or external activity.

In cycle 1 the initiator attempts to start transmitting a message by asserting ImsgStart and ImsgVal and sourcing the first message beat on ImsgData. The first message beat contains information used by the XB's internal arbiter.

The XB indicates readiness to accept a beat in any cycle by asserting ImsgRdy. If the initiator samples ImsgRdy asserted at the end of any cycle during which the initiator is driving ImsgVal, the initiator should present a new message beat, if any, in the next cycle. This process continues until all message beats are transferred. After the last beat of a message are transferred, a new message may start in the next cycle.

The XB employs pipelined arbitration, but does not expose the details of arbitration to the initiator. The XB may assert ImsgRdy when ImsgVal is not asserted. The protocol prevents the need for any initiator output signals to depend on any of its input signals. To accomplish this, the XB includes a limited amount of input buffering for each initiator interface.

7.1. LX4580 CPU Overview

SP-1 includes four LX4580 CPUs, each of which provides four independent hardware contexts. 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 KByte 4-way set associative instruction cache.
- 16 KByte 4-way set associative data cache.
- Data coherency using MESI algorithm.
- Performance counters.

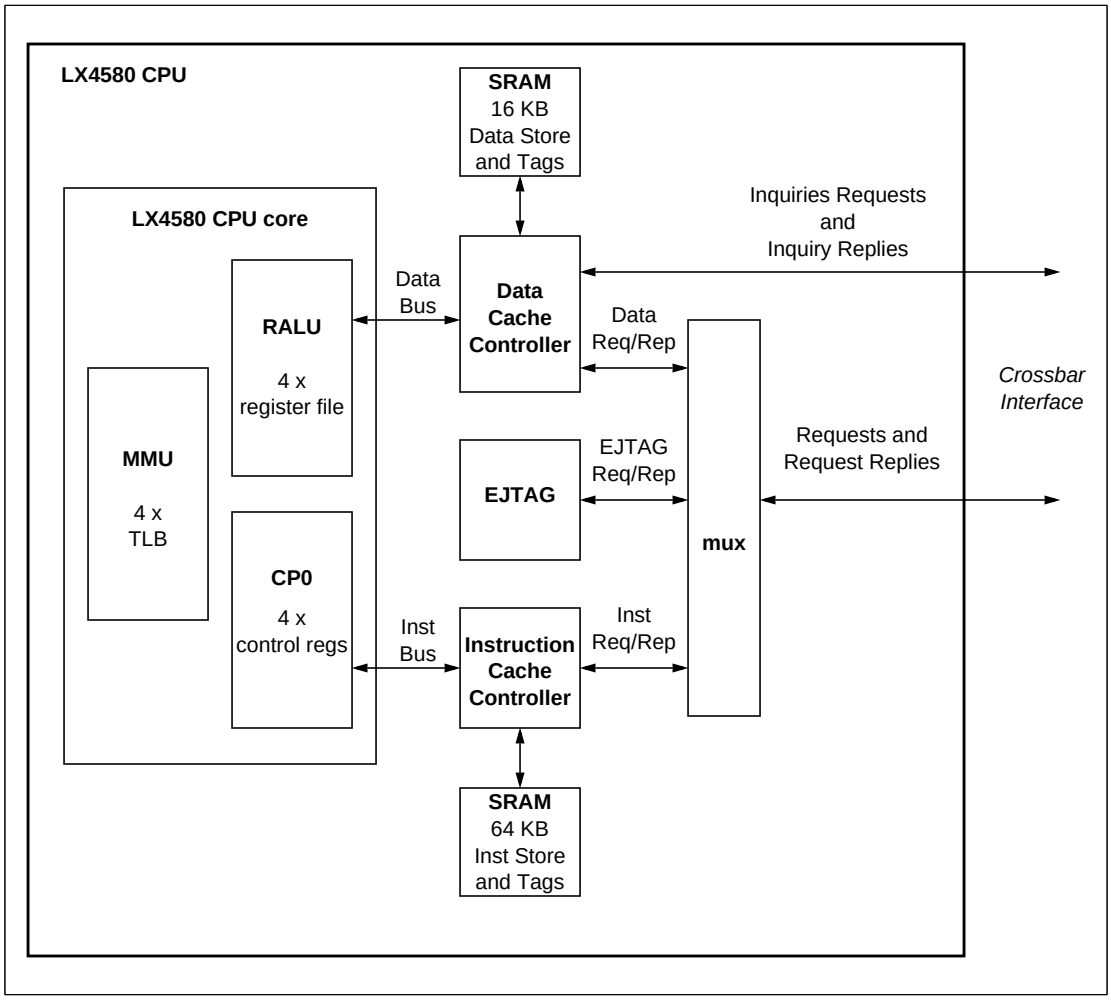

Figure 14: LX4580 CPU and Crossbar Interface

7.2. 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.

7.3. Instruction Cache

The LX4580 CPU includes a 64 KByte 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.

Table 20 describes the actions that are taken by the CPU for the different possible outcomes of a reference to the instruction cache. This table is from the perspective of the CPU only, and does not indicate additional chip-level coherency interactions that can take place within SP-1. A complete description of these interactions is given in Chapter 9.

The Tag column indicates the possible outcomes of the instruction cache's tag compare logic. For a tag compare hit, the Current State indicates the state of the cache line that is referenced. For a tag compare miss, the Current State indicates the state that the cache line that is selected for replacement. The New State is the state of the cache line after the transaction is completed. The Crossbar Request column indicates the type of message that the CPU initiates over the crossbar to complete the transaction.

Table 20: Instruction Cache Transactions

Command	Tag	Current State	New State	Crossbar Request
Fetch Cached	x	I	V	RL
Fetch Cached	Hit	V	V	none
Fetch Cached	Miss	V	V	RL
Fetch Uncached	x	x	(unchanged)	RW

7.4. Data Cache

The LX4580 CPU includes a 16 KByte 4-way set associative data cache that operates at the processor clock speed. Data in the cache is organized in 64-byte lines, and is held in one of four states corresponding the MESI (Modified, Exclusive, Shared, Invalid) cache coherency algorithm.

Table 21 describes the actions that are taken within the CPU for the different possible outcomes of a reference to the data cache. This table is from the perspective of the CPU only, and does not indicate additional chip-level coherency interactions that can take place within SP-1. A complete description of these interactions is given in Chapter 9.

The Tag column indicates the possible outcomes of the data cache's tag compare logic. For a tag compare hit, the Current State indicates the state of the cache line that is referenced. For a tag compare miss, the Current State indicates the state that the cache line that is selected for replacement. The New State is the state of the cache line after the transaction is completed. When data is returned from the crossbar, the request reply

indicates whether the data should be installed in the Shared or Exclusive states. The Crossbar Request column indicates the type of message that the CPU initiates over the crossbar to complete the transaction.

Table 21: Data Cache Transactions

Command	Tag	Current State	New State	Crossbar Request
Rd Cached	x	I	S or E	RL
Rd Cached	Hit	S	S	none
Rd Cached	Miss	S	S or E	RL
Rd Cached	Hit	E	E	none
Rd Cached	Miss	E	S or E	RL
Rd Cached	Hit	M	M	none
Rd Cached	Miss	M	S or E	RLE
Rd Uncached	x	I	I	RB, RH, RT or RW
Rd Uncached	Hit	S E M	I	RB, RH, RT or RW[a]
Rd Uncached	Miss	S E M	(unchanged)	RB, RH, RT or RW
Wr Cached	x	I	M	RLM
Wr Cached	Hit	S	M	UM
Wr Cached	Miss	S	M	RLM
Wr Cached	Hit	E	M	none
Wr Cached	Miss	E	M	RLM
Wr Cached	Hit	M	M	none
Wr Cached	Miss	M	M	RLME
Wr Uncached	x	I	I	WB, WH, WT or WW
Wr Uncached	Hit	S E M	I	WB, WH, WT or WW[b]
Wr Uncached	Miss	S E M	(unchanged)	WB, WH, WT or WW

a. Before reading data from main memory, the SP-1 coherency engine performs an inquiry to the CPU cache to invalidate or evict the data from the cache.
b. Before writing data to main memory, the SP-1 coherency engine performs an inquiry to the CPU cache to invalidate or evict the data from the cache.

7.5. 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 a 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.

Table 22: Cache Line Replacement Algorithm

Tag state	MRF	Way selected
Way 0 invalid	xx	Way 0
Way 0 valid, Way 1 invalid	xx	Way 1
Way 0,1 valid, Way 2 invalid	xx	Way 2
Way 0,1,2 valid Way 3 invalid	xx	Way 3
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

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.

7.6. CPU Error Handling

To be supplied.

8.1. CPU Crossbar Interface Overview

Each of SP-1's LX4580 CPU is connected to the crossbar with a dedicated Crossbar Interface (XBI). The interface includes the following features:

- Split transaction interface between the CPU and crossbar.
- Processes Interrupt messages (IN) received from the crossbar.
- Processes Error messages (BE) received from the crossbar.
- System address space configuration registers.
- Dedicated transaction FIFOs.
- Matches CPU frequency (500 MHz) with crossbar frequency (250 MHz).
- CBUS interface to be used for requests and request replies.
- IBUS interface to be used for inquiries and inquiry replies.

The organization of the CPU Crossbar Interface is shown in Figure 15.

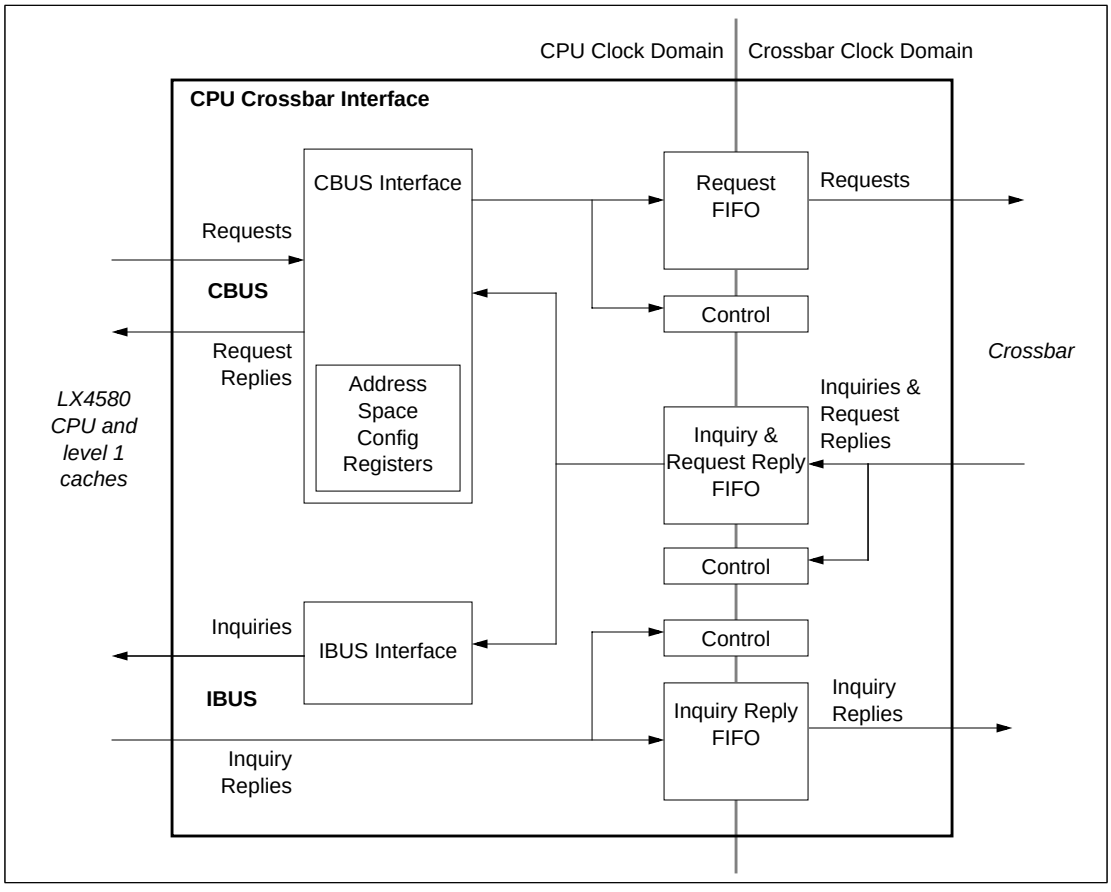

Figure 15: CPU Crossbar Interface Architecture

This interface serves as the filter of incoming and outgoing messages. Westbound inquiries and request replies are examined and redirected to the appropriate internal CPU bus (IBUS or CBUS).

Some messages, such as the interrupt message or error messages, are consumed by the interface and not retransmitted on the IBUS nor CBUS. Instead, those messages are converted to an interrupt or exception signal for the CPU.

Eastbound requests and inquiry replies are examined to determine the destination device for the message. The destination could also be the configuration registers within the crossbar interface.

8.2. CBUS Interface

The CBUS is a simple signalling layer between the CPU and EJTAG, instruction cache and data cache. The CBUS handles crossbar requests and request replies. The protocol supports a single cycle request and a single or multi-beat request replies.

8.3. IBUS Interface

The IBUS is a crossbar style interface between the CPU and data cache. The IBUS handles crossbar inquiries and inquiry replies. This bus is separate from the CBUS to ensure requests and request replies do not block inquiries and inquiry replies.

8.4. Request FIFO

The Request FIFO receives only single and dual beat messages. The Request FIFO contains four (4) beats. This allows 2 dual beat messages or 4 single beat messages to be buffered inside the XBI. There is additional buffering locally in each cache.

8.5. Inquiry & Request Reply FIFO

The Inquiry & Request Reply FIFO receives two types of traffic from the crossbar. Request replies from crossbar devices can vary in size from a single beat to a full nine beat cache line. Entire line read replies are loaded into the FIFO before the reply is sent to the cache. The Inquiry & Request Reply FIFO contains ten (10) beats of buffering. This allows an entire line and two beats of other transaction(s) to be buffered in the XBI.

8.6. Inquiry Reply FIFO

The Inquiry Reply FIFO accepts single beat or nine beat messages. The data is taken from the data cache's eviction buffers. To help free the line immediately, the XBI contains a nine (9) beat Inquiry Reply FIFO.

8.7. System Address Space Configuration Registers

CBUS requests may target Address Space Configuration Registers within the crossbar interface to configure the system address space. These registers identify the valid physical address spaces within the SP-1 and determine where in the physical memory map different crossbar devices are placed. For each configurable space, there is a mask and base value. Bits from a request address are ANDed with the mask and logically compared to the base value. If equal, the indicated address space is selected. The exception to this is the SDRAM address space configuration, which does not have a base value. The base for SDRAM is always zero.

8.7.1. AS_DRAMMask Register

Name: AS_DRAMMask.
Size: 32 bits.
Address: AS_Base + 0x0000.
SW Init: Must be set during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31-12	11-0
0	DRAMMask

Field	Bits	Description	R/W	Reset
0	31-12	Reserved and must be 0	R	0
DRAMMask	11-0	Mask for address bits 35-24. These bits are used to declare the actual size of addressable SDRAM. The SDRAM space begins at physical address 0x0_0000_0000. The initial state specifies a 32MB SDRAM region.	R/W	0xfff

8.7.2. AS_PCIABase Register

Name: AS_PCIABase.
Size: 32 bits.
Address: AS_Base + 0x0010.
SW Init: Must be set this during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31-24	23-0
0	PCIABase

Field	Bits	Description	R/W	Reset
0	31-24	Reserved and must be 0	R	0
PCIABase	23-0	Bits 35-12 of the base of physical address space serviced by components attached to the Stream Processor PCI-X bridge region A.	R/W	0

8.7.3. AS_PCIAMask Register

Name: AS_PCIAMask.
Size: 32 bits.
Address: AS_Base + 0x0014.
SW Init: Must be set this during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31	30-24	23-0
PCIAEn	0	PCIAMask

Field	Bits	Description	R/W	Reset
PCIAEn	31	When set, declares that PCI region A is enabled and accesses to the region are steered to the PCI device. When clear this region is disabled.	R/W	0
0	30-24	Reserved and must be 0	R	0
PCIAMask	23-0	Mask for address bits 35-24 of the PCI-X address space. This mask will specify the size of the window for PCI-X bridge region A.	R/W	0

8.7.4. AS_PCIBBase Register

Name: AS_PCIBBase.
Size: 32 bits.
Address: AS_Base + 0x0018.
SW Init: Must be set this during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31-24	23-0
0	PCIBBase

Field	Bits	Description	R/W	Reset
0	31-24	Reserved and must be 0	R	0
PCIBBase	23-0	Bits 35-12 of the base of physical address space serviced by components attached to the Stream Processor PCI-X bridge region B.	R/W	0

8.7.5. AS_PCIBMask Register

Name: AS_PCIBMask.
Size: 32 bits.
Address: AS_Base + 0x001C.
SW Init: Must be set this during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31	30-24	23-0
PCIBEn	0	PCIBMask

Field	Bits	Description	R/W	Reset
PCIBEn	31	When set, declares that PCI region B is enabled and accesses to the region are steered to the PCI device. When clear this region is disabled.	R/W	0
0	30-24	Reserved and must be 0	R	0
PCIBMask	23-0	Mask for address bits 35-24 of the PCI-X address space. This mask will specify the size of the window for PCI-X bridge region B.	R/W	0

8.7.6. AS_GIOBase Register

Name: AS_GIOBase.
Size: 32 bits.
Address: AS_Base + 0x0020.
SW Init: Must be set this during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31-24	23-0
0	GIOBase

Field	Bits	Description	R/W	Reset
0	31-24	Reserved and must be 0	R	0
GIOBase	23-0	Bits 35-12 of the base of physical address space serviced by components attached to GIO.	R/W	0

8.7.7. AS_GIOMask Register

Name: AS_GIOMask.
Size: 32 bits.
Address: AS_Base + 0x0024.
SW Init: Must be set this during system configuration.
Restrictions: Must be set to the same value for all CPUs.

31	30-24	23-0
GIOEn	0	GIOMask

Field	Bits	Description	R/W	Reset
GIOEn	31	When set, declares that the GIO region is enabled and accesses to the region are steered to the GIO device. When clear this region is disabled.	R/W	0
0	30-24	Reserved and must be 0	R	0
GIOMask	23-0	Mask for address bits 35-24 of the GIO address space. This mask specifies the size of the window for GIO.	R/W	0

8.7.8. CPUX_IntPend Register

Name: CPUX_IntPend.
Size: 32 bits.
Address: AS_Base + 0x0100 + (Context * 0x08).
SW Init: None.
Restrictions: Write value is used to clear bits.

31-16	15-0
0	CPUX_IntPend

Field	Bits	Description	R/W	Reset
0	31-16	Reserved and must be 0	R	0
CPUX_IntPend	15-0	Cpu cross-interrupt pending bits for the given context. These bits are set when a cpu cross-interrupt message is received from the interrupt reflector. Individual bits are cleared when written with '1'. Bits are untouched when written with '0'.	R/W	0

8.7.9. CPUX_IntMask Register

Name: CPUX_IntMask.
Size: 32 bits.
Address: AS_Base + 0x0104 + (Context * 0x08).
SW Init: None.
Restrictions: None.

31-16	15-0
0	CPUX_IntMask

Field	Bits	Description	R/W	Reset
0	31-16	Reserved and must be 0	R	0
CPUX_IntMask	15-0	Cpu cross-interrupt mask bits for the given context. They are logically ANDed with the CPUX_IntPend bits to determine if the context has received an interrupt.	R/W	0

8.7.10. DEV_IntPend Register

Name: DEV_IntPend.
Size: 32 bits.
Address: AS_Base + 0x0200 + (Context * 0x08).
SW Init: None.
Restrictions: Write value is used to clear bits.

31-8	7-0
0	DEV_IntPend

Field	Bits	Description	R/W	Reset
0	31-8	Reserved and must be 0	R	0
DEV_IntPend	7-0	Device interrupt pending bits for the given context. These bits are set when a device interrupt message is received. Individual bits are cleared when written with '1'. Bits are untouched when written with '0'.	R/W	0

8.7.11. DEV_IntMask Register

Name: DEV_IntMask.
Size: 32 bits.
Address: AS_Base + 0x0204 + (Context * 0x08).
SW Init: None.
Restrictions: None.

31-8	7-0
0	DEV_IntMask

Field	Bits	Description	R/W	Reset
0	31-8	Reserved and must be 0	R	0
DEV_IntMask	7-0	Device interrupt mask bits for the given context. They are logically ANDed with the DEV_IntPend bits to determine if the context has received an interrupt.	R/W	0

8.7.12. EXT_IntPend Register

Name: EXT_IntPend.
Size: 32 bits.
Address: AS_Base + 0x0300 + (Context * 0x08).
SW Init: None.
Restrictions: None.

31-4	3-0
0	EXT_IntPend

Field	Bits	Description	R/W	Reset
0	31-4	Reserved and must be 0	R	0
EXT_IntPend	3-0	External interrupt pending bits for the given context. These bits are set when an external interrupt is received. The interrupt is cleared by accessing the interrupt reflector.	R	0

8.7.13. EXT_IntMask Register

Name: EXT_IntMask.
Size: 32 bits.
Address: AS_Base + 0x0304 + (Context * 0x08).
SW Init: None.
Restrictions: None.

31-4	3-0
0	EXT_IntMask

Field	Bits	Description	R/W	Reset
0	31-4	Reserved and must be 0	R	0
EXT_IntMask	3-0	External interrupt mask bits for the given context. These bits are logically ANDed with the EXT_IntPend bits to determine the interrupt.	R/W	0

THE REMAINDER OF THIS CHAPTER IS FOR INTERNAL LEXRA USE.

8.8. CBUS Interface

8.8.1. CBUS Request Interface

Table 23: CBUS Request Internal Interface

Signal Name	Direction	Description
CBUS_YREQO	output	CBus Request
CBUS_YADDRO[35:0]	output	Physical Address for Request
CBUS_YDATAO [31:0]	output	Data for Request
CBUS_YCMDO [3:0]	output	CBus Request Command
CBUS_YSZO [1:0]	output	Size of Data for Request
CBUS_YSRCO [2:0]	output	Source of Request
CBUS_YDWAYO[1:0]	output	data cache way L1 duplicate tag update
CBUS_YLTIDO [1:0]	output	Thread ID of Request
CBUS_YBUSYI	input	Crossbar Busy

8.8.2. CBUS Command Encoding

Table 24: CBUS 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

8.8.3. RLE & RLME Eviction Address

The Eviction address for the RLE and RLME requests is transferred through the CBUS_YDATAO line to the CPU Crossbar Interface. Since physical addresses are 36-bits and the CBUS_YDATAO line is 32-bits wide, the entire address cannot be transferred. The assumption is made that the Eviction Address is line-aligned. Therefore, only 30-bits are needed to transfer the address. The format of the address is as follows:

31-30	29-6	5-0
(0)	Address Tag	Address Index

8.8.4. CBUS Request Reply Interface

Table 25: CBUS Request Reply Internal Interface

Signal Name	Direction	Description
CBUS_YDESTI[2:0]	input	Destination for Reply Data
CBUS_YLSTEI[2:0]	input	Line State and Transaction Reply Type
CBUS_YRDLTIDI[1:0]	input	Thread ID for Reply
CBUS_YDATAI[63:0]	input	Reply Data
CBUS_YDBUSYO	output	data cache Busy

8.8.5. CBUS Request Reply Destination Encoding

Table 26: CBUS 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 27: CBUS Line State and Transaction Encoding

CBUS_YLSTEI[2:0]	Description
000	Uncached Sub-Line Read
001	Cached Shared Line Read
010	Cached Exclusive Line Read
011	Cached Modified Line Read
100	Sub-Line Write Ack
101	Reserved
110	Reserved
111	Upgrade Line to Modified State

8.9. IBUS Interface

8.9.1. Inquiry Interface

Table 28: IBUS Request Internal Interface

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

8.9.2. IBUS Command Encoding

Table 29: IBUS Commands

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

8.9.3. Inquiry Reply Interface

Table 30: IBUS Reply Internal Interface

Signal Name	Direction	Description
IBUS_RDYO	output	Inquiry Reply Ready
IBUS_STARTO	output	Inquiry Reply Header Data Valid
IBUS_HDRDATAO[64:0]	output	Inquiry Reply Header & Data Bus
IBUS_XBUSYI	input	Crossbar Interface Busy

8.9.4. IBUS Header Encoding

The header of an IBUS reply transfer is equivalent to the crossbar header format. The only IBUS replies supported are IA, IEA, and IRA. The data cache will leave the header untouched from the inquiry except for the message type field. Refer to Section 6.3.5 for a detailed description of the crossbar header format.

8.10. Interrupt Interface

Table 31: Interrupt Interface

Signal Name	Direction	Description
EXT_INTREQ[3:0]	input	Active High External Interrupt
GLOBAL_INT	input	Active High Hardware Error Interrupt

9.1. Memory Subsystem Overview

The *Memory Subsystem (MS)* is used by the Stream Processor to manage memory requests. It provides a L2 cache, and maintains coherency between main memory and all caches in the Stream Processor. This coherency support allows use of software applications that do not have knowledge of the hardware architecture.

The major components of the MS are the Coherency Engine (Che), cache tags, and L2 cache. There are two Memory Subsystems within SP-1, which are interleaved on cache line boundaries. There are also two memory controllers (MCs) per SP-1. Depending on required memory depth and bandwidth, one or two MC's may be enabled. A Memory Interconnect network (MI) is provided to connect the MS's to the MC's.

The total available L2 cache size is 64KB. Up to 8 GB external memory and 51 Gb/s memory bandwidth are supported by the Stream Processor.

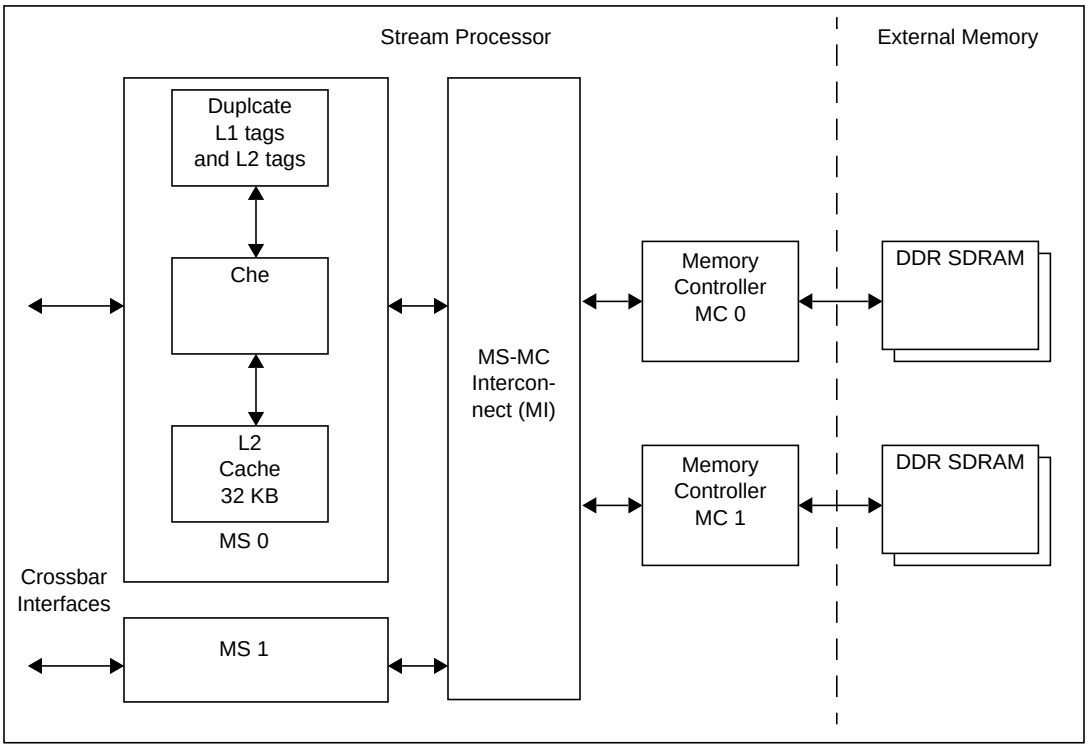

Figure 16: Memory Subsystem Blocks

9.2. Memory Subsystem Requirements

Each of the two Memory Subsystems provides the following features:

- Memory Controller Interface
  - Each MC services 1 or 2 MSs through the MI.
  - ECC protected memory.
  - Interface to PC-2100 (133 MHz) DDR SDRAM DIMMs.
  - Interface to 200 MHz DDR SDRAM components.
  - 32 MB minimum SDRAM/controller using 3 128Mb components.
  - 4 GB maximum SDRAM/controller using 2 2GB DIMMs.
  - 2.5v SSTL_2 compatible I/O.
  - up to 26 Gb/s SDRAM bandwidth.
- L2 Cache
  - 32KB on-chip shared L2 cache/subsystem, 4-way set associative, parity protected.
  - 32 Gb/s L2 cache bandwidth.
  - Parity protected on-chip cache tags.
- Che
  - Hardware-enforced coherency between main memory and all caches.
  - 32 Gb/s crossbar interface at 250 MHz.
  - Handles 50M transactions/s.
  - 64 byte line size.
  - L2 Cache management and control: Write-Through Protocol, Pseudo-LRU replacement.
  - Supports CPU L1 caches using MESI protocol.
  - Automatic initialization of cache tags.
  - BIST for on-chip memory arrays including tags and L2 cache.
  - Error management for coherency and data errors.
  - Performance monitor for message types and miss rates.

9.2.1. Transaction Rate and Bandwidth

The crossbar interface to each memory subsystem provides 32 Gb/s of bandwidth. Assuming that each transaction uses 10 crossbar beats (A one beat Request message and a nine beat ReqReply message), the maximum transaction rate is determined as follows:

```
32 Gb/s = transaction rate * 10 beats/transaction * 64 bits/beat

transaction rate = 50M transactions/s
```

9.3. Supported Memory Configurations

One or two memory controllers are supported per Stream Processor. The available memory bandwidth and memory depth vary with the number of memory controllers. When using DIMM memory, the memory clock rate/date rate is limited to 133 MHz / 266 MHz. Configurations using DIMM memory are shown in Table 32.

Table 32: Memory Configurations w/DIMMs

Number of MC	Memory Bandwidth	Total Memory Size / DIMM Quantity				
		128 MB DIMM	256 MB DIMM	512 MB DIMM	1 GB DIMM	2 GB DIMM
1	17 Gb/s	128 MB / 1	256 MB / 1	512 MB / 1	1 GB / 1	2 GB / 1
2	34 Gb/s	256 MB / 2 512 MB / 4	512 MB / 2 1 GB / 4	1 GB / 2 2 GB / 4	2 GB / 2 4 GB / 4	4 GB / 2 8 GB / 4

When using component memory, the memory clock rate/date rate can reach 200 MHz / 400 MHz. These rates require careful board layout and routing, and may not be achievable with the larger component quantities. Configurations using component memory are shown in Tables 33, 34 and 35.

Table 33: Memory Configurations w/ 128 Mb Components

Number of MC	Memory Bandwidth	Total Memory Size / Component Quantity		
		128 Mb 16 Mb x 8	128 Mb 8 Mb x 16	128 Mb 4 Mb x 32
1	26 Gb/s	128 MB / 9	64 MB / 5	32 MB / 3
2	51 Gb/s	256 MB / 18	128 MB / 10	64 MB / 6

Table 34: Memory Configurations w/ 256 Mb Components

Number of MC	Memory Bandwidth	Total Memory Size / Component Quantity		
		256 Mb 32 Mb x 8	256 Mb 16 Mb x 16	256 Mb 8 Mb x 32
1	26 Gb/s	256 MB / 9	128 MB / 5	64 MB / 3
2	51 Gb/s	512 MB / 18	256 MB / 10	128 MB / 6

Table 35: Memory Configurations w/ 512 Mb Components

Number of MC	Memory Bandwidth	Total Memory Size / Component Quantity	
		512 Mb 64 Mb x 8	512 Mb 32 Mb x 16
1	26 Gb/s	512 MB / 9	256 MB / 5
2	51 Gb/s	1GB / 18	512 MB / 10

9.4. Messages and Transactions

The Stream Processor utilizes the *coherency engine (Che)* to ensure that the L1 caches, L2 caches, and memory are coherent. Each memory subsystem has it own Che and L2 cache.

Crossbar data are grouped in *messages*, which are 1, 2, or 9 64-bit beats long. The first beat of every message contains a message header. This header identifies the message and specifies what action to perform. Crossbar messages in the Stream Processor directed towards Che are called *eastbound* messages. Messages coming from Che are *westbound* messages.

Crossbar messages are grouped in the following message *classes*:

- *Request*: Eastbound message from a DMA engine or CPU for a memory or device operation.
- *ReqReply*: Westbound message in reply to an eastbound Request. May contain acknowledgement and data.
- *Inquiry*: Westbound message from Che to a CPU. The Inquiry is used to alter L1 cache state, and may initiate an eviction.
- *InqReply*: Eastbound message in reply to a westbound Inquiry. May contain acknowledgement and data.

Eastbound Request and InqReply messages classes have separate crossbar queues that allow the messages to pass each other. This is done because an InqReply may be needed to complete a pending Request. Westbound Inquiry and ReqReply message classes share the same crossbar queue.

Certain Request messages spawn a sequence of other messages. The sequence of messages required to coherently complete a Request message is called a *transaction*. Che is responsible for managing the sequence of messages in a transaction. Che may perform L2 cache and/or memory accesses in order to complete a transaction.

A transaction consists of the following messages:

- One eastbound Request.
- Zero or more westbound Inquiry messages requests with their associated eastbound InqReply messages.
- One westbound ReqReply message in reply to the original Request. (ReqReply messages are not provided for CPU cacheop or L1 eviction requests.)

Before the ReqReply message is sent, all Inquiry/InqReply message pairs required to maintain coherency must complete. The CPU must prioritize Inquiry messages to prevent stalls at the memory subsystem. When all InqReply messages are returned to Che, the following events occur:

- The L2 tags and duplicate L1 tags are updated.
- The ReqReply message is returned.

Che maintains a pipeline of Request messages in various stages of completion. This pipeline is organized so that the transaction rate requirement is met.

9.5. Memory Ordering and Interleave

The Stream Processor two independent interleaved memory subsystems. Therefore, it is possible return ReqReply messages in a different order than the initiating Requests occurred.

To maintain strong memory ordering, an individual CPU thread has only one outstanding read and one outstanding write request at a time. A DMA engine may have multiple requests outstanding, and manages completion at a higher level.

Memory subsystems are interleaved on cache line boundaries. Message address bit A6 determines the relationship between a cache line and a memory subsystem. Messages are directed to the appropriate memory subsystem when they enter the crossbar. Message address bit A6 has a constant value when it reaches the memory subsystem and is ignored.

9.6. L2 Cache

Each memory subsystem has a 32 KB 4-way set associative L2 cache. The cache uses a write-through protocol with cache states as defined in Table 36. The replacement algorithm is pseudo-LRU. The cache line size is 64 bytes. L2 cache tag updates occur when Che sends a ReqReply message, indicating transaction completion.

Table 36: L2 Cache States

State	Abbreviation	Meaning
Invalid	I	not in cache
Shared	S	valid in cache

9.7. Duplicate L1 Tags

Che maintains duplicates of the L1 data cache (DCACHE) tags for each CPU. These tags are required because the L2 cache does not maintain inclusion with the L1 caches. In addition, the duplicate tags act as a filter to prevent directing unnecessary Inquiries to CPUs. Duplicate L1 cache states are defined in Table 37. Che does not track the state of the ICACHE.

The duplicate tags are updated by Che based on the outcome of a transaction. In the cases when a Request message leads to an update of the duplicate L1 tag, the Request message header contains the L1 way selected for update. Duplicate L1 tag updates occur when Che sends a ReqReply message, indicating transaction completion.

In some cases, there may be differences between the L1 tags and the duplicate L1 tags. These cases are:

- The CPU writes to a line that was originally exclusive. The L1 tag becomes modified while the duplicate L1 tag remains exclusive.
- The CPU executes a Cache Index Store Tag instruction. The L1 tag is updated, but the duplicate L1 tag is not.

Unless otherwise noted, references to the L1 cache in this document refer to the L1 DCACHE.

Table 37: Duplicate L1 Cache States

State	Abbreviation	Meaning
Invalid	I	not in cache
Shared	S	unmodified in this cache and possibly others
Exclusive	E	unmodified in this cache
Modified	M	modified in this cache

9.8. Coherency Protocol Overview

Che is responsible for maintaining coherency between the L1 caches, L2 cache, and memory in the Stream Processor. The unit of coherency is a cache line. Che does not maintain inclusion between the L1 and L2 caches. This means that a line in an L1 cache is not required to be in the L2 cache.

Coherency must be maintained when an L1 cache has been updated (modified), but other caches and memory have not (they are "stale"). For example, when CPU C0 holds a modified line, and CPU C1 attempts to read that line, the line is stale in memory. Che issues an Inquiry message to request that C0 evict the line, and returns it to C1.

The L2 cache does not hold modified lines, and so always contains the same information as memory. Modified L1 lines are written-through to memory. The L2 cache is allocated only for data line reads when the requested line is exclusive or shared in some L1 cache.

The L2 cache is intended to hold data shared between CPUs. A line enters the shared state when it is read by more than one CPU. However, a modified line does not enter the shared state. Instead, it is returned only to the requesting CPU.

If a line is marked exclusive in the duplicate L1 tag, it is possible that the same line may be stale in the L2 cache and memory. This occurs because the CPU can modify the line without external notification.

The coherency protocol is defined in the transactions tables (Section 9.17). The tables show what actions and messages result from all combinations of Request message types and tag lookups.

9.9. Cacheability and Coherence

If a request is *cacheable*, the target line may be allocated in the L2 cache. If a request is *coherent*, caches (L1 and L2) and main memory are left in a state such that it is possible to determine where fresh and stale copies of the target line reside. The relationship between requests and these attributes are shown in Table 38.

Table 38: Request Attributes

Request Agent and Type	L2 Cacheable	Coherent
CPU DCACHE line request	yes	yes[a]
CPU ICACHE line request	no	yes
CPU (ICACHE, DCACHE, or EJTAG) or DMA sub-line requests	no	yes
DMA read line requests (no allocation)	no	yes
DMA write line requests (no allocation)	no	yes

a. Certain CPU address regions are not Coherent. Requests for these regions never target the MS.

If a non-cacheable request hits a valid line in a L2 cache, the data in that cache is used to fulfill the request. Non-cacheable requests never lead to allocation of the L2 cache.

If a sub-line or DMA request hits a valid line in a L1 cache, that line is invalidated. If the line is modified, it is evicted.

Certain cacheable requests do not lead to allocation of the L2 cache if there is no performance advantage in doing so.

9.10. Inquiry Messages

Inquiry messages are sent to L1 data caches as a result of Request messages that require L1 data or changes in L1 state. Che determines that an L1 data cache Inquiry is required from the Request type and duplicate L1 tags. L1 inquiries should never be received by a CPU for a line that is not valid. Inquiries always lead to a state change in the target L1 cache line. If the line is modified, it is evicted from the L1 cache.

Three types of Inquiries are initiated by Che. When an eviction results, the evicted line moves from the CPU to Che as part of an InqReply message.

1. Unsolicited Inquiry for a valid line (II, IIE, or IDE).

   - Inquiry trigger: A Request where the requested line is valid in a CPU's L1 cache.
   - Inquiry expected by CPU: No (unsolicited inquiry)
   - Inquiry address source: 1st beat of Request message.
   - Tag lookup requirements: Determine if Inquiry is required, and target.
   - Inquiry target: Identified by duplicate L1 tag lookup. If the Request is non-cacheable or from the ICACHE, the Inquiry target may be the same CPU as the Request source.

2. Expected Inquiry due to read with replacement eviction (IRE).

   When the CPU requests a line, it is possible that the line being replaced is modified. The Request message indicates (in the request type) that a L1 eviction is required. In these cases, Che performs two tag lookups when servicing the Request. The first lookup uses the address of the line being read, and the second lookup uses the address of the evicted line. Tag consistency checks are performed on both tag lookups.

   If a replacement eviction is required, Che sends a Replacement Inquiry (IRE) to the requesting CPU. Replacement Inquiries differ from other Inquiries in that Che does not wait for the InqReply before sending the ReqReply for the original Request. This can occur because the evicted line is not required to complete the coherency operations for the requested line. In anticipation of the Inquiry, the CPU stores the evicted line in an eviction buffer. This frees space in the L1 cache for the requested line.

   - Inquiry trigger: RLE or RLME Request
   - Inquiry expected by CPU: Yes
   - Inquiry address source: 2nd beat of Request message.
   - Tag lookup requirements: Determine if inquiry still required.
   - Inquiry target: Request originator.

3. Expected Inquiry due to the CPU cache instruction or independent replacement eviction (IRE).

   - Inquiry trigger: WLI or WLS Request
   - Inquiry expected by CPU: Yes
   - Inquiry address source: 1st beat of Request message.
   - Tag lookup requirements: Determine if Inquiry is still required.
   - Inquiry target: Request originator.

9.11. L2 Cache Line Replacement

When the L2 cache line must be allocated as indicated in the transaction table, a pseudo-LRU policy is used to select the way for replacement:

- If one or more invalid ways are available, the lowest numbered invalid way is selected.
- If all ways are valid, the least recently or next to least recently used is selected.

The pseudo-LRU algorithm require a three bit field to store state information. This field is updated on all L2 cache accesses as follows:

```
If access to way 0 or 1
   LRU[2] = 0
   LRU[1] = LRU[1]
   LRU[0] = way
Else If access to way 2 or 3
   LRU[2] = 1
   LRU[1] = way - 2
   LRU[0] = LRU[0]
```

The following algorithm is used to select a way for replacement:

```
If one or more ways invalid
   way = lowest numbered invalid way
Else If LRU[2] = 0
   way = !LRU[1] + 2
Else If LRU[2] = 1
   way = !LRU[0]
```

No transaction table is included for L2 replacements. When a replacement occurs, the new L2 state is based on the requesting transaction.

9.12. Coherency Effects of Crossbar Queues

When the CPU issues a Request message, the message type is a function of the CPU operation and the L1 cache state. In certain cases, the L1 cache state that led to a particular Request message may change before the Request is serviced.

The XB provides a queue that orders and serializes Request messages from CPUs and DMA engines. It is possible for the queue to contain Requests from different CPUs that target the same cache line. If one CPU's Request leads to a state change in a second CPU's L1 cache, an inconsistency may exist when the second CPU's Request is subsequently processed. The inconsistency appears as a mismatch between the message type and the duplicate L1 tags. These inconsistencies and the required actions are reflected in the transaction tables.

Inconsistencies of this type occur in cases where a CPU issues Requests involving lines that are valid. This happens in the following cases:

1) When a CPU issues an UM Request, it intends to upgrade a L1 line from S to M. Consider this sequence of events:

```
Line: L0
CPUs: C0, C1

L0 is S at C0
L1 is I at C1
      C0 issues UM for L0
      C1 issues RLM for L0
      Che services C1 RLM for L0
L0 is I at C0
L0 is M at C1
      Che services C0 UM for L0
L0 is M at C0
L0 is I at C1
```

Che assumes that although an UM Request from C0 for L0 is received, L0 was invalidated by another CPU. This invalidation occurred between the time that the UM Request was issued and serviced. As a result, Che returns the target line to C0.

2) When a M state L1 line must be replaced to make room for a new line, the CPU indicates this in the Request message. Consider this sequence of events:

```
Lines: L0, L1
CPUs: C0, C1

L0 is I at C0
L1 is I at C0
L0 is M at C1
L1 is I at C1
      C0 issues RL for L0
      C1 issues RLE for L1 evicting L0
      Che services C0 RL for L0
L0 is E at C0
L0 is I at C1
L1 is unchanged at C0 and C1
      Che services C1 RLE for L1 evicting L0
L0 is unchanged at C0 and C1
L1 is unchanged at C0
L1 is E at C1
```

Che assumes that although a RLE request from C1 to evict L0 is received, L0 was invalidated by another CPU. As a result, Che does not send an IRE for L0 to C1.

9.13. Configuration

In the following sections, "n" refers to the MS number.

9.13.1. MSnCfg Register

Name: MSnCfg.
Size: 32 bits.
Address: MSBase + 0x0 + n*0x40.
Restrictions: Modify only during system configuration. Must be set to the same value for all MS.

31-24	23-16	15-6	5-4	3-2	1	0
Branch0	Branch1	TagNum	Reserved	NumCS	NumMC	InitTag

Field	Bits	Description	R/W	Reset
Branch0	31-24	First character of branch (ASCII format)	R	valid
Branch1	23-16	Second character of branch (ASCII format)	R	valid
TagNum	15-6	Tag Number	R	valid
NumCS	3-2	Number of physical memory ranks (chip selects) 00: 1 CS 01: 2 CS 10: reserved 11: 4 CS	R/W	0
NumMC	1	Number of MCs to use 0: 1 MC 1: 2 MC	R/W	0
InitTag	0	Initialize the state of the L2 tags and L1 duplicate tags to Invalid. This bit is cleared automatically when tag initialization is complete.	R/W	0

9.13.2. MSnPld Register

Name: MSnPld.
Size: 32 bits.
Address: MSBase + 0x100 + n*0x40.
Restrictions: Modify only during system configuration. Must be set to the same value for all MS.

31-11	10-1	0
Reserved	MemReqDly	Pld

Field	Bits	Description	R/W	Reset
Pld	0	Implementation type 0: ASIC implementation 1: PLD implementation	R/W	0
MemReqDly	10-1	Memory clocks that must elapse between memory requests in PLD implementations	R/W	0

9.13.3. MSnMemCtl Registers

Name: MSnMemCtl.
Size: 16 bits/register
Address: MSBase + Offset + n*0x40
Restrictions: Modify only during system configuration. Must be set to the same value for all MS.

These registers reside within the Denali memory controller. Refer to the "Denali Databahn Memory Processor" document for register details.

Table 39: Memory Controller Register Offsets

MC Register	Offset
MSnMem_Main	0x0200
MSnMem_CTLA	0x0202
MSnMem_CTLB	0x0204
MSnMem_CTLC	0x0206
MSnMem_CTLD	0x0208
MSnMem_CTLE	0x020A
MSnMem_CTLF	0x020C
MSnMem_CTLG	0x020E
MSnMem_CTLH	0x0210
MSnMem_CTLI	0x0212
MSnMem_CTLJ	0x0214
MSnMem_ECC1	0x0216
MSnMem_ECC2	0x0218
MSnMem_ECC3	0x021A
MSnMem_ECC4	0x021C
MSnMem_ECC5	0x021E
MSnMem_ECC6	0x0220
MSnMem_ECC7	0x0222
MSnMem_ECC8	0x0224
MSnMem_ECC9	0x0226
MSnMem_ECC10	0x0228
MSnMem_ECC11	0x022A
MSnMem_ECC12	0x022C
MSnMem_ECC13	0x022E
MSnMem_ECC14	0x0230
MSnMem_ECC15	0x0232
MSnMem_ECC16	0x0234
MSnMem_ECC17	0x0236

9.14. Performance Counters

There are two performance counters per MS. The event counted by each counter may be independently selected in the MSnPcntEv* registers. The counters are enabled by writing to the MSnPcntEn register. The local enable bits are local to each MS. The global bits from MS0 are ANDed with the local bits in each MS to form the counter enable signal. The global bits in MS1 are unused.

9.14.1. MSnPcnt0, MSnPcnt1 Register

Name: MSnPcnt0, MSnPcnt1.
Size: 32 bits.
Address: MSBase + 0x1000 + n*0x40, MSBase + 0x1100 + n*0x40.
Restrictions:

31-0
Count

Field	Bits	Description	R/W	Reset
Count	31-0	Count value	R/W	0

9.14.2. MSnPcntEn Register

Name: MSnPcntEn.
Size: 32 bits.
Address: MSBase + 0x1200 + n*0x40.
Restrictions: Global enables in MS 1 are unused.

31-4	3	2	1	0
Reserved	GblEnCnt1	GblEnCnt0	LclEnCnt1	LclEnCnt0

Field	Bits	Description	R/W	Reset
LclEnCnt0	0	Local counter 0 enable	R/W	0
LclEnCnt1	1	Local counter 1 enable	R/W	0
GblEnCnt0	2	Global counter 0 enable (Only valid in MS0)	R/W	0
GblEnCnt1	3	Global counter 1 enable (Only valid in MS0)	R/W	0

9.14.3. MSnPcntEv0, MSnPcntEv1 Register

Name: MSnPcntEv0, MSnPcntEv1.
Size: 32 bits.
Address: MSBase + 0x1300 + n*0x40, MSBase + 0x1400 + n*0x40.
Restrictions Some fields are mutually exclusive; e.g. Wr AND Rd can never occur.

31-29	28-27	26-25	24-23	22-21	20-19	18-17	16-15
Reserved	InqSh	InqMod	InqExMod	InqEx	InqEvict	Up	Wr

14-13	12-11	10-9	8-7	6-5	4-3	2-1	0
RdEvict	RdIntent	Rd	L2Hit	Size	CpuAgnt	Agnt	Mode

Field	Bits	Description	R/W	Reset
Mode	0	0: count clocks 1: count requests (match conditions from the fields below are ANDed together to qualify this)	R/W	0
Agnt	2-1	0x: don't care 10: DMA 11: CPU	R/W	0
CpuAgnt	4-3	0x: don't care 10: ICACHE 11: DCACHE	R/W	0
Size	6-5	0x: don't care 10: sub-line 11: line	R/W	0
L2Hit	8-7	0x: don't care 10: L2 miss 11: L2 hit	R/W	0
Rd	10-9	0x: don't care 10: not read 11: read	R/W	0
RdIntent	12-11	0x: don't care 10: not read with intent to modify 11: read with intent to modify	R/W	0
RdEvict	14-13	0x: don't care 10: not read with replacement eviction 11: read with replacement eviction	R/W	0
Wr	16-15	0x: don't care 10: not write 11: write	R/W	0
Up	18-17	0x: don't care 10: not upgrade 11: upgrade	R/W	0
InqEvict	20-19	0x: don't care 10: no inquiry for replacement eviction 11: inquiry for replacement eviction	R/W	0
InqEx	22-21	0x: don't care 10: no inquiry to exclusive line 11: inquiry to exclusive line	R/W	0
InqExMod	24-23	0x: don't care 10: no inquiry to exclusive line that returns dirty data 11: inquiry to exclusive line that returns dirty data	R/W	0
InqMod	26-25	0x: don't care 10: no inquiry to modified line 11: inquiry to modified line	R/W	0
InqSh	28-27	0x: don't care 10: no inquiry to shared line 11: inquiry to shared line	R/W	0

9.15. Error Handling

Several types of errors are detected and reported in the MS. When an enabled error condition occurs, information on the type of the error and the associated message header is stored. The CPU responsible for managing errors is informed of the error via INT, PANIC0, or PANIC1. When an enabled error condition not related to data occurs, Inquiries and Tag updates are inhibited. In the case of errors associated with data, it is not possible to inhibit these actions.

The errors detected and possible actions are shown in Table 40.

Table 40: MS Errors

Error Type	Encoding	Description	Inquiries and Tag Updates	Header Type Saved
TagCons	0000	Inconsistency between tags: - L1 = M and L2 = S - one L1 = E and another L1 valid - one L1 = M and another L1 valid - more than one L1 way valid - more than one L2 way valid	No	Request
TypCons	0001	Inconsistency between request type and L1 tag. These cases are defined in the transaction tables.	No	Request
AdrCons	0010	Inconsistency between request type and address: - Line request to register address space	No	Request
AgntCons	0011	Inconsistency between request type and agent: - RLN, WLN from CPU - RL, RLE, RLM, RLME, UM from DMA	No	Request
ReqUn	0100	Request message unexpected	No	Request
IqrUn	0101	InqReply message unexpected or of incorrect type	No	InqReply
ReqBt	0110	Incorrect number of beats in Request message	No	Request
IqrBt	0111	Incorrect number of beats in InqReply message	No	InqReply
IqrTO	1000	InqReply message timeout	No	Inquiry
WesTO	1001	Crossbar unavailable for westbound beat	No Yes	Inquiry or ReqReply
TagPar	1010	Tag parity error	No	Request
L2DatPar	1011	L2 data parity error	Yes	ReqReply
UncorrEcc	1100	Uncorrectable memory ECC error	Yes	ReqReply
CorrEcc	1101	Correctable memory ECC error	Yes	ReqReply

9.15.1. MSnErrEn0, MSnErrEn1 Register

The action performed in response to each error type may be separately enabled. In all cases, the normal ReqReply cycle is returned to the requesting CPU. The MSnErrEn register is used to enable error actions. Each error has three possible actions, which are enabled by setting the enable to "1".

Name: MSnErrEn0, MSnErrEn1.
Size: 32 bits.
Address: MSBase + 0x2000 + n*0x40, MSBase + 0x2004 + n*0x40.
Restrictions: Modify only during system configuration. Must be set to the same value for all MS.

MSnErrEn0: MSnErrEn1:

31-24	23-0
Reserved	*En

31-18	17-0
Reserved	*En

Field	Register	Bits	R/W	Reset
TagConsEn	0	2-0	R/W	0
TypConsEn	0	5-3	R/W	0
AdrConsEn	0	8-6	R/W	0
AgntConsEn	0	11-9	R/W	0
ReqUnEn	0	14-12	R/W	0
IqrUnEn	0	17-15	R/W	0
ReqBtEn	0	20-18	R/W	0
IqrBtEn	0	23-21	R/W	0
IqrTOEn	1	2-0	R/W	0
WesTOEn	1	5-3	R/W	0
TagParEn	1	8-6	R/W	0
L2DatParEn	1	11-9	R/W	0
UncorrEccEn	1	14-12	R/W	0
CorrEccEn	1	17-15	R/W	0

Bit	Enable
0	PANIC0
1	PANIC1
2	ERR_INT

9.15.2. MSnErrTO Register

Name: MsnErrTO.
Size: 32 bits.
Address: MSBase + 0x2100 + n*0x40.
Restrictions: Modify only during system configuration. Must be set to the same value for all MS.

31-16	15-8	7-0
Reserved	WesTOVal	IqrTOVal

Field	Bits	Description	R/W	Reset
WesTOVal	15-8	Bits 15:8 of timeout count on crossbar availability for a westbound beat	R/W	0
IqrTOVal	7-0	Bits 15:8 of timeout count on receiving all expected InqReply messages.	R/W	0

9.15.3. MSnErrStat0, MSnErrStat1 Register

When an error occurs, the MS stores the error type and a copy of the message header associated with the error. This data is stored in the MSnErrStat registers. If only correctable ECC errors occur, these registers hold information about the first correctable ECC error. If any fatal errors occur, these registers contain information about the first fatal error.

Name: MSnErrStat0, MSnErrStat1.
Size: 32 bits.
Address: MSBase + 0x2200 + n*0x40, MSBase + 0x2204 + n*0x40.
Restrictions:

MSnErrStat0:

63-60	59-32
Error Type	Message Header

MSnErrStat1:

31-0
Message Header

Field	Bits	Description	R/W	Reset
Error Type	63-60	Error Type (see Table 40, "MS Errors")	R/W	0
Message Header	59-0	Message Header (see crossbar chapter for format)	R/W	0

9.16. Interfaces

9.16.1. Crossbar Interface

Table 41: Control Interface

Signal	Bits	I/O	Description
XB_CLK	1	I	Crossbar clock
XB_RESET_N	1	I	System reset

Table 42: Eastbound Request Message Interface

Signal	Bits	I/O	Description
REQ_VLD	1	I	Message beat present on REQ_DAT
REQ_STA	1	I	First message beat present on REQ_DAT
REQ_RDY	1	O	MS accepts current message beat
REQ_DAT[63:0]	64	I	Message beat contents

Table 43: Eastbound InqReply Message Interface

Signal	Bits	I/O	Description
IQR_VLD	1	I	Message beat present on IQR_DAT
IQR_STA	1	I	First message beat present on IQR_DAT
IQR_RDY	1	O	MS accepts current message beat
IQR_DAT[63:0]	64	I	Message beat contents

Table 44: Westbound Message Interface

Signal	Bits	I/O	Description
WES_VLD	1	I	Message beat present on WES_DAT
WES_STA	1	I	First message beat present on WES_DAT
WES_RDY	1	O	MS accepts current message beat
WES_DAT[63:0]	64	I	Message beat contents

9.16.2. Interrupt Interface

Table 45: Interrupt Interface

Signal	Bits	I/O	Description
MS_ERR_INT_R	1	O	CPU interrupt
MS_PANIC0_R	1	O	PANIC0 signal
MS_PANIC1_R	1	O	PANIC1 signal

9.16.3. SDRAM Interface

Table 46: SDRAM Interface

Signal	Bits	I/O	Description
MC_RAS_N	1	O	Row address strobe
MC_CAS_N	1	O	Column address strobe
MC_WE_N	1	O	Write enable
MC_S_N[3:0]	4	O	Chip select
MC_BA[1:0]	2	O	Bank address
MC_A[13:0]	14	O	Address
MC_DQ[71:0]	72	I/O	Data bus
MC_DQS[8:0]	9	I/O	Data strobe
MC_DQM[8:0]	9	O	Data mask
MC_CKE	1	O	Clock enable
MC_MEMCLK	1	I	SDRAM clock

9.17. Che Transactions

The following transaction tables define the Actions and New State for all possible combinations of Requests and Current cache states. The following notes apply to all tables:

- Valid states (S E M) imply address match.
- *All L1* state refers to aggregate state of all CPU DCACHEs as reflected in the duplicate L1 tags.
- *Req L1* state refers to state of requestor's CPU DCACHE as reflected in the duplicate L1 tags.
- *Other L1* state refers to aggregate state of all CPU DCACHEs other than the requestor as reflected in the duplicate L1 tags.
- When enabled TagCons or TypeCons errors occur, inquiries and tag updates do not occur. However, the appropriate ReqReply is always returned. The ReqReply when an error is enabled is the same as the ReqReply when the error is disabled.

Eastbound Transaction Request	Current State		Actions	New State		Notes
	All L1	L2		All L1	L2	
RL (ICACHE)	I	I	data mem - RQR DLS to requestor	I	I	
	S	I	data mem - RQR DLS to requestor	S	I	
	E	I	IDE to L1 data mem - RQR DLS to requestor	S	I	if L1 returns modified data treat as MI
	M	I	IIE to L1 data IQR - mem, RQR DLS to requestor	I	I	
	I	S	data L2 - RQR DLS to requestor	I	S	
	S	S	data L2 - RQR DLS to requestor	S	S	
	E	S	IDE to L1 data L2 - RQR DLS to requestor	S	S	if L1 returns modified data treat as MS (error disabled)
	M	S	TagCons Error	NC	NC	
			If error disabled: IIE to L1 data IQR - L2, mem, RQR DLS to requestor	I	S	

Note:

There is no L1 DCACHE state change due to the ReqReply message. For ICACHE requests, DCACHE state changes occur only due to Inquiry messages.

Eastbound Transaction Request	Current State			Actions	New State			Notes
	Req L1	Other L1	L2		Req L1	Other L1	L2	
RL RLE	I	I	I	data mem - RQR DLE to requestor	E	I	I	
	I	S	I	allocate L2 data mem - L2, RQR DLS to requestor	S	S	S	
	I	E	I	IDE to other allocate L2 data mem - L2, RQR DLS to requestor	S	S	S	if other returns modified data treat as IMI
	I	M	I	IIE to other data IQR - mem, RQR DLE to requestor	E	I	I	
	I	I	S	data L2 - RQR DLE to requestor	E	I	S	
	I	S	S	data L2 - RQR DLS to requestor	S	S	S	
	I	E	S	IDE to other data L2 - RQR DLS to requestor	S	S	S	if other returns modified data treat as IMS (error disabled)
	I	M	S	TagCons Error	NC	NC	NC	
				If error disabled: IIE to other data IQR - L2, mem, RQR DLE to requestor	E	I	S	

Eastbound Transaction Request	Current State			Actions	New State			Notes
	Req L1	Other L1	L2		Req L1	Other L1	L2	
RLM RLME UM	I	I	I	data mem - RQR DLM to requestor	M	I	I	
	I	S	I	II to other data mem - RQR DLM to requestor	M	I	I	
	I	E	I	IIE to other data mem - RQR DLM to requestor	M	I	I	if other returns modified data treat as IMI
	I	M	I	IIE to other data IQR - RQR DLM to requestor	M	I	I	
	I	I	S	data L2 - RQR DLM to requestor	M	I	I	
	I	S	S	II to other data L2 - RQR DLM to requestor	M	I	I	
	I	E	S	IIE to other data L2 - RQR DLM to requestor	M	I	I	if other returns modified data treat as IMS (error disabled)
	I	M	S	TagCons Error	NC	NC	NC	
				If error disabled: IIE to other data IQR - RQR DLM to requestor	M	I	I	

Note:

Cases where "Req L1" = I on UM occur due to invalidation by other CPU between the time the request is issued and serviced

.

Eastbound Transaction Request	Current State			Actions	New State			Notes
	Req L1	Other L1	L2		Req L1	Other L1	L2	
RL, RLE RLM, RLME	S E M	*	*	TypCons Error	NC	NC	NC	If error disabled, treat as I** in previous tables

Eastbound Transaction Request	Current State			Actions	New State			Notes
	Req L1	Other L1	L2		Req L1	Other L1	L2	
UM	S	I	I	UMA to requestor	M	I	I	
	S	S	I	II to other UMA to requestor	M	I	I	
	S	E	I	TagCons Error	NC	NC	NC	If error disabled, treat as SSI
	S	M	I	TagCons Error	NC	NC	NC	If error disabled, treat as SSI
	S	I	S	UMA to requestor	M	I	I	
	S	S	S	II to other UMA to requestor	M	I	I	
	S	E	S	TagCons Error	NC	NC	NC	If error disabled, treat as SSS
	S	M	S	TagCons Error	NC	NC	NC	If error disabled, treat as SSS
	E M	*	*	TypCons Error	NC	NC	NC	If error disabled, treat as S**

Eastbound Transaction Request	Current State			Actions	New State			Notes
	Req L1	Other L1	L2		Req L1	Other L1	L2	
VE WLI WLS	I	I	I	None	I	I	I	
	I	S	I	None	I	S	I	
	I	E	I	None	I	E	I	
	I	M	I	None	I	M	I	
	I	I	S	None	I	I	S	
	I	S	S	None	I	S	S	
	I	E	S	None	I	E	S	
	I	M	S	TagCons Error	NC	NC	NC	If error disabled, treat as IIS
	S	*	*	TypCons Error	NC	NC	NC	If error disabled, treat as I**
	E M	I	I	IRE to requestor data IQR - mem	I	I	I	On WLS, New Req L1 = S
	E M	S E M	I	TagCons Error	NC	NC	NC	If error disabled, treat as [EM]II
	E	I	S	IRE to requestor data IQR - L2, mem	I	I	S	On WLS, New Req L1 = S
	M	I	S	TagCons Error	NC	NC	NC	If error disabled, treat as EIS
	E M	S E M	S	TagCons Error	NC	NC	NC	If error disabled, treat as EIS

Eastbound Transaction Request	Current State			Actions	New State			Notes
	Req L1	Other L1	L2		Req L1	Other L1	L2	
LI	*	*	*	None	I	NC	NC	

Notes:

Address supplied with LI is only a tag index. Address match is not checked. Requested way is unconditionally invalidated.

Eastbound Transaction Request	Current State		Actions	New State		Notes
	All L1	L2		All L1	L2	
RLN	I	I	data mem - RQR DL to requestor	I	I	
	S	I	II to L1 data mem - RQR DL to requestor	I	I	
	E	I	IIE to L1 data mem - RQR DL to requestor	I	I	if L1 returns modified data treat as MI
	M	I	IIE to L1 data IQR - mem, RQR DL to requestor	I	I	
	I	S	data L2 - RQR DL to requestor	I	S	
	S	S	II to L1 data L2 - RQR DL to requestor	I	S	
	E	S	IIE to L1 data L2 - RQR DL to requestor	I	S	if L1 returns modified data treat as MS (error disabled)
	M	S	TagCons Error	NC	NC	
			If error disabled: IIE to L1 data IQR - L2, mem, RQR DL to requestor	I	S	

Eastbound Transaction Request	Current State		Actions	New State		Notes
	All L1	L2		All L1	L2	
WLN	I	I	data REQ - mem WLA to requestor	I	I	
	S	I	II to L1 data REQ - mem WLA to requestor	I	I	
	E	I	II to L1 data REQ - mem WLA to requestor	I	I	
	M	I	II to L1 data REQ - mem WLA to requestor	I	I	
	I	S	data REQ - L2, mem WLA to requestor	I	S	
	S	S	II to L1 data REQ - L2, mem WLA to requestor	I	S	
	E	S	II to L1 data REQ - L2, mem WLA to requestor	I	S	
	M	S	TagCons Error	NC	NC	
			If error disabled: II to L1 data REQ - L2, mem WLA to requestor	I	S	

Eastbound Transaction Request	Current State		Actions	New State		Notes
	All L1	L2		All L1	L2	
RB RH RT RW	I	I	data mem - RQR DS to requestor	I	I	
	S	I	II to L1 data mem - RQR DS to requestor	I	I	
	E	I	IIE to L1 data mem - RQR DS to requestor	I	I	if L1 returns modified data treat as MI
	M	I	IIE to L1 data IQR - mem data mem - RQR DS to requestor	I	I	
	I	S	data L2 - RQR DS to requestor	I	S	
	S	S	II to L1 data L2 - RQR DS to requestor	I	S	
	E	S	IIE to L1 data L2 - RQR DS to requestor	I	S	if L1 returns modified data treat as MS (error disabled)
	M	S	TagCons Error	NC	NC	
			If error disabled: IIE to L1 data IQR - L2, mem data L2 - RQR DS to requestor	I	S	

Eastbound Transaction Request	Current State		Actions	New State		Notes
	All L1	L2		All L1	L2	
WB WH WT WW	I	I	data REQ - mem WSA to requestor	I	I	
	S	I	II to L1 data REQ - mem WSA to requestor	I	I	
	E	I	IIE to L1 data REQ - mem WSA to requestor	I	I	if L1 returns modified data treat as MI
	M	I	IIE to L1 data IQR - mem data REQ - mem WSA to requestor	I	I	
	I	S	data REQ - mem WSA to requestor	I	I	
	S	S	II to L1 data REQ - mem WSA to requestor	I	I	
	E	S	IIE to L1 data REQ - mem WSA to requestor	I	I	if L1 returns modified data treat as MS (error disabled)
	M	S	TagCons Error	NC	NC	
			If error disabled: IIE to L1 data IQR - mem data REQ - mem WSA to requestor	I	I	

10.1. Direct Memory Access Overview

The Stream Processor provides multiple Direct Memory Access (DMA) controllers for high-speed data transfer between the SDRAM and external interfaces, and from SDRAM to SDRAM.

- Three Ethernet DMA controllers.
  - Ethernet to SDRAM, SDRAM to Ethernet.
  - 1 Gbps full duplex bandwidth for Gigabit Ethernet.
  - Sustainable 64-byte packet transfers.
  - Programmable hashing to depth of 128 bytes to determine input queue assignment.
- One PCI-X DMA controller.
  - PCI-X to SDRAM, SDRAM to PCI-X.
  - 2 Gbps data bandwidth.
- One Memory Move DMA controller.
  - SDRAM to SDRAM transfer.
  - 10 Gbps data bandwidth.
- Features common to all DMA controllers:
  - 36-bit physical address space.
  - Memory-resident buffer descriptors and queues, up to 1M entries each.
  - Atomic enqueue and dequeue supports multi-processor access to DMA controller.
  - Minimal use of interrupts.
  - Performance counters.
  - Checksum calculation.

Figure 17 shows the Stream Processor's DMA controllers. Representative DMA pathways are shown.

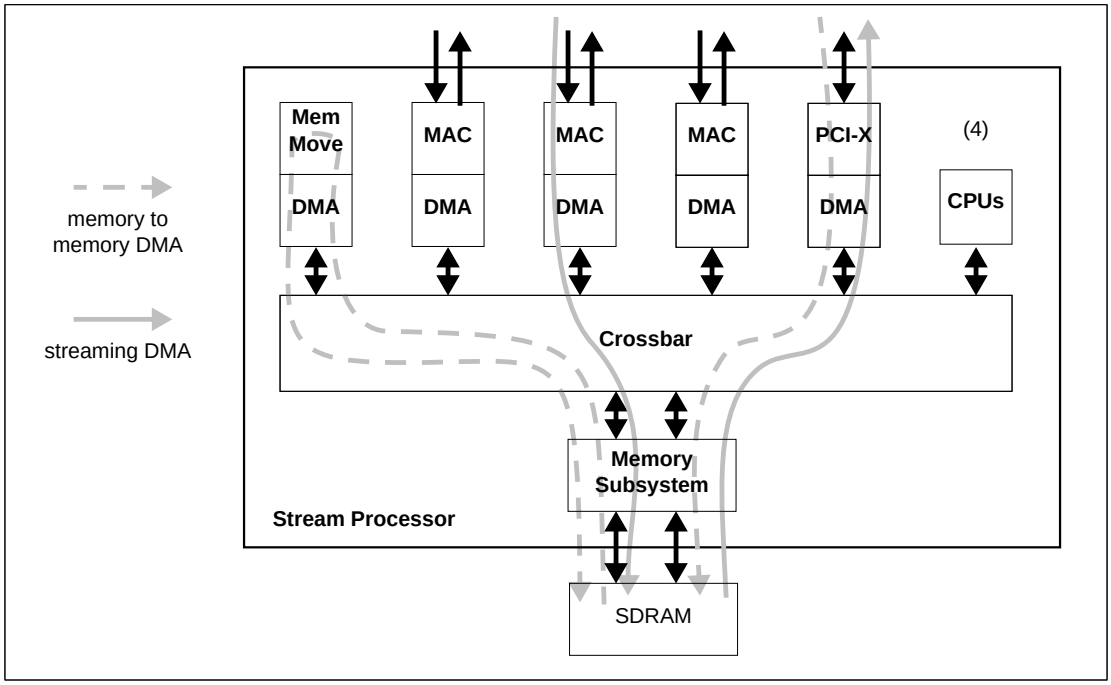

Figure 17: Direct Memory Access System Overview

The Stream Processor's Ethernet interfaces and PCI-X interface include a dedicated DMA controller. Depending on the nature of the interface, each controller supports a streaming or memory to memory data transfer mode. The streaming mode transfers packet data between a FIFO and SDRAM. The memory to memory mode transfers data from SDRAM to SDRAM, or between SDRAM and memory on the PCI-X bus.

Table 47 indicates the bandwidths associated with each DMA controller. A more detailed view of performance is given in Section 10.14.

Table 47: DMA Capabilities

Interface	Modes	Data BW (Gbps)
ETH0_Rx	streaming	1.0
ETH0_Tx	streaming	1.0
ETH1_Rx	streaming	1.0
ETH1_Tx	streaming	1.0
ETH2_Rx	streaming	1.0
ETH2_Tx	streaming	1.0
PCIX	streaming, memory to memory	2.0
MM	memory to memory	10.0

10.2. Addressing

The DMA controllers operate in the Stream Processor's 36-bit *physical* address space (64 GBytes). Software typically employs the CPU's MMU to translate addresses in the CPU's 32-bit logical address space to the Stream Processor's 36-bit physical address space. These logical to physical mappings are not accessible to the DMA controllers. All pointers that are stored in DMA data structures consist of physical addresses. Software must perform logical to physical translations to construct DMA data structures. Software must also ensure that DMA data structures spanning more than one physical page frame occupy consecutive page frames in physical memory.

Depending on the memory management and buffer allocation methods employed by software, the buffer space addressable by DMA may be limited to 2 GBytes, or may be as large as 64 GBytes. The 2 GByte configuration allows the simplest memory management and buffer allocation scheme. Larger DMA configurations require more software overhead for buffer management.

Although the Stream Processor supports a 64 GByte physical address space, the Memory Subsystem supports up to 8 GBytes of attached SDRAM. Thus, Ethernet MAC DMA and Memory Move DMA operations are limited to the low 8 GBytes of the physical address space. Memory that is attached to the Stream Processor through PCI-X may increase total physical memory beyond 8 GBytes. This additional memory is generally accessible by the processors, and can also be accessed for DMA through the Stream Processor's PCI-X DMA controller.

10.2.1. Ethernet and PCI-X DMA Controller Organization

Figure 18 illustrates the structure of the Stream Processor's Ethernet and PCI-X DMA controllers, including the on-chip Ethernet MAC.

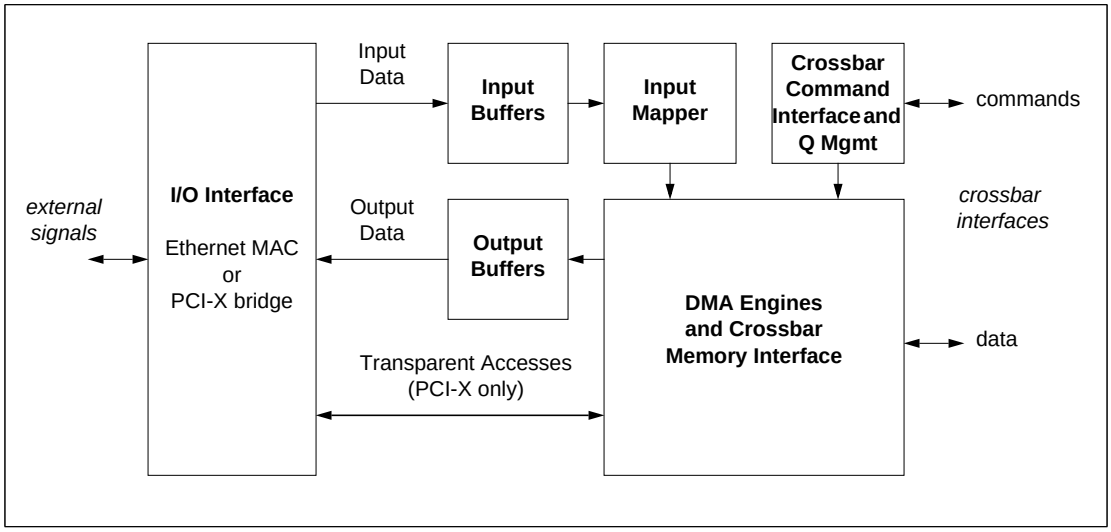

Figure 18: Ethernet and PCI-X DMA Controller Organization

The Ethernet and PCI-X DMA Controllers are composed of the following modules:

- Ethernet MAC or PCI-X bridge - Provides standard GMII/MII or PCI-X interface. Converts the external data paths to internal 250 MHz data paths. Sources input data and requests output data. The PCI-X bridge also supports transparent master/target access between the PCI-X bus and the crossbar.
- Input Buffers - Obtains data from the MAC's input interface. Provides buffering.
- Input Mapper - Parses the beginning of each packet (up to 128 bytes) and generates the input queue assignments. Passes all packet data and queue assignments to the DMA engines.
- Crossbar Processor Interface and Queue Management - Provides interface functions between processors and the DMA controller (through the crossbar). Responds to uncached load and store instructions that are used to configure the DMA controller, monitor status and interact with the queue logic.
- DMA Engines and Crossbar Memory Interface - Accepts input packets and queue assignments from the Input Mapper, and passes output packets to the Output Buffers. Performs memory reads and writes for descriptors and packet data (through the crossbar). Includes multiple parallel engines to maintain high concurrency.
- Output Buffers - Provides buffering of transmit data. Supplies data requests to the output interface.

The Ethernet and PCI-X DMA controllers include a programmable packet mapper that examines each input packet up to a depth of 128 bytes to determine the ID of a input queue to which the packet is assigned. Typically, the mapper is programmed to generate a hash number from fields in the layer 3 and 4 headers to determine the input queue assignment. Alternatively, the mapper can also be programmed to extract a queue assignment directly from the packet, or perform a round-robin assignment. The mapper is described in more detail in Sections 10.6.

The Ethernet and PCI-X DMA controllers provide five categories of queues.

- 128 input pending queues.
- 128 input completed queues.
- 1 high priority output pending queue.
- 1 low priority output pending queue.
- 128 output completed queues.

The output pending queues are shared among all software threads. The association of all other queues to software threads is under software control. There may be a one-to-one mapping of queues to threads. Alternatively, a many-to-one mapping allows the software to dynamically adjust the workload by changing the assignment of input queues to software threads. The DMA controller provides an efficient exclusive access mechanism for all queues that does not require software to resort to semaphores, even when multiple threads are accessing the same queue.

Figure 19 illustrates the flow of Ethernet and PCI-X DMA transactions. Two basic alternatives are shown. One involves software in the recycling of buffers from transmit to receive operations. The other alternative allows the DMA controller to directly recycle free buffers after a transmit operation is successfully completed, shown with dotted lines.

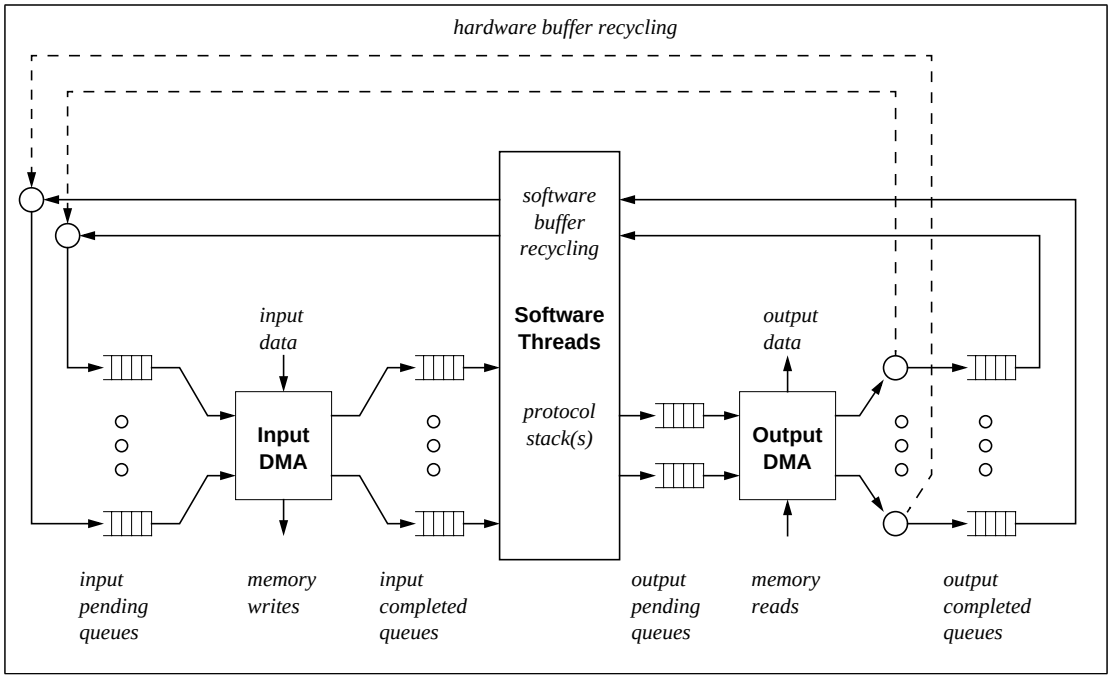

Figure 19: Ethernet and PCI-X DMA Flow

To set up an input or output DMA operation, software enqueues a buffer descriptor into a pending queue. The DMA controller dequeues descriptors from this queue, performs a transfer, and enqueues the descriptor into a completed queue.

The PCI-X DMA Controller can perform streaming and memory-to-memory DMA operations. While the PCI-X interface itself is not inherently streaming, the DMA architecture include provisions for transfers to and from FIFO-like devices on the PCI-X bus. Static addresses may be used for read and write operations on the PCI-X bus, in contrast to incrementing the address as a transfer progresses The mapper can be used to parse the headers of streaming PCI-X data to determine the transfer length.

10.2.2. Memory Move DMA Controller Organization

Figure 20 illustrates the structure of a the Stream Processor's Memory Move (MM) DMA controller.

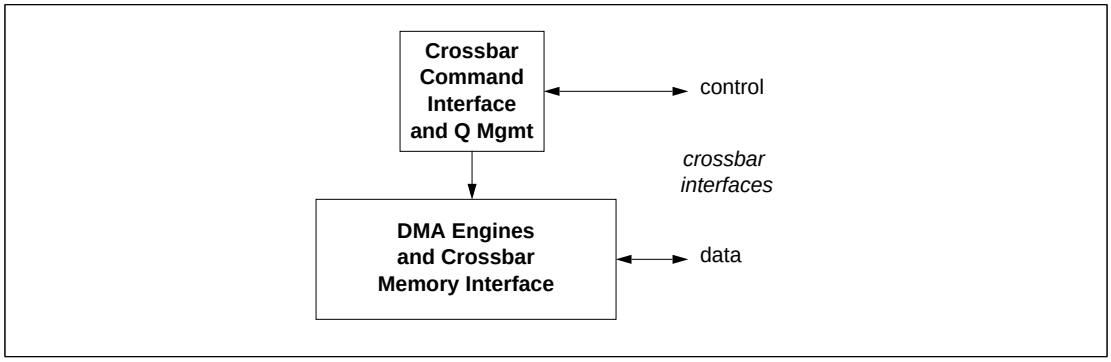

Figure 20: Memory Move DMA Controller Organization

The Memory Move DMA Controller is composed of the following modules:

- Crossbar Processor Interface and Queue Management - Provides interface functions between processors and the DMA controller (through the crossbar). Responds to uncached load and store instructions that are used to configure the DMA controller, monitor is status and interact with the queue logic.
- DMA Engines and Crossbar Memory Interface - Accepts queue information from the Queue Management logic. Performs memory reads and writes for descriptors and packet data (through the crossbar). Includes multiple parallel engines to maintain concurrency.

In he MM DMA controller, data is transferred from SDRAM to SDRAM.

The memory to memory DMA controller provides two categories of queues.

- 16 transfer pending queues.
- 16 transfer completed queues.

The association between queues and software threads is software defined.

Figure 21 illustrates the flow of Memory Move DMA transactions.

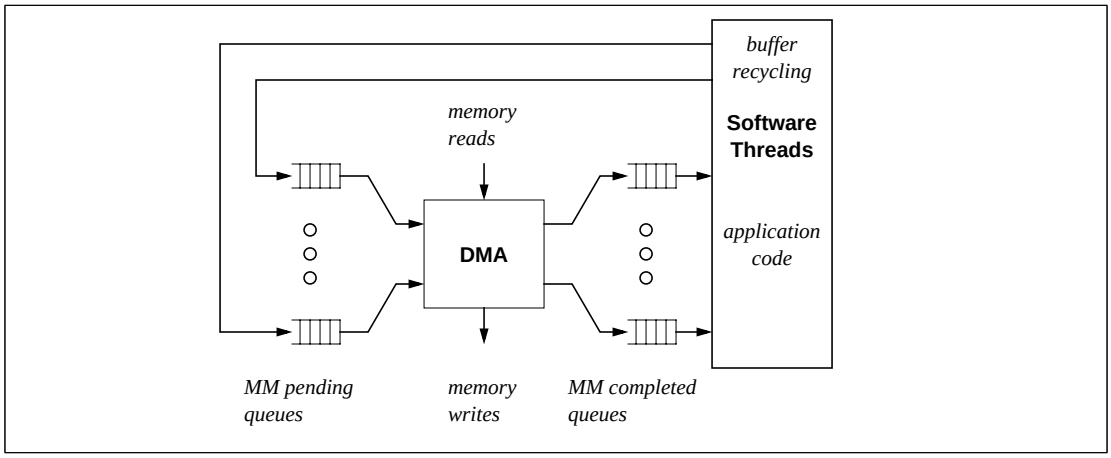

Figure 21: Memory Move DMA Flow

To set up a Memory Move DMA transfer, software enqueues a buffer descriptor into a pending queue. The DMA controller dequeues descriptors from this queue, performs a transfer, and enqueues the descriptor into a completed queue.

10.3. Queue Configuration

The location and size of the DMA controller's queues in main memory are specified with five sets of queue configuration registers, corresponding to the four queue types. All queues of a type have the same characteristics, which are governed by the configuration registers.

Table 48 summarizes the registers that configure a queue type. There are four such register sets in the DMA controller.

Table 48: Queue Configuration Registers

Name	Size (bits)	Description
Flags	?	Control and status information. TBD.
StartPtr	30	Address of start of queue. High order 30 bits of a 36-bit pointer (i.e. 64-byte aligned).
Size	20	Number of queue entries.
HighCount	20	High water entry count.
LowCount	20	Low water entry count.

The DMA controller also maintains hardware registers specific to each individual queue, which contain the state summarized in Table 49. These registers are not accessed by software during normal operation.

Table 49: Per-Queue State Registers

Name	Size (bits)	Description
Flags	?	Control and status information. TBD.
EnqPtr	20	Current enqueue offset.
DeqPtr	20	Current dequeue offset.
EntryCount	20	Current number of allocated entries.

10.4. Queue Operation

Software follows these steps to enqueue a buffer descriptor.

1. Software executes an uncached load word instruction that targets a memory mapped register within the DMA controller. (See Table 53.) The DMA controller returns the byte offset of a newly allocated location in the queue. The DMA controller sets the high order bit (bit 31) in the returned value if the queue is full, in which case software should repeat step 1. (The DMA controller can be configured to also cause an interrupt for this condition.)

2. Software adds the byte offset to the logical address of the base of the queue (which it maintains independent of the DMA controller). The result is a byte pointer to a 64-byte region in memory that is 64-byte aligned. The Valid flag (in word 0) is currently zero, indicating that the descriptor is currently invalid.

3. Software writes the required descriptor words, described in Section 10.5. The last write must be to the word that holds the descriptor's Valid flag (set to 1) to ensure mutual exclusion with DMA controller's access to the queue. After software sets the Valid flag in the descriptor it is available for use by the DMA controller. Software must not read or write the descriptor after the Valid flag has been set.

Up to 64 consecutive queue descriptors can be allocated with a single load word instruction in step 1. The DMA controller provides 64 target addresses for the load word operation in step 1, corresponding to allocation requests from 1 to 64 entries. The DMA controller reserves the appropriate number of queue entries before returning the queue offset value. The high order bit is set appropriately to indicate the outcome of the allocation attempt. When a valid offset is obtained, software must perform steps 2 and 3 for each allocated descriptor. However, if the software is setting up descriptors specify a series of buffers to transfer a single packet, the Valid flag for the *first* descriptor must not be set until all other descriptors of the packet have been written.

Software follows these steps to dequeue a buffer descriptor.

1. Software executes an uncached load word instruction that targets a memory mapped register in the DMA controller. (See Table 53.) The controller returns the byte offset within the queue of the buffer descriptor to be dequeued. The DMA controller sets the high order bit (bit 31) in the returned value if the queue is empty, in which case software should repeat step 1. (The DMA controller can be configured to also cause an interrupt for this condition.)

2. Software adds the byte offset to logical address of the base of the queue (which it maintains independent of the DMA controller). The result is a byte pointer to a 64-byte region in memory that is 64-byte aligned. The Valid flag (in word 0) indicates the validity of the entry. However, the descriptor is not necessarily valid at this time.

3. If the Valid bit is 0, the queue entry is invalid (i.e. has not yet been updated by the DMA controller). This is a queue underflow condition, and is rare in normal operation. Software must loop and test the valid bit until it is set to 1 by the DMA controller. Note that the descriptor is resident in the CPU's cache, so repeated testing does not cost system performance. Because the Stream Processor maintains memory coherency, software will automatically observe the new descriptor contents when it is updated by the DMA controller.

4. Software clears the Valid bit in the descriptor within the queue. After software clears the Valid flag in the descriptor, the queue entry is free for re-use by the DMA controller. Software must not read or write the descriptor within the queue after the Valid flag has been cleared. Instead, it must use its descriptor copy.

For the Ethernet and PCI-X DMA controllers, it is not possible to dequeue multiple entries in step 1.

For the Memory Move DMA controller, up to 64 consecutive queue descriptors can be de-allocated with a single load word instruction in step 1. The DMA controller provides 64 target addresses for the load word operation in step 1, corresponding to de-allocation requests from 1 to 64 entries. The DMA controller de-allocates the appropriate number of queue entries before returning the queue offset value. The high order bit is set appropriately to indicate the outcome of the de-allocation attempt. When a valid offset is obtained, software must perform steps 2 through 4 for each de-allocated descriptor.

The association of DMA controller queues to threads is managed by software, and is not known to the DMA controller. The types of associations have an impact on the amount of memory that can be shared in software. A globally managed queue may accessed by all software threads. A dedicated queue is accessed only by a specific software thread. The use of global queues requires globally shared buffers in both the logical and physical address spaces, which effectively limits DMA buffers to 2 GBytes of SDRAM. The use of dedicated queues allows separate logical address spaces to be maintained for each software thread (or pools of threads), which in turn allows DMA buffers to access all 8 GBytes of available SDRAM.

10.5. Buffer Descriptors

Buffer descriptors are stored in SDRAM resident queues, as described in Section 10.4. A given queue is used for input, output or memory to memory DMA operations, and holds only the corresponding type of buffer descriptor. Each descriptor is stored in one 64-byte aligned location within the queue, and includes a pointer to the data buffer, the buffer size, and other parameters.

The contents of the buffer descriptors are shown in Table 50. The eight columns at the right of the table correspond to the ways that descriptors may be used:

IP	Input buffer descriptor in Ethernet or PCI-X input pending queue.
IC	Input buffer descriptor in Ethernet or PCI-X input completed queue.
OP	Output buffer descriptor in Ethernet or PCI-X output pending queue.
OC	Output buffer descriptor in Ethernet or PCI-X output completed queue.
DP	Destination buffer descriptor in MM pending queue.
DC	Destination buffer descriptor in MM completed queue.
SP	Source buffer descriptor in MM pending queue.
SC	Source buffer descriptor in MM completed queue.

A code in the eight columns indicate when the descriptor field is used:

E	Ethernet transfer.
P	PCI-X transfer.
B	Ethernet or PCI-X transfer.
M	Memory to memory (SDRAM) transfer.
(blank)	Field is not used, must be zero.

Software must supply zeroes for unused fields when writing descriptors. The DMA controller supplies zeroes for any unused fields when writing descriptors.

Table 50: Buffer Descriptor Contents

Word	Field	Name	Description	IP	IC	OP	OC	DP	DC	SP	SC
0	31	Valid	Indicates the validity of the buffer descriptor. 1 - Descriptor is valid. 0 - Descriptor is not valid. Set by software when descriptor is enqueued in a pending queue, cleared by the DMA controller when descriptor is dequeued. Set by the DMA controller when descriptor is enqueued in a completed queue, cleared by software when descriptor is dequeued.	B	B	B	B	M	M	M	M
0	30	ErrTruncated	1 - Data was truncated because the packet is larger than the configured maximum packet size. (Configuration method is TBD.) 0 - Data was not truncated.		E						
0	29	ErrUnderrun	1 - Transmit FIFO underrun occurred, and automatic retransmit was not enabled via DMA configuration. 0 - No underrun occurred.				E				
0	28	ErrIPHdrChk	1 - Error in IP header checksum. 0 - P header checksum OK or IP header not checked.		E						
0	27	ErrFrameCRC	1 - Error in frame CRC. 0 - Frame CRC OK.		E						
0	26	ErrAbort	1 - Packet was aborted by the receiver interface. 0 - Packet was not aborted.		E						

Table 50: Buffer Descriptor Contents (Continued)

Word	Field	Name	Description	IP	IC	OP	OC	DP	DC	SP	SC
0	25	ErrAddr	1 - An address error occurred while transferring data to or from memory. 0 No address error occurred.		B		B		M		M
0	24	IPHdrChecked	1 - IP header checksum was verified, ErrIPHdr=1 if error. 0 - IP header checksum was not verified.		B						
0	23	More	1 - Next descriptor specifies more data or storage for this transfer. 0 - Data or storage is concluded with this descriptor.		B	B	B	M	M	M	M
0	22	Continuation	1 - Data or storage is continued from previous descriptor. 0 - First descriptor.		B	B	B	M	M	M	M
0	21	Interrupt	Enables interrupt generation upon completion of transfer. The target of the interrupt is defined by the (TBD) register. This feature should be used sparingly, as frequent interrupts can reduce performance substantially. 1 - Generate an interrupt upon completion of the transfer. 0 - No interrupt is generated.	B	B	B	B	M	M	M	M
0	20	ZeroBuffer	Controls clearing of data buffers after the transfer. When enabled, the DMA controller writes the entire buffer with zeroes after the transfer is completed. The BuffSize value, not the XferSize value, is used to determine the size of the region that is cleared. The DMA controller delays enqueueing the descriptor onto the completed queue until the buffer is cleared. 1 - Clear the buffer after the transfer is completed. 0 - Do not clear the buffer after the transfer.							M	M
0	19	InhibitTranfser	Inhibits the use of the descriptor for data transfer. 1 - Do not use descriptor for data transfer. Software can use this mode to measure the elapsed time from software enqueue into a pending queue to software dequeue from a completed queue. For this use, software typically puts a timestamp into the pending descriptor's SWValue field. A buffer clear operation without transmission is performed when ZeroBuffer=1 and Inihibit=1 for an output descriptor. 0 - Normal use of descriptor.	B	B	B	B	M	M	M	M
0	18	StaticAddress	1 - Do not increment the buffer pointer during data transfer. This mode is used to access a FIFO attached to the PCI-X bus. 0 - Increment the buffer pointer during data transfer.	P	P	P	P	M	M	M	M
0	17:12	(reserved)	Must be zero.								
	11:8	PortNumber	DMA interface through which input packet was received. 0 - Ethernet MAC 0. 1 - Ethernet MAC 1. 2 - Ethernet MAC 2. 3 - PCI-X.		B						
0	7:0	BuffPtrHi	High order 4 bits of the 36-bit byte pointer to the data buffer. The 4 bits are stored right justified in this 8-bit field, and the remaining 4 bits of the field must be zero.	B	B	B	B	M	M	M	M
1	31:0	BuffPtrLo	Low order 32 bits of the 36-bit byte pointer to the data buffer. For input buffer descriptors, this address must be 64-byte aligned.	B	B	B	B	M	M	M	M

Table 50: Buffer Descriptor Contents (Continued)

Word	Field	Name	Description	IP	IC	OP	OC	DP	DC	SP	SC
2	31:16	BuffSize	For pending descriptors, written by software to indicate the buffer's allocated byte size, in 64-byte increments. For completed descriptors, the DMA controller copies the value from corresponding pending descriptor. The value is encoded modulo 64K (65,536). A value of 0 represents a buffer size of 65,536 bytes. Must be a multiple of 64 bytes.	B	B	B	B	M	M	M	M
2	15:0	XferSize	Transfer size in bytes, modulo 64K (65,536). Updated by the DMA controller to indicate the actual number of bytes transferred.		B		B		M		M
3	31:16	StartOffset	Byte offset of first data byte within buffer. For input packets the cache lines that are skipped as a result of a non-zero StartOffset are not modified by the DMA controller, and the contents of the beginning of the first partially modified cache line are undefined.	B		B	B	M	M	M	M
3	15:8	CompQIfGood	Identifies the queue to which the descriptor is enqueued after the transfer is successfully completed.			B					
3	7:0	CompQIfBad	Identifies the queue to which the descriptor is enqueued if the transfer is not successfully completed.			B					
4	31:24	TimeStamp	An 8-bit packet timestamp, representing the time that the packet entered the mapper. The 8-bit timestamp is obtained from timestamp counter, which is described in Section 10.9		E						
4	23:0	(reserved)	Must be zero.								
5	31:0	SoftwareValue	This field is application dependent. Typically it is used to hold the logical address of the data buffer, or a pointer to a data structure that software uses to access the data buffer. The DMA controller preserves the contents of this field when modifying or copying an input descriptor from the pending queue to the completed queue. The data in this field has no effect on packet receipt or transmission.	B	B	B	B	M	M	M	M
6	31:16	Checksum	Updated with the 16-bit one's complement of the one's complement sum of all 16-bit words in the packet, not including the MAC header. If the data to be checksummed consists of an odd number of bytes, the last byte is right-padded with zeroes for the purpose of the checksum calculation.		B				M		
6	15:8	DestOffset	Offset of the destination buffer descriptor, relative to the location of this descriptor. This field is valid only in the first descriptor of a memory-to-memory copy, which is the first source buffer descriptor.					M	M	M	M
6	7:0	(reserved)	Must be zero.								
7	31:0	MapperOut0	The contents of mapper output registers R0 through R3 at the conclusion of the mapper program.		B						
8	31:0	MapperOut1	The contents of mapper output registers R4 through R7 at the conclusion of the mapper program.		B						
9	31:0	MapperOut2	The contents of mapper output registers R8 through R11 at the conclusion of the mapper program.		B						
10	31:0	MapperOut3	The contents of mapper output registers R12 through R15 at the conclusion of the mapper program.		B						

Table 50: Buffer Descriptor Contents (Continued)

Word	Field	Name	Description	IP	IC	OP	OC	DP	DC	SP	SC
11	31:0	DeviceStatus0	Status bits supplied by the interface device after completion of transfer. See Section 11.3.		B		B				
12	31:0	DeviceStatus1	Status bits supplied by the interface device after completion of transfer. See Section 11.3.		B		B				
13-15	31:0	(pad)	Pad descriptor to 64 bytes.								

The following figures illustrate the relationships between queues, the buffer descriptors, and buffers.

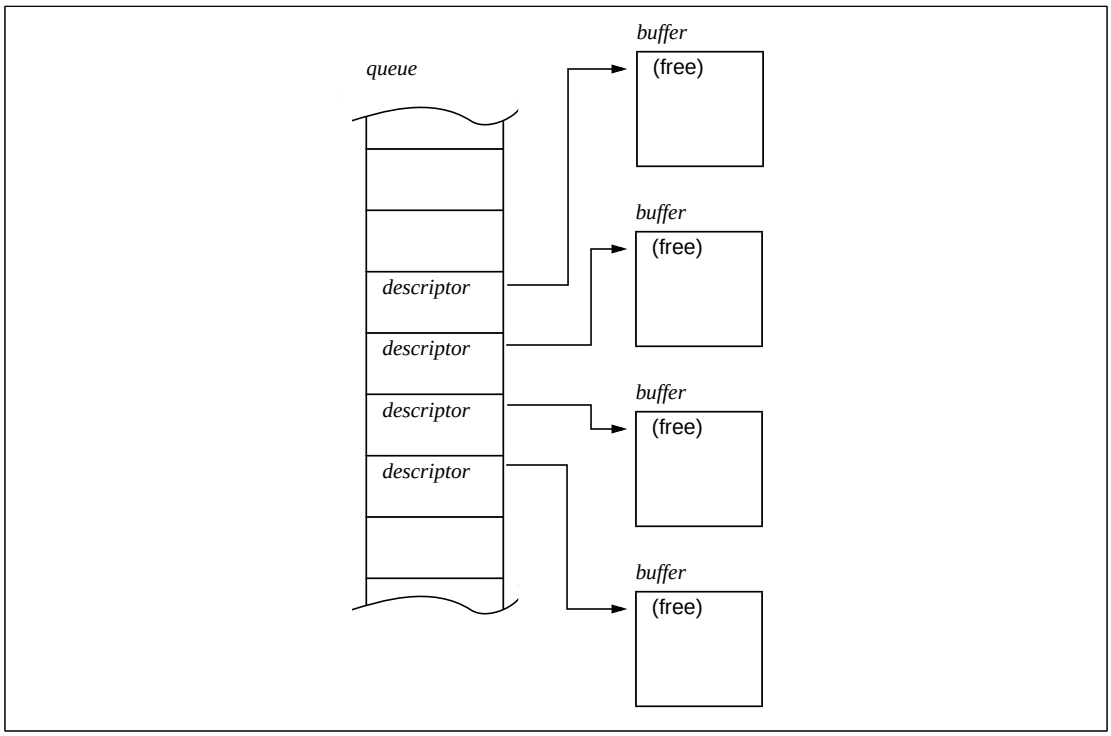

Figure 22: Input Pending Queue

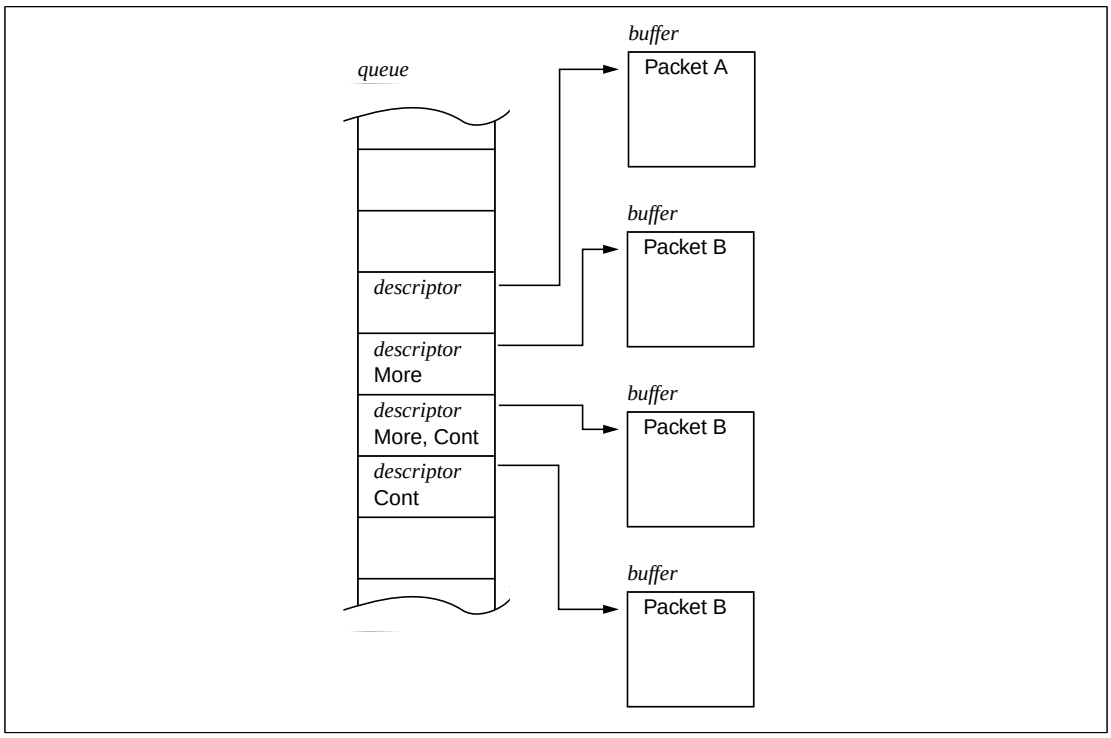

Figure 23: Input Completed, Output Pending, Output Completed Queues

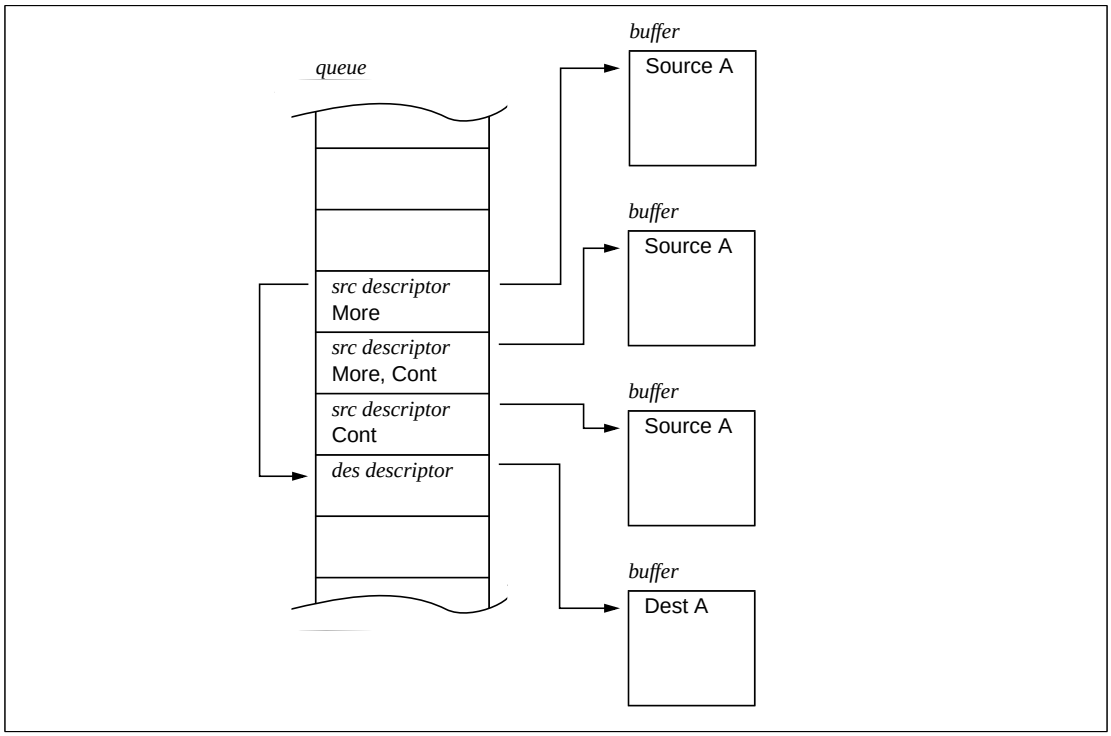

Figure 24: Memory to Memory Copy Pending and Completed Queues

When software does not employ page-level logical to physical address mapping, there are no restrictions on the sizes and page alignment of individual buffers. The direct mapping approach imposes very little overhead on buffer management, but exposes all of memory to different processes and requires more complex safe programming practices. Page level mapping requires more overhead in buffer management, but shields processes from each other. The method used for a given operating system and application depends on required levels of performance and protection.

Software that uses page-level mapping must ensure that individual data buffers fit within a page of memory, as defined by the MMU TLB settings. In its logical address space a buffer may have an arbitrary size and page alignment. Due to page-level logical to physical address mapping, a data region that spans more than one logical page cannot be expected to reside in contiguous physical pages. It is the responsibility of software to account for this by constructing an appropriate list of buffer descriptors.

10.6. Input Queue Assignment with Packet Mapper

The Ethernet and PCI-X DMA controllers include a packet mapper that determines the pending and completed queue assignments for inbound packets. The mapper parses packets to a depth of 128 bytes to identify protocols and extract bit fields. Programmable mapper functions operate on extracted bits to generate the assigned packet queue numbers and other values as shown in 25.

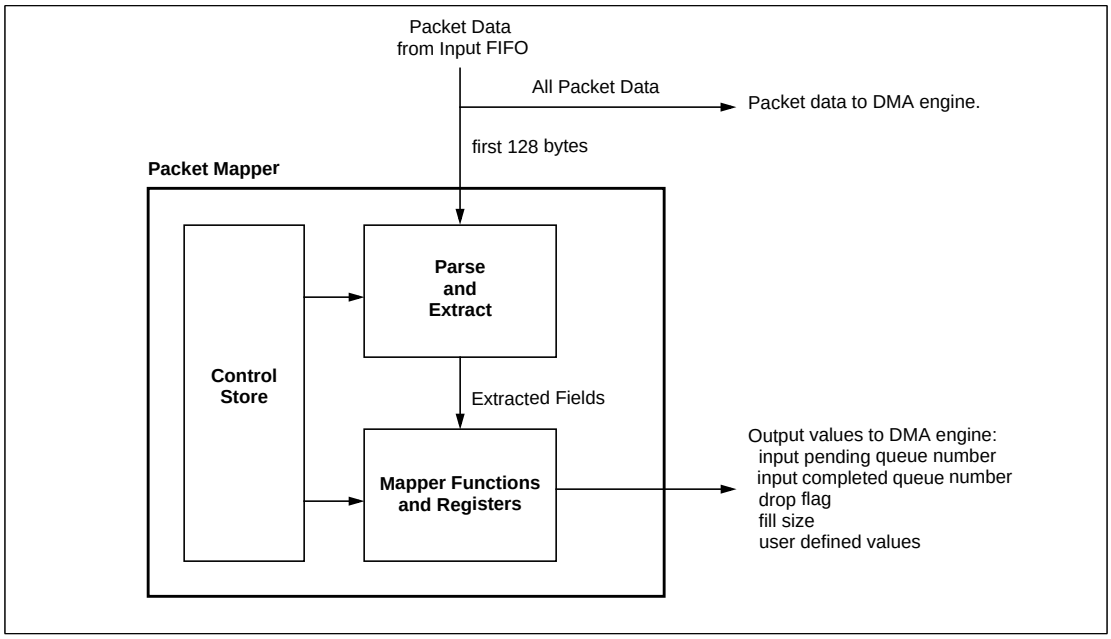

Figure 25: Packet Mapper Data Flow

When the Stream Processor is reset, the mapper is disabled and all register contents are cleared to zero. Until the mapper program is loaded, all input packets bypass the mapper and are referred to input pending queue 0 and input completed queue 0. The mapper program is loaded with a Lexra-supplied API that runs on one of the Stream Processor's CPUs, typically during system initialization. The API copies the program binary from the Stream Processor's shared SDRAM into the mapper's control store by storing the program words into memory-mapped locations that fall within the mappers configuration space. When the copying is completed, the API enables mapper operation by setting a bit in a mapper control register.

A total of sixteen general purpose 8-bit registers are available to the mapper's user program. When the mapper program has completed processing of the first 128 bytes, the final contents of these registers are passed to the DMA engines and determine how the packet will be handled. (See Table 51.)

Table 51: Mapper Registers

Register	Bits	Use Upon Completion of Program
0	7:0	Input pending queue assignment.
1	7:0	Input completed queue assignment.
2	7:3	Reserved. Must be zero.
2	2:1	Fill size (0 to 64 bytes).
2	0	Packet drop flag.
3	7:0	Byte offset for start of CRC calculation.
4-15	7:0	User defined values written to descriptor.

A variety of queue assignment schemes can be programmed. For example, packet field values can directly determine the assigned queues. Alternatively, the packet field values can be constructed into a key that is hashed to distribute the packets among a pool of queues. Hash keys can be selected such that related packets are always assigned to the same queue. These details are application-dependent, and are supported with the mapper's programmable architecture.

The mapper's parser validates each packet and passes packet fields to the field processing functions. Table 52 summarizes the field processing functions that are supported. The following conventions are used:

boldface	Statement keywords.
statements	One or more of the statements listed in Table 52.
reg	Identifies one of 16 general-purpose 8-bit registers. The final values of the following registers determine how the packet is handled by the DMA engines. See Table 51
packet_field	Identifiles a bit field from the parsed packet. This consists of the field name from the formal packet description, followed by an optional bit range in square brackets. Bits are numbered with 0 in the least signficant bit position.
constant8	An 8-bit constant specified with standard Verilog notation
constant	Any size constant specified with standard Verilog notation.

Table 52: Field Processing Statements

Statement	Description
reg = *constant8*;	Load register with 8-bit constant. *reg* ← *packet_field*
regd = *regs*;	Copy register contents. *regd* ← *regs*

Table 52: Field Processing Statements (Continued)

Statement	Description
reg = *packet_field*;	Load register from packet field. *reg* ← *packet_field* The data from *packet_field* is loaded right-justified into *reg*. The field must specify 8 bits or less of data.
reg = **offset** (*packet_field*);	Load register with byte offset of packet field. *reg* ← **offset** (*packet_field*) The byte offset of the beginning of *packet_field* relative to the start of the frame is loaded into reg.
reg = *reg & constant8*;	Load register with bit-wise AND of register contents and 8-bit constant. *reg* ← *reg & constant8*
reg = *reg* + *constant8*;	Load register with sum of register contents and 8-bit constant. *reg* ← *reg* + *constant8*
reg = **rotr** (*reg*);	Rotate register right by 1 bit. *save* ← *reg*[0] *reg*[6:0] ← *reg*[7:1] *reg*[7] ← *save*
reg = **rotl** (*reg*);	Rotate register left by 1 bit. *save* ← *reg*[7] *reg*[7:1] ← *reg*[6:0] *reg*[0] ← *save*
reg = **crc8** (*reg*, *packet_field*);	CRC-8 accumulation of register with packet field. *reg* ← **crc8** (*reg*, *packet_field*) The data specified by *packet_field* is right-padded to an 8-bit boundary and is CRC8-accumulated with the current contents of *reg*. The result is stored in *reg*.
reg = **xor8** (*reg*, *packet_field*);	8-bit XOR accumulation of register with packet field. reg ← XOR8 (*reg*, *packet_field*) The data specified by *packet_field* is right-padded to an 8-bit boundary. Individual bytes from the padded field are XOR-accumulated with the current contents of *reg*. The result is stored into *reg*.
finish;	Finish program. The mapper terminates processing of the current packet. Packet data and register values are passed to the DMA engine. The mapper waits for a new packet.
if (*expression*) { *statements* }	If statement. If *expression* evaluates true, execute the statements in the if clause. The *expression* is one of the following forms. *reg* == *constant8* *reg* <= *constant8* (unsigned arithmetic) *packet_field* == *constant* *packet_field* <= *constant* (unsigned arithmetic)

Table 52: Field Processing Statements (Continued)

Statement	Description
if (*expression*) { *statements* } **else** { *statements* }	If-else statement. If *expression* evaluates true, execute the *statements* in the if clause. Otherwise execute the *statements* in the else clause. The *expression* forms are listed above.

By assigning values to the output registers, the user's mapper program determines the input pending and input completed queues that are used to process a packet. The association of these queues to software threads is managed in the software domain, and is not known to the DMA controller. Although it is possible for any software thread to dequeue a packet from any queue, software typically assigns queues to specific threads. The queue to thread associations that are maintained by software need not be static. An efficient form of load balancing can be implemented in software by configuring sufficiently more input queues than threads, and modifying the queue to thread associations when a thread workload imbalance arises.

Figure 26 illustrates an example use of the mapper. It discriminates three types of traffic, and directs each packet type to a pool of queues reserved for that type. Application specific software threads in turn dequeue packets from their assigned queues. The figure also illustrates how software can perform load balancing by modifying the queue to thread associations.

1. The DMA controller directs the first 128 bytes of each input packet to the mapper.

2. The mapper program parses the packet to determine the packet type.

3. Depending on the type, the mapper program selects hash key fields from the packet and accumulates a hash number. The hash number is added to base numbers to determine the input pending and input completed queues from pools that are dedicated to the packet type. Alternatively, a direct queue assignment can be made based on the protocol type.

4. The DMA controller dequeues a descriptor from the assigned input pending queue, performs the DMA transfer, and enqueues the descriptor to the assigned input completed queue.

5. The software thread that is responsible for the input completed queue dequeues the packet.

6. Software processes the packet.

7. Software typically maintains state for the packet and other packets in the same microflow.

8. Software can re-assign an entire input completed queue and associated state to a different thread. This process is efficient if software maintains separate microflow state tables for each queue.

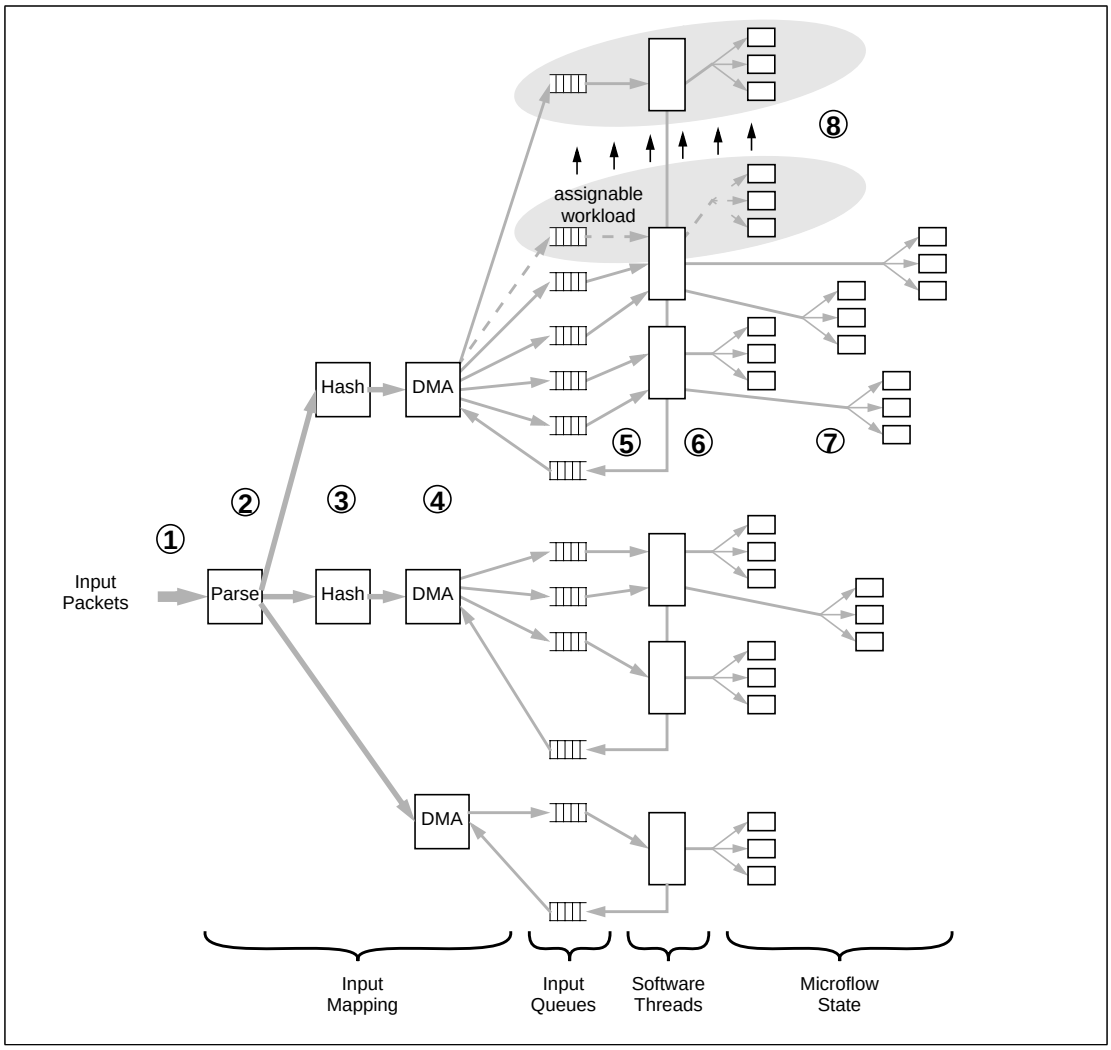

Figure 26: Input Mapping and Workload Assignment

10.7. Inserting Leading Fill Into Input Packets

When the mapper program completes the processing of a packet, the value in mapper register 2 specifies the number of zero fill bytes (0 to 64) to insert at the front of the packet in memory.

One use of fill is to align frequently accessed packet fields at a 32-bit word boundary in memory to allow efficient access to the fields from the application running on the Stream Processor's CPUs. For example, without inserting fill an IP header within a MAC frame would start at a half-word boundary and unaligned loads would be required to reference the 32-bit fields within the IPv4 and higher layer headers. When the mapper is programmed to insert two bytes of fill at the start of a MAC frame, the application can access the 32-bit fields with aligned memory references.

Another use of fill is to make room for proprietary headers that are subsequently updated and maintained by the application.

10.8. Skipping Leading Fill From Output Packets

Skipping data from an output packet is the analogous to inserting fill into input packets, as described in the previous section. When an application running on the Stream Processor's CPUs prepares an output packet, the most efficient program execution is realized if packet headers are properly aligned in memory. If the output packet was originally an input buffer, fill bytes may have been inserted by the user's mapper program. (The number of fill bytes inserted by the mapper can be find in the MapperOut0 word of the buffer descriptor.) If the packet is held in a buffer created by CPU software, fill bytes may have been inserted by the software. In either case, software can cause the fill bytes to be skipped when the packet is enqueued to the DMA controller for an output operation by storing an appropriate value the StartOffset field of the output descriptor. Note that skipping leading fill for output packets does not involve the DMA controller's mapper.

10.9. Input Packet Timestamp

The Ethernet DMA controller generates an 8-bit timestamp for all input packets and includes it in the descriptor that is written to the input completed queue. The timestamp is derived from a 24-bit free-running counter that increments every 256 ns (about 1/2 the time for a minimum size packet). A configuration register (TBD) controls the selection of the contiguous 8-bit field from the counter that is used as the timestamp. The timestamp is captured at the moment the first byte of a packet enters the packet mapper.

A software thread that manages multiple input queues can use the timestamp to ensure that it services the queues fairly.

10.10. Output Queue Selection

The Ethernet and PCI-X DMA controllers provide a low priority output pending queue and a high priority output pending queue. The DMA controller services all entries in the high priority output queue ahead of any entries in the low priority queue.

10.11. Interrupts

Interrupts can be enabled independently for controller and targeted to specific CPU contexts and interrupt lines. The following interrupt events can be enabled per queue.

- Queue underflow.
- Queue overflow.
- Low water crossing.
- High water crossing.
- Packet-specific events as defined in buffer flags (See Table 50 on page 104.)
- First packet, i.e. a input packet is enqueued to a previously empty queue.

The generation of an interrupt for the first packet can be delayed up to TBD processor cycles. This allows the application to establish a maximum interrupt delay for low rate data streams without causing excessive interrupts when a burst of packets arrives.

10.12. Checksum Calculation

The DMA controller performs checksum calculations for data transferred in streaming and memory to memory modes.

- The Ethernet DMA controller verifies the IP header checksum of input packets and stores the result in ErrIPHdrChk flag of the buffer descriptor.
- The Ethernet DMA controller calculates the 16-bit checksum over the entire input packet, starting at a byte offset that is determined by the packet mapper and excluding the trailing FCS field. The result is stored in the Checksum field of the buffer descriptor.
- The memory to memory and PCI-X controllers calculate the 16-bit checksum over the entire data transfer and stores the result in the Checksum field of the destination buffer descriptor.

Software can utilize the Checksum in the buffer descriptor of packets received through Ethernet DMA to simplify the calculation of higher layer checksums such as TCP and UDP. To compute the TCP checksum, for example, software can subtract IP header contents from the entire packet Checksum value.

10.13. Error Detection and Handling

The following error conditions are detected by the DMA controller.

- Bus error.
- Bad frame.
- MAC status.
- Output FIFO underflow.
- Input FIFO overflow.
- Interface parity error.
- Internal SRAM parity error.

All errors are indicated in descriptor that is enqueued on the transfer completed queue. The DMA controllers can be also configured signal a hardware error in the event of parity error. See Chapter 17.

Additional details to be supplied.

10.14. Memory Bandwidth Requirement

Figure 27 shows the memory bandwidth required for the DMA of different packet sizes through the Gigabit Ethernet interface. For example, receiving 500 byte packets over a 1 Gbps interface requires about 1.5 Gbps of memory bandwidth.

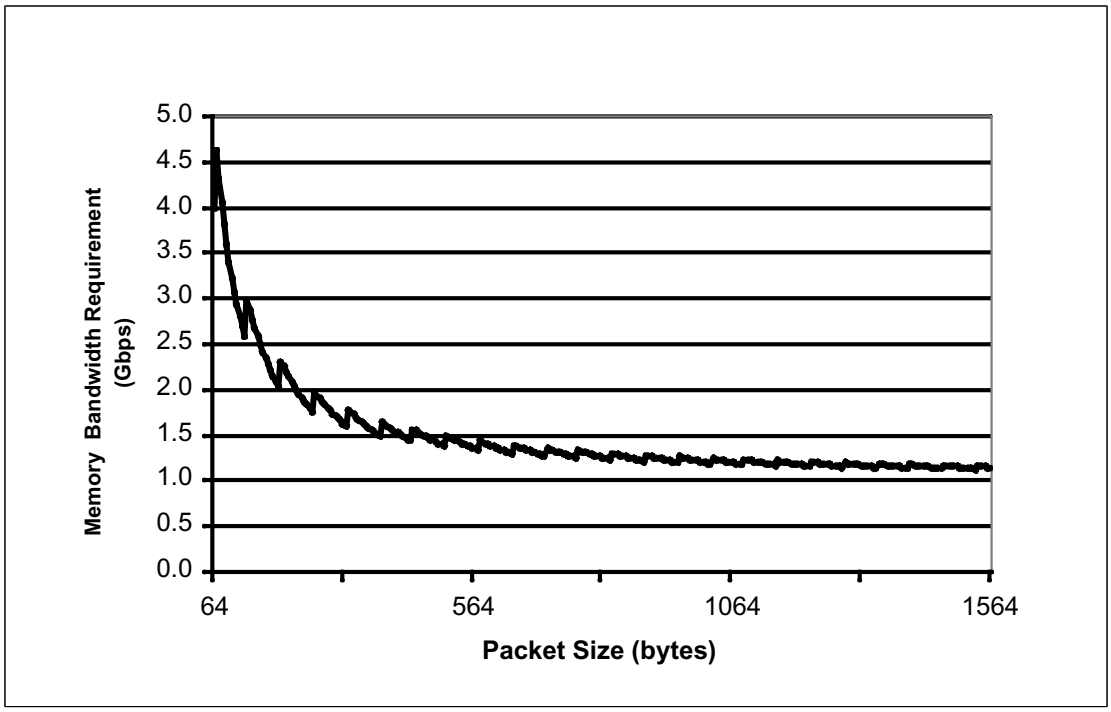

Figure 27: Gigabit Ethernet Memory Bandwidth Requirement

There are several important characteristics of this curve. First, the total memory bandwidth required generally decreases as the packet size increases. This is because a fixed amount of memory bandwidth overhead is present for each packet, due the buffer descriptors which must be read and written by the DMA controller. Second, the curve shows spikes as a result of 64-byte quantization that occurs when data is transferred.

10.15. DMA Controller Registers

All DMA capabilities are accessed through registers that are mapped within the in a 64 KByte address space that is dedicated to each DMA controller. This region also provides access specific interface functions, as summarized in Section 11.4. Table 53 summarizes the DMA controller's queue access, configuration and status registers. The system address of each register is determined by adding the offset to the applicable base value defined in Section 5.5

Table 53: MAC Configuration and Status Registers

Offset	Bit	Definition	Access	Reset
Ethernet and PCI-X Enqueue and Dequeue Operations				
0x0000	31:0	Input pending queue 0, enqueue 1 entry.	Read	0
0x0004	31:0	Input pending queue 0, enqueue 2 entries.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x00FC	31:0	Input pending queue 0, enqueue 64 entries.	Read	0
0x0100	31:0	Input pending queue 1, enqueue 1 entry.	Read	0
0x0104	31:0	Input pending queue 1, enqueue 2 entries.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x01FC	31:0	Input pending queue 1, enqueue 64 entries.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x7F00	31:0	Input pending queue 127, enqueue 1 entry.	Read	0
0x7F04	31:0	Input pending queue 127, enqueue 2 entries.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x7FFC	31:0	Input pending queue 127, enqueue 64 entries.	Read	0
0x8000	31:0	Input complete queue 0, dequeue 1 entry.	Read	0
0x8004	31:0	Input complete queue 1, dequeue 1 entry.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x81FC	31:0	Input complete queue 127, dequeue 1 entry.	Read	0
0x8200	31:0	Output pending queue 0, enqueue 1 entry.	Read	0
0x8204	31:0	Output pending queue 0, enqueue 2 entries.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x82FC	31:0	Output pending queue 0, enqueue 64 entries.	Read	0
0x8300	31:0	Output pending queue 1, enqueue 1 entry.	Read	0
0x8304	31:0	Output pending queue 1, enqueue 2 entries.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x83FC	31:0	Output pending queue 1, enqueue 64 entries.	Read	0
0x8400	31:0	Output complete queue 0, enqueue 1 entry.	Read	0
0x8404	31:0	Output complete queue 1, enqueue 1 entry.	Read	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x85FC	31:0	Output complete queue 127, enqueue 1 entry.	Read	0

Table 53: MAC Configuration and Status Registers (Continued)

Offset	Bit	Definition	Access	Reset
Memory Move Enqueue and Dequeue Operations				
0x0000 0x0004 . . . 0x00FC	31:0 31:0 . . . 31:0	Transfer pending queue 0, enqueue 1 entry. Transfer pending queue 0, enqueue 2 entries. . . . Transfer pending queue 0, enqueue 64 entries.	Read Read . . . Read	0 0 . . . 0
0x0100 0x0104 . . . 0x01FC	31:0 31:0 . . . 31:0	Transfer pending queue 1, enqueue 1 entry. Transfer pending queue 1, enqueue 2 entries. . . . Transfer pending queue 1, enqueue 64 entries.	Read Read . . . Read	0 0 . . . 0
. . .	. . .	. . .	. . .	. . .
0x0F00 0x0F04 . . . 0x0FFC	31:0 31:0 . . . 31:0	Transfer pending queue 16, dequeue 1 entry. Transfer pending queue 16, enqueue 2 entries. . . . Transfer pending queue 16, enqueue 64 entries.	Read Read . . . Read	0 0 . . . 0
0x1000 0x1004 . . . 0x10FC	31:0 31:0 . . . 31:0	Transfer complete queue 0, dequeue 1 entry. Transfer complete queue 0, dequeue 2 entries. . . . Transfer complete queue 0, dequeue 64 entries.	Read Read . . . Read	0 0 . . . 0
0x1100 0x1104 . . . 0x11FC	31:0 31:0 . . . 31:0	Transfer complete queue 1, dequeue 1 entry. Transfer complete queue 1, dequeue 2 entries. . . . Transfer complete queue 1, dequeue 64 entries.	Read Read . . . Read	0 0 . . . 0
. . .	. . .	. . .	. . .	. . .
0x0F00 0x0F04 . . . 0x0FFC	31:0 31:0 . . . 31:0	Transfer complete queue 16, dequeue 1 entry. Transfer complete queue 16, dequeue 2 entries. . . . Transfer complete queue 16, dequeue 64 entries.	Read Read . . . Read	0 0 . . . 0
Ethernet and PCI-X Mapper Configuration and Status Registers				
0x8680	31:16	Maximum instruction count.	Read/Write	0
0x8680	7:0	Interface clock divider.	Read/Write	0
0x8684	11:0	Mapper instruction store deposit/examine address.	Read/Write	0
0x8688	18:0	Mapper instruction store deposit/examine data.	Read/Write	0

11.1. Ethernet Media Access Controller Overview

The Stream Processor provides three (3) Media Access Controllers (MACs), each connected to a dedicated DMA Controller. This chapter describes the operation of the MACs. The DMA operation is described separately in Chapter 10.

Ethernet is a widely used packet communications protocol. Its operation is specified by the IEEE 802.3 series of documents. In particular, the MACs in the Stream Processor are compliant with 802.3, 802.3u, 802.3x, 802.3z, and 802.3ac. They can be connected to external 10/100/1000 Mbps PHYs through Gigabit Media Independent Interfaces (GMII).

The Ethernet Media Access Controller provides the following:

- 10/100 Mbps full- and half-duplex operation.
- 1000 Mbps full-duplex operation.
- GMII/MII interface with PHY management.
- Unbounded frame size.
- Automatic pad and CRC appending.
- Configurable deference.
- PAUSE frame flow control.
- Error condition signalling.
- Metering and statistics data.

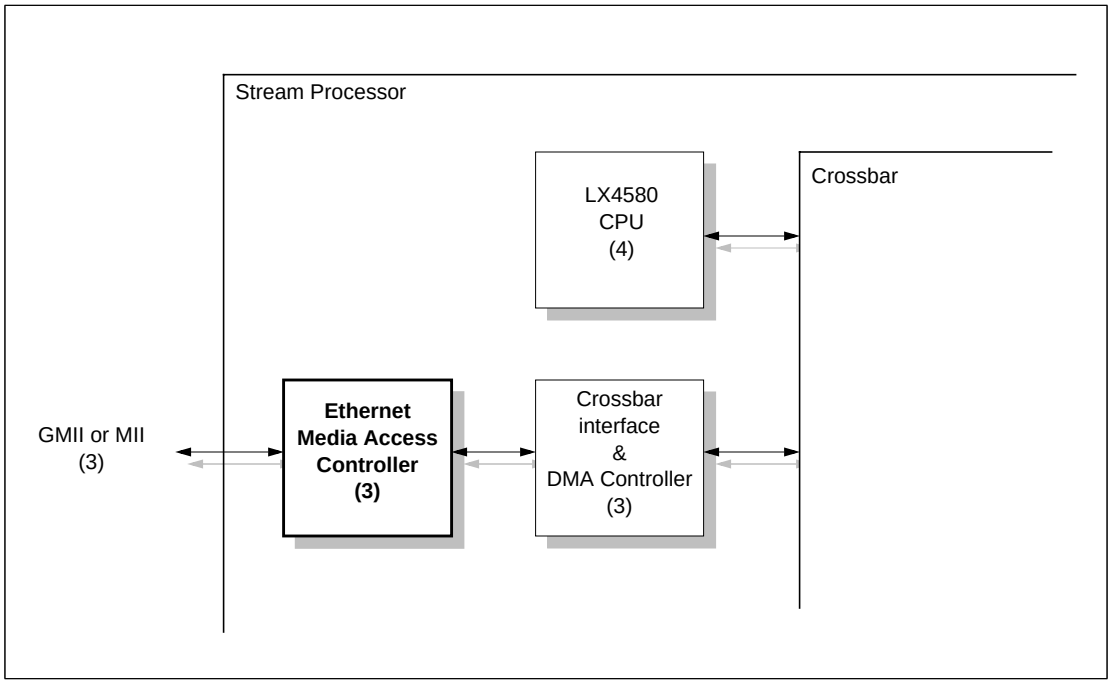

Figure 28: Ethernet Media Access Controller Connectivity

11.2. Gigabit Media Independent Interface (GMII)

Each MAC can be connected to an external PHY using a standard GMII/MII interface augmented with two optional clock enable pins from the PHY: RXCEN and TXCEN.

Table 54: GMII External Interface

Name	Pins	Direction	Description
COL	1	input	Collision detect from PHY.
CRS	1	input	Carrier sense from PHY.
MDC	1	output	Management clock to PHY.
MDIO	1	inout	Management data to/from PHY.
RX_CLK	1	input	Receive clock from PHY.
RX_DV	1	input	Receive data valid from PHY.
RX_ER	1	input	Receive error control from PHY.
RXCEN	1	input	Receive clock enable from PHY.
RXD[7:0]	8	input	Receive data from PHY.
GTX_CLK	1	input	Transmit reference clock.
TX_CLK	1	output	Transmit clock to PHY.
TX_EN	1	output	Transmit data enable to PHY.
TX_ER	1	output	Transmit error control to PHY.
TXCEN	1	input	Transmit clock enable from PHY.
TXD[7:0]	8	output	Transmit data to PHY.

11.3. Error Signalling and Statistics Reporting

After the reception or transmission of each frame, the MAC generates a status vector containing error and statistics information, including the link condition, deference delay, frame length consistency, and payload integrity. The DMA Controller attached to the MAC saves this information for later use by software.

Table 55: Receive Status Vector

Bit	Definition
32	Frame Truncated
31	Long Frame
30	VLAN Tag Detected
29	Unsupported Opcode
28	PAUSE Control Frame
27	Control Frame
26	Dribble Nibble

Table 55: Receive Status Vector (Continued)

Bit	Definition
25	Broadcast Frame
24	Multicast Frame
23	OK
22	Length Out Of Range
21	Length Check Error
20	CRC Error
19	Code Error
18	False Carrier
17	Short Frame
16	Previous Packet Dropped
15:0	Byte Count

Table 56: Transmit Status Vector

Bit	Definition
51	VLAN Tagged Frame
50	Back-pressure Applied
49	PAUSE Control Frame
48	Control Frame
47:32	Total Bytes Transmitted
31	Under-run
30	Long Frame
29	Late Collision
28	Maximum Collisions
27	Excessive Defer
26	Packet Defer
25	Broadcast Frame
24	Multicast Frame
23	Done

Table 56: Transmit Status Vector (Continued)

Bit	Definition
22	Length Out Of Range
21	Length Check Error
20	CRC Error
19:16	Collision Count
15:0	Byte Count

11.4. Registers

All MAC capabilities are accessed through registers that are mapped within the in a 64 KByte address space that is dedicated to each DMA controller. This region also provides access to other DMA functions, as summarized in Table 53 on page 117.

Table 57 summarizes the MAC configuration and status registers. The system address of each register is determined by adding the offset to the applicable base value defined in Section 5.5.

Table 57: MAC Configuration and Status Registers

Offset	Bit	Definition	Access	Init
MAC Configuration and Status Registers				
0x8600	31	Soft reset. Resets all of the MAC except the host interface.	Read/Write	1
0x8600	30	Simulation reset.	Read/Write	0
0x8600	19	RX control reset.	Read/Write	0
0x8600	18	TX control reset.	Read/Write	0
0x8600	17	RX data reset.	Read/Write	0
0x8600	16	TX data reset.	Read/Write	0
0x8600	8	Loop back.	Read/Write	0
0x8600	5	RX flow control enable.	Read/Write	0
0x8600	4	TX flow control enable.	Read/Write	0
0x8600	3	Synchronized RX enable.	Read	0
0x8600	2	RX enable.	Read/Write	0
0x8600	1	Synchronized TX enable.	Read	0
0x8600	0	TX enable.	Read/Write	0
0x8604	15:12	Preamble length.	Read/Write	7
0x8604	9:8	Interface mode. 1 = 10Mbps/100Mbps. 2 = 1000Mbps.	Read/Write	0
0x8604	5	Maximum frame disable.	Read/Write	0

Table 57: MAC Configuration and Status Registers (Continued)

Offset	Bit	Definition	Access	Init
0x8604	4	Length field check enable.	Read/Write	0
0x8604	2	Pad and CRC enable.	Read/Write	0
0x8604	1	CRC enable.	Read/Write	0
0x8604	0	Full-duplex.	Read/Write	1
0x8608	30:24	IPGR1 carrier window.	Read/Write	64
0x8608	22:16	IPGR2 carrier window.	Read/Write	96
0x8608	15:8	Minimum inter-frame gap.	Read/Write	80
0x8608	6:0	Minimum inter-packet gap.	Read/Write	96
0x860C	23:20	Alternate back-off limit.	Read/Write	10
0x860C	19	Alternate back-off enable.	Read/Write	0
0x860C	18	Back-pressure no back-off. Retransmit immediately during back-pressure.	Read/Write	0
0x860C	17	No Back-off. Retransmit immediately after collision.	Read/Write	0
0x860C	16	Maximum defer disable.	Read/Write	1
0x860C	15:12	Maximum defer limit.	Read/Write	15
0x860C	9:0	Collision window.	Read/Write	55
0x8610	15:0	Maximum frame length.	Read/Write	1536
0x8620	31	Management reset.	Read/Write	0
0x8620	5	Multi-PHY scan enable. Read round-robin from 32 attached PHYs.	Read/Write	0
0x8620	4	Preamble suppression. IEEE 802.3/22.2.4.4.2.	Read/Write	0
0x8620	2:0	Management clock select. 0 = 31.25Mhz 1 = 31.25Mhz 2 = 20.83Mhz 3 = 15.63Mhz 4 = 12.50Mhz 5 = 8.93Mhz 6 = 6.25Mhz 7 = 4.46Mhz	Read/Write	0
0x8624	1	Scan cycle. Execute continuous management read from PHY.	Read/Write	0
0x8624	0	Read cycle. Execute single management read from PHY.	Read/Write	0
0x8628	12:8	PHY address.	Read/Write	0
0x8628	4:0	Register address.	Read/Write	0

Table 57: MAC Configuration and Status Registers (Continued)

Offset	Bit	Definition	Access	Init
0x862C	15:0	Register write data	Write	0
0x8630	15:0	Register read data.	Read	0
0x8634	2	Register data not valid.	Read	0
0x8634	1	Scan cycle in progress.	Read	0
0x8634	0	PHY access in progress.	Read	0
0x8638	31	Interface reset. Reset only host interface.	Read/Write	0
0x8638	27	TBI mode.	Read/Write	0
0x8638	26	Half-duplex GMII mode.	Read/Write	0
0x8638	25	Half-duplex MII mode.	Read/Write	0
0x8638	24	PHY mode.	Read/Write	0
0x8638	23	MII reset.	Read/Write	0
0x8638	16	Speed select. 0 = 10Mbps 1 = 100Mbps	Read/Write	0
0x8638	15	Cipher reset.	Read/Write	0
0x8638	10	Force transmit quiet.	Read/Write	0
0x8638	9	Cipher disable.	Read/Write	0
0x8638	8	Link fail disable.	Read/Write	0
0x8638	7	GPSI reset.	Read/Write	0
0x8638	0	Jabber protection enable.	Read/Write	0
0x863C	9	Excessively deferring.	Read	0
0x863C	8	Misconfigured interface.	Read	0
0x863C	7	Jabber detected.	Read	0
0x863C	6	Link OK.	Read	0
0x863C	5	Full-duplex.	Read	0
0x863C	4	Speed setting. 0 = 10Mbps 1 = 100Mbps	Read	0
0x863C	3	Link fail.	Read	0
0x863C	2	Loss of carrier.	Read	0
0x863C	1	SQE error.	Read	0
0x863C	0	Jabber detected.	Read	0
0x8640	31:24	Station address octet 1.	Read/Write	0

Table 57: MAC Configuration and Status Registers (Continued)

Offset	Bit	Definition	Access	Init
0x8640	23:16	Station address octet 2.	Read/Write	0
0x8640	15:8	Station address octet 3.	Read/Write	0
0x8640	7:0	Station address octet 4.	Read/Write	0
0x8644	31:24	Station address octet 5.	Read/Write	0
0x8644	23:16	Station address octet 6.	Read/Write	0

12.1. PCI-X Bridge Overview

The Stream Processor's PCI-X bridge allows a standard connection to be made from the Stream Processor to other devices in the system. The bridge includes a dedicated DMA Controller. This chapter describes the operation of the PCI-X bridge. The DMA programming architecture is described separately in Chapter 10.

The PCI-X Bridge provides the following:

- 133 MHz, 32-bit operation.
- 4.2 Gbps bandwidth.
- TBD configurable address windows.
- Internal or external arbiter.
- As bus master, supports
  - Access from LX4580 CPUs to PCI-X bus devices.
  - Access from dedicated PCI-X DMA controller to PCI-X bus devices.
  - Non-coherent line reads and writes to PCI-X memory.
  - Non-coherent sub-line (I/O) reads and writes.
  - PCI-X configuration cycles.
- As a bus target, supports
  - Coherent line reads and writes to the Stream Processor's attached SDRAM.
  - Non-coherent sub-line (I/O) reads and writes to the Stream Processor's UART, I^2C and Generic I/O modules.
  - Non-coherent line reads and writes to the Stream Processor's Generic I/O module.
  - PCI-X configuration cycles.

Devices on the PCI-X bus may signal interrupts to the LX4580 CPUs over the Stream Processor's external level-sensitive interrupt inputs (See Section 4.2.1).

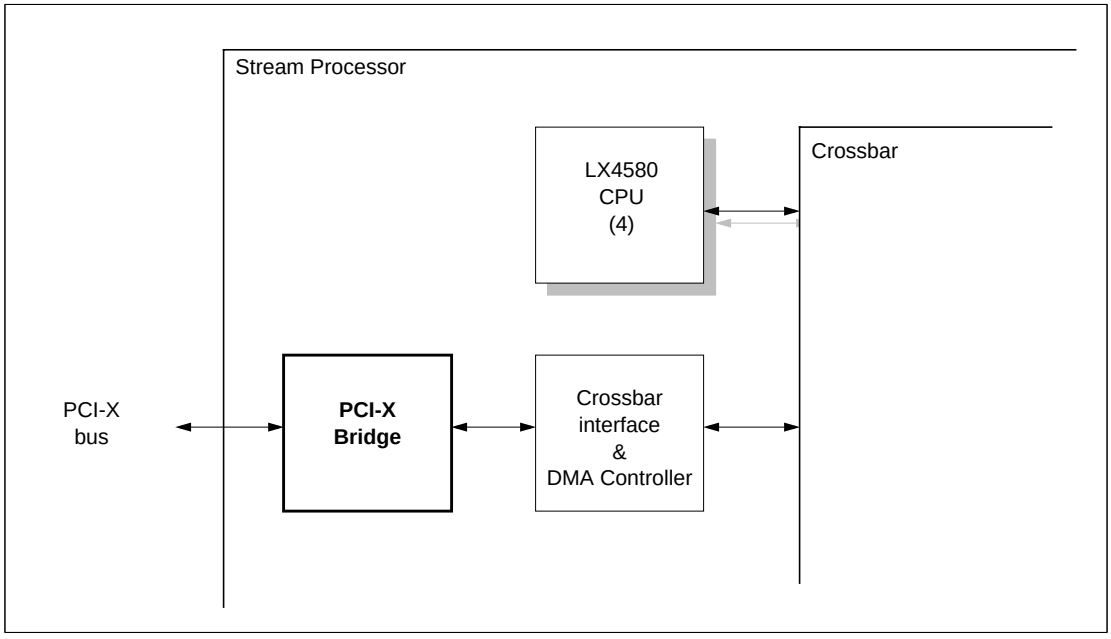

Figure 29: Overview of PCI-X Bridge

12.2. PCI-X Interface

Table 58: PCI-X Interface

Name	Pins	Direction	Description
PCIX_CLK	1	input	Clock
PCIX_PAR	1	inout	Parity.
PCIX_RST_N	1	input	Reset.
PCIX_AD[31:0]	32	inout	Address or data.
PCIX_CBE_N[3:0]	4	inout	Command and byte enable flags.
PCIX_FRAME_N	1	inout	Frame.
PCIX_IRDY_N	1	inout	Initiator ready.
PCIX_TRDY_N	1	inout	Target ready.
PCIX_STOP_N	1	inout	Stop.
PCIX_DEVSEL_N	1	inout	Device select.
PCIX_IDSEL	1	input	Initialization device select.
PCIX_LOCK_N	1	inout	Lock target.
PCIX_PERR_N	1	inout	Parity error.
PCIX_SERR_N	1	inout	System error.
PCIX_REQ_N	1	output	Stream Processor's request to external arbiter.
PCIX_GNT_N	1	input	Grant from external arbiter.
PCIX_REQIN_N[2:0]	3	input	Requests from external PCI devices to Stream Processor's internal arbiter.
PCIX_GNTOUT_N[2:0]	3	output	Grants to external PCI devices to Stream Processor's internal arbiter.

12.3. PCI-X Arbitration

The PCI-X bridge includes an internal arbiter that supports four masters - three external masters and the bridge itself. The internal arbiter may be disabled via a configuration register, in which case the bridge communicates with an external arbiter through a request/grant signal pair.

Details of the internal arbiter to be supplied.

12.4. PCI-X Master Operation

The PCI-X bridge operates as a PCI-X master. Internal transactions received from the bridge's dedicated DMA controller an the LX4580 CPUs (through the crossbar) are converted to PCI-X bus transactions as listed in Table 59. Coherency is *not* maintained for accesses made from the CPUs to PCI-X bus.

Table 59: Conversion of Internal Transactions to PCI-X Transactions

Internal Transaction	PCI-X Transaction
Line Read	Memory Read Block (64 bytes)
Line Write	Memory Write Block (64 bytes)
Sub-line Read (1, 2 or 4 bytes)	Memory Read Block (1, 2 or 4 bytes)
Sub-line Write (1, 2 or 4 bytes)	Memory Write (1, 2 or 4 bytes)

12.5. PCI-X Target Operation

The PCI-X bridge operates as a PCI-X target. PCI-X transactions that target the bridge are converted into crossbar transactions as listed in Table 60. These transactions may reference main memory and the I2C, UART and GIO interfaces. Access to main memory from the PCI-X bus is coherent with L1 and L2 caches.

Table 60: PCI-X Transactions to Crossbar Transactions

PCI-X Transaction	Crossbar Transaction
Memory Read Block	Line Read(s). As many line reads as required are issued to satisfy the size of the PCI-X transaction. The address and size of the Memory Read Block command need not be aligned to 64 bytes. Data is truncated as needed before it is sourced on the PCI-X bus.
Memory Write Block	Line Write(s). As many line writes as required are issued to satisfy the size of the PCI-X transaction. The address and size of the Memory Write Block command need not be aligned to 64 bytes. Write merging with current memory contents is performed as needed.
Memory Read Block (1, 2 or 4 bytes)	Sub-line Read (1, 2 or 4 bytes)
Memory Write (1, 2 or 4 bytes)	Sub-line Write (1, 2 or 4 bytes)

12.6. PCI-X Registers

To be supplied.

13.1. System Control Module Overview

The Stream Processor's System Control Module (SC) functions provide:

- I^2C interface to external devices.
- Cross-interrupt message reflection.
- Two general-purpose 32-bit countdown timers with interrupts.
- One RS-232 serial UART
- A generic I/O interface for up to 4 external devices.

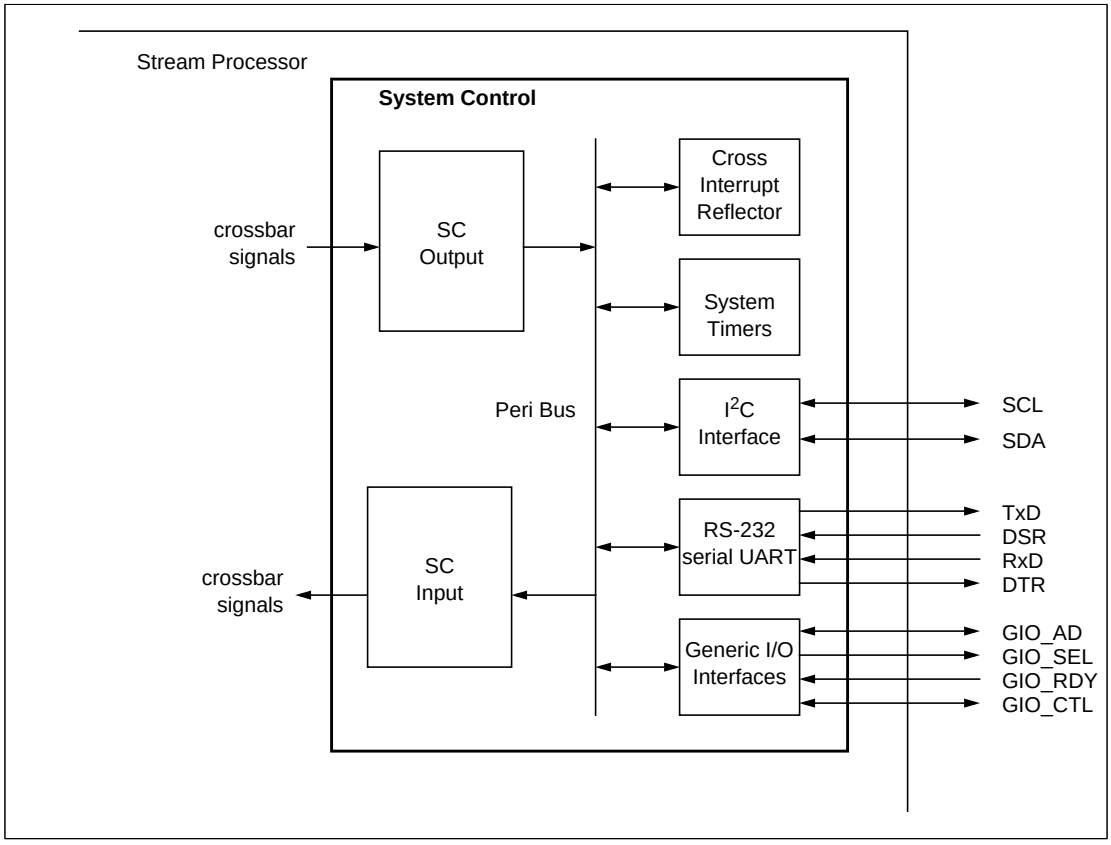

Figure 30: System Control Module Overview

The crossbar ports connect to a shared peripheral bus, the Peri Bus. The crossbar SC output module is the only initiator of peripheral bus transactions, and it initiates a transaction for each crossbar message received. The SC input module captures read data, error responses, or write acknowledgements from the peripherals and generates an appropriate crossbar response message. Only one transaction at a time is in progress in the SC, though multiple transactions may be buffered in the output module fifo.

The address driven on the Peri Bus is taken directly from the address specified in the header of the initiating SC output message. The address space required by each peripheral varies and for the generic I/O devices is configurable. The address range, within the 36-bit physical address space, to which the generic I/O devices are mapped may be configured by writing base address and address mask registers.

ATTENTION: When the address range registers for a generic I/O device are configured, the address mapping in each CPU's crossbar interface must be configured correspondingly so that peripheral accesses in the CPU generate messages destined for the System Control Module crossbar port.

13.2. Cross Interrupt Reflector

The cross interrupt reflector is mapped to the physical address range starting at IRR_Base. See Section 4.3.

13.3. System Timers

There are two system timers in the System Control Module block of the Stream Processor. One is driven by the CPU clock, the other is driven by an external real-time clock input. The timers can be used to generate two different interrupts at precisely specified numbers of clock ticks.

Each timer has 4 word length registers that are mapped to ranges starting at Timer_Base for the system clock timer and Timer_Base + 0x10 for the real time clock timer.

Name: Timer_Count.
Size: 32 bits.
Address: Timer_Base + 0x0.
Restrictions: Read Only.

31-0
Count

Field	Bits	Description	R/W	Reset
Count	31-0	Count value. The count increments once for every cycle of the clock that drives the counter. The count can neither be written nor reset. The count wraps to zero when it overflows.	R	0

Name: Timer_Config
Size: 32 bits.
Address: Timer_Base + 0x4.

31-16	15-12	11-8	7-4	3	2	1	0
reserved	TGTid1	TGTid0	reserved	M1	M0	P1	P0

Field	Bits	Description	R/W	Reset
reserved	31-16	Must be zero.	R/W	0
TGTid1	15-12	Target global thread ID for interrupt 1.		
TGTid0	11-8	Target global thread ID for interrupt 0.		
reserved	7-4	Must be zero.	R/W	0
M1	3	Interrupt Mask 1. Interrupt 1 is enabled when set to 1.	R/W	0
M0	2	Interrupt Mask 0. Interrupt 0 is enabled when set to 1.	R/W	0
P1	1	Interrupt 1 Pending. This active-high signal indicates that match 1 has occurred. This bit is asserted regardless of the interrupt mask bits. Writing a 1 to this bit clears the pending interrupt.	R/W	0
P0	0	Interrupt 0 Pending. This active-high signal indicates that match 0 has occurred. This bit is asserted regardless of the interrupt mask bits. Writing a 1 to this bit clears the pending interrupt.	R/W	0

Name: Timer_Compare_0.
Size: 32 bits.
Address: Timer_Base + 0x8.

31-0
Compare0

Field	Bits	Description	R/W	Reset
Compare0	31-0	Compare value 0. When the count reaches this value, the interrupt pending 0 bit is asserted. If the interrupt 0 mask is not asserted, an interrupt signal will be asserted.	R/W	0

Name: Timer_Compare_1.
Size: 32 bits.
Address: Timer_Base + 0x8.

31-0
Compare1

Field	Bits	Description	R/W	Reset
Compare1	31-0	Compare value 1. When the count reaches this value, the interrupt pending 1 bit is asserted. If the interrupt 1 mask is not asserted, an interrupt signal will be asserted.	R/W	0

The clock-tick count is read-only. The compare registers can be read and written. The count increments on every clock cycle. When the count value equals the value in a compare register the corresponding interrupt will be set as pending. The timer signals the interrupt using an Interrupt (IN) crossbar message to the target CPU and context indicated by the TGTid1 or TGTid0 fields of the TimerConfig register. The state of interrupts can be polled by reading from the status/control bit register at offset 4. Interrupts are cleared by writing a 1 to their corresponding bits.

Implementation note: To generate an interrupt at a precise regular interval, the exact number of desired clock-ticks between interrupts should be added to the compare register in the interrupt service routine. Reading the count register, adding to it, and writing to the compare register is an imprecise method, and should be done only during system initialization. To calculate the number of desired clock ticks between interrupts, divide the timer clock source frequency by the desired interrupt frequency.

13.4. I^2C Interface

The I^2C interface peripheral is an I^2C bus master and slave. It conforms to the Philips I^2C specification version 2.1, including multi-master arbitration. The interface supports 10 bit addressing in standard (100 Kbps) and fast (400 Kbps) modes.

Additional details to be supplied.

13.5. Test And Set

The Test And Set register is implemented in the System Control Module. See Section 3.4.1 for description of this register.

13.6. RS-232 Serial UART

The Stream Processor includes a simple serial UART. It supports RS-232 communication with 1 start bit, 8 data bits, 1 stop bit, and no parity. The UART supports a range of baud rates. It defaults to 9600 baud at reset.

The UART has a data and a configuration word-sized memory-mapped register. Both respond only to word-sized accesses. For a data register access, only the lower 8-bits of data are valid. A write to the data register initiates the transmission of a character. A read from the data register returns the next character in the receive buffer. An interrupt line, output from the UART, is asserted whenever one or more characters are queued in the receive buffer.

Name: UART_Data.
Size: 32 bits.
Address: SCBase + 0x4.

31-8	7-0
reserved	UARTData

Field	Bits	Description	R/W	Reset
reserved	31-8	unused	R/W	0
UART data	7-0	Data written is transmitted by the UART. Data in the UART receive buffer is read.	R/W	0

Name: UART_Status_Config.
Size: 32 bits.
Address: SCBase + 0x0.
SW Init: To use UART interrupts, the interrupt enable fields of this register must be written.

31-28	27	26	25	24	23-20	19-16	15-0
reserved	TxIntEn	TxRdy	RxIntEn	RxRdy	reserved	TGTid	BaudRate

Field	Bits	Description	R/W	Reset
reserved	31-28	Must be zero.	R/W	0
TxIntEn	27	Transmit Interrupt Enable. Active high.	R/W	0
TxRdy	26	Transmit Ready. Active high. A character may be transmitted over the UART. Software should poll this bit before writing data to the UART.	R	1
RxIntEn	25	Receive Interrupt Enable. Active high.	R/W	0
RxRdy	24	Receive Data pending. Active high. There is receive data in the receive buffer.	R	0
reserved	23-20	Must be zero.	R/W	0
TGTid	19-16	Target global thread ID for interrupt.		
BaudRate	15-0	Baud rate. See table below for baud rate encodings.	W	0

The 16 least significant bits of a write to the configuration register define the baud rate.

baud rate	config (hex)	config (binary)
614400	0x000f	0000000000001111
307200	0x001e	0000000000011110
153600	0x003c	0000000000111100
76800	0x0078	0000000001111000
57600	0x00a0	0000000010100000
38400	0x00f0	0000000011110000
28800	0x0140	0000000101000000
19200	0x01e0	0000000111100000
14400	0x0280	0000001010000000
9600	0x03c0	0000001111000000
4800	0x0780	0000011110000000
2400	0x0f00	0000111100000000
1200	0x1e00	0001111000000000
600	0x3c00	0011110000000000
300	0x7800	0111100000000000
150	0xf000	1111000000000000

If the receive interrupt enable flag is set, then an Interrupt (IN) message will be generated whenever a character is waiting in the receive buffer. If the transmit interrupt enable flag is set, then an interrupt will be generated whenever the transmit buffer is empty and ready for a new character. All configuration register accesses must be word sized.

13.7. Generic I/O Interface

4 I/O devices are accessible through the generic I/O interface within the System Control Module Their address ranges are configurable. By default, device zero responds to a 4 megabytes range starting at physical address 0x1fc00000 and devices 1-3 are disabled. For a description of the generic I/O interface, see Chapter 14.

14.1. Generic I/O Interface Overview

The Generic I/O Interface (GIO) provides access to off-chip low bandwidth asynchronous devices. Software running on the Stream Processor accesses GIO devices through memory mapped spaces. These accesses may be cacheable in L1 instruction and data caches, depending on the nature of the GIO device. However, cache coherency is not maintained for GIO accesses. Applications requiring cache coherency for GIO accesses must mange it in software.

The Generic Input/Output Interface includes the following features.

- Decode of four distinct GIO devices with independently configurable timing and protocols.
- Conversion between crossbar transactions and external I/O transactions.
- Multiplexed address/data bus mode, with 32-bit address and 8- 16- or 32-bit data.
- Shared address/data bus mode, with 24-bit address, 8-bit data.
- Single beat and 2, 4, 8, 16, 32, 64, and unlimited beat burst transactions.
- Data handshake and pure timing modes.

The GIO interface accepts transaction requests from the Stream Processor's internal crossbar. The address from each request is decoded to determine which of four GIO devices is targeted. Configurable timing and control parameters determine how crossbar transactions are converted into external GIO transactions. There is a separate set of configuration parameters for each target device.

The active state (high or low) of all signals except GIO_AD[31:0] is configurable. The GIO_AD[31:0], GIO_SEL[3:0] and GIO_RDY signals provide dedicated protocol functions. The GIO_CTL[3:0] signals can be configured to support a wide variety of interface methods.

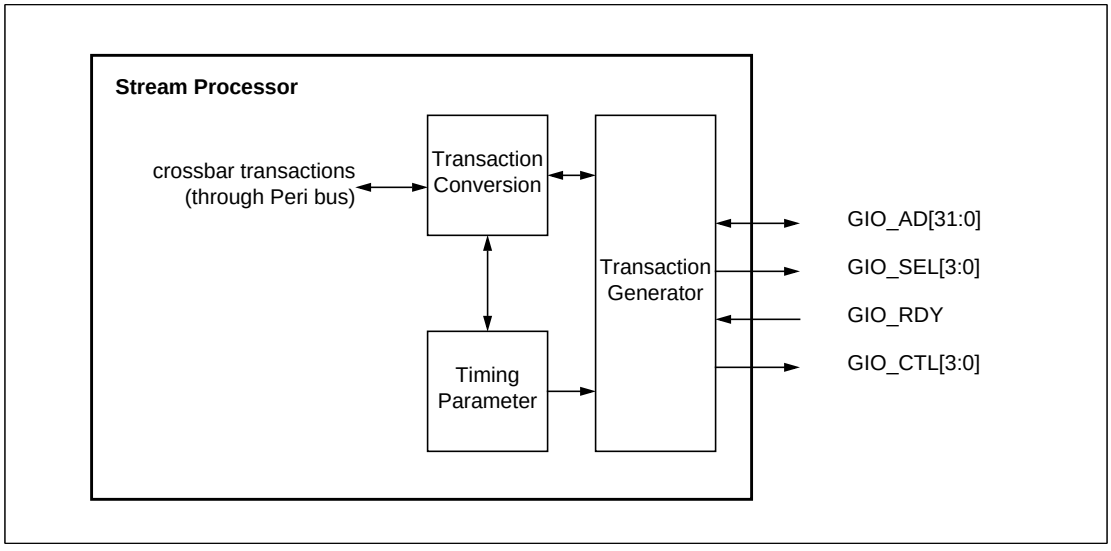

Figure 31: Generic I/O Interface

Throughout this chapter, the variable *m* represents a Generic I/O Interface device number and the variable *n* represents a signal number within a bus.

14.2. Generic I/O Interface Signals and Timing

The generic I/O interface provides four independent GIO device select signals and a set of signals that are shared by all connected GIO devices. For each transaction, one of the select lines is asserted and the shared signals provide address, data, read/write control and handshaking for the transaction. The active level of all signals except AD[31:0] can also be set through the configuration registers.

Table 61: Generic I/O Signals

Signal Name	Direction	Description
GIO_SEL[3:0]	output	GIO device select lines. One of the these lines is asserted by the Stream Processor for the duration of a generic I/O transaction to identify the selected device.
GIO_AD[31:0]	inout	Address and data lines. These lines operate in multiplexed or non-multiplexed modes, depending how the generic I/O interface is configured to support the selected GIO device. For a multiplexed transaction, the Stream Processor drives the low order 32-bits of the physical address onto AD[31:0] during the transaction's address phase, and drives or receives data over AD[31:0] during the transaction's data phase. For a non-multiplexed transaction, the Stream Processor drives the low order 24 bits of the physical address onto AD[23:0] and drives or receives data over AD[31:24]. For clarity, the names A[23:0] and D[7:0] refer to the AD signals when they are used in the non-multiplexed mode.
GIO_RDY	input	Device Ready. Asserted by the selected GIO device to indicate that a read or write operation has completed. The Stream Processor observes this signal only if the generic I/O interface is configured to honor the data handshake for the selected device. Otherwise, the configured timing values are used to control the length of the transaction's data phase.
GIO_CTL[3:0]	output	Programmable control signals. Output enable. Asserted low by the Stream Processor to enable the selected GIO device's data output driver.

The timing for all signals is controlled through configuration registers, specific to each GIO device. Table 62 shows the supported timing parameters. All timing parameters are specified as a time period in units of multiples of a crossbar clock period.

Table 62: Generic I/O Timing Parameters

Parameter	Description
t_{Am1}	If a single beat transaction, duration that the address is driven. If a multiple beat transaction duration that the first address is driven. If a handshaking transaction, the duration that the address is driven after RDY is asserted.
t_{Am2}	Minimum duration that address is not driven prior to starting the next transaction.
t_{Am3}	If the transaction is four beats or more, duration that the address is driven for the second and subsequent beats, other than the last beat. Not used for single beat and two beat transactions.
t_{Am4}	If a multiple beat transaction, duration that the address is driven for the last beat. Not used for single beat transactions.
t_{Dm1}	For write transactions, delay to the start of driving the first data beat. Not used for read transactions.
t_{Dm2}	For write transactions, duration that each data beat is driven. Not used for read transactions.
t_{RSm}	For handshake reads, delay from assertion of RDY until the sampling of read data. For burst reads, delay from last address transition until the sampling of the last data. Otherwise, delay from the start of the transaction until the sampling of read data.
t_{SELm1}	Delay for the assertion of GIO_SEL[*m*].
t_{SELm2}	Delay for the de-assertion of GIO_SEL[*m*].
t_{CTLmn1}	Delay for the assertion of GIO_CTL[*n*].
t_{CTLmn2}	Delay for the de-assertion of GIO_CTL[*n*].
t_{CTLmn3}	Optional timing parameter for multi-beat transactions. Delay between consecutive assertions of GIO_CTL[*n*].

14.3. Generic I/O Configuration Overview

This section provides an overview of generic I/O configuration. The configuration registers are described in detail in Section 14.7.

The Stream Processor's generic I/O address space is configured with on-chip registers that identify the total generic I/O address space, and additional on-chip registers that identify the address space and timing parameters for up to four GIO devices. The address space of each device must fall within the total generic I/O address space. The total address space of each device may be 64 KBytes to 4 GBytes, and must be a power of two in size.

Applying reset to the Stream Processor configures Generic I/O device 0 to support a simple ROM located at physical address range 0x0_1fc0_0000 through 0x0_1fff_ffff. The remaining three devices are disabled following reset. The reset configuration allows processors to execute boot code from the ROM starting at

physical address 0x0_1fc0_0000. Boot code may then configure the generic I/O interface to support the specific GIO devices that are attached to the Stream Processor.

The address space for generic I/O is configured within each processor's address space decode logic, via the AS_GIOBase and AS_GIOTop registers. (See Section 8.7.) When processor makes requests to the crossbar, it decodes the physical address to identify the crossbar target, using the contents of these registers (and other address space decode registers). If the address falls within the range specified in the AS_GIOBase and AS_GIOTop registers, the crossbar request is directed to the generic I/O interface.

When a request reaches the GIO interface, decode logic within the GIO interface uses the address and the values in the GIO_Mask*m* and GIO_Addr*m* registers (*m*=0-3) to determine which GIO device is to be accessed. In the case of an overlap in the device address spaces, the lowest numbered device is selected. If there is no address match, the generic I/O interface sends an error reply to the requestor.

The GIO configuration registers provide the following information for each of the four decoded GIO devices.

- Timing parameters in units of the Stream Processor's crossbar clock (250 MHz).
- Signal polarity (except AD[31:0]).
- Read data sample time.
- Address/data mode (multiplexed or non-multiplex).
- Data signaling mode (handshake or pure timing).
- Data transfer size (1, 2 or 4 bytes).
- Supported data burst lengths (1, 2, 4, 8, 16, 32, 64, and unlimited beats).

14.4. Generic I/O Transaction Conversion

The GIO interface receives transaction requests from the CPUs over the crossbar, and converts each crossbar transaction to one or more transactions over the GIO pins. For example, the GIO interface converts a line read request from the crossbar into a series of read requests to the selected GIO device, gathers the read data to form a 64-byte line of data, and returns the line read result over the crossbar.

The conversion process follows these rules:

- The address from the original crossbar transaction determines which GIO device is accessed.
- An error response is generated if the request is to an address at which no device or configuration register is mapped.
- An error response is generated if the request is smaller than the device's configured size.
- The crossbar transaction is broken into a series of same-size GIO transactions.
- The largest size possible GIO transaction is used (including data bursting).

14.5. Generic I/O Transactions

This section describes all of the GIO transaction types:

- Non-multiplexed read.
- Non-multiplexed write.
- Multiplexed read.
- Multiplexed write.
- Non-multiplexed read with data handshake.
- Non-multiplexed write with data handshake.
- Multiplexed read with data handshake.
- Multiplexed write with data handshake.
- Non-multiplexed burst read.
- Non-multiplexed burst write.

Note that burst transactions with address/data multiplexing or data handshake are not supported. The length for burst transactions may be configured for two, four, eight, sixteen, thirty-two, sixty-four, and an unlimited number of beats. A representative burst length of four beats is shown in burst transaction timing diagrams.

To illustrate how GIO transactions work, this section uses a representative control protocol with address latch enable, read enable, write enable and output enable controls, as shown in Figure 32. For this application there is a mix of active high and active low signals. The GIO_CTL[3:0] outputs are configured to provide address latch enable, read enable, write enable and output enable signals for the GIO devices attached to the Stream Processor, as described in Table 63.

Other control protocols are possible. For example, instead of separate read enable and write enable controls, a strobe signal and read/write flag could be used.

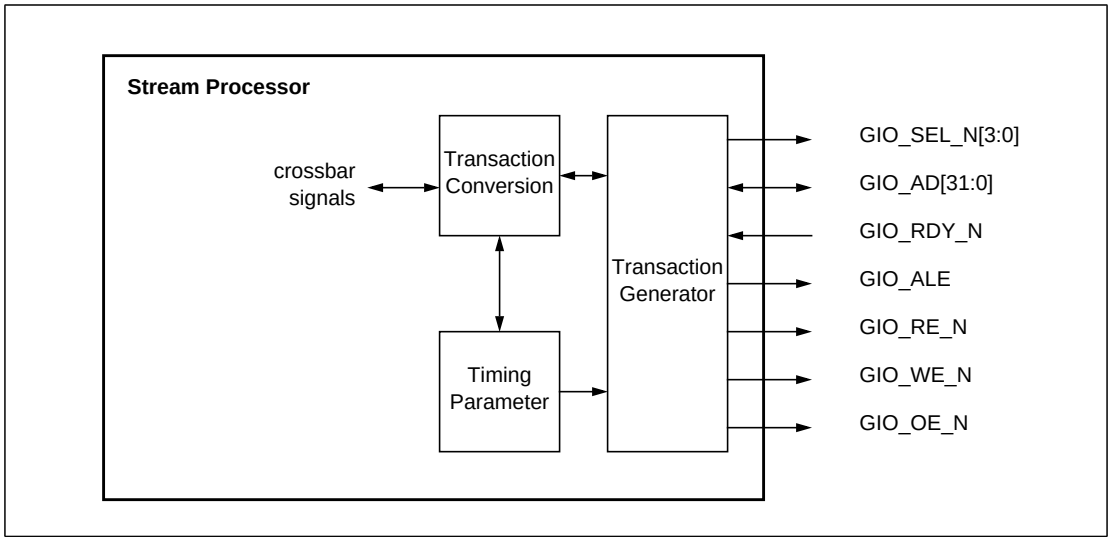

Figure 32: Example Generic I/O Interface Application

Table 63: Example GIO Application Signals

GIO Signal	Application	Direction	Description
GIO_SEL[3:0]	SEL_N[3:0]	output	GIO device select lines. One of the these lines is asserted (low) by the Stream Processor for the duration of a generic I/O transaction to identify the selected device.
GIO_AD[31:0]	AD[31:0]	inout	Address and data lines. These lines operate in multiplexed or non-multiplexed modes, depending on how the generic I/O interface is configured to support the selected device. For a multiplexed transaction, the Stream Processor drives the low order 32-bits of the physical address onto AD[31:0] during the transaction's address phase, and drives or receives data over AD[31:0] during the transaction's data phase. For a non-multiplexed transaction, the Stream Processor drives the low order 24 bits of the physical address onto AD[23:0] and drives or receives data over AD[31:24]. For clarity, the names A[23:0] and D[7:0] refer to the AD signals when they are used in the non-multiplexed mode.
GIO_CTL[0]	OE_N	output	Output enable. Asserted low by the Stream Processor to enable the selected GIO device's data output driver.
GIO_CTL[1]	ALE	output	Address latch enable. For multiplexed transactions, asserted high by the Stream Processor to signal a valid address.
GIO_CTL[2]	RE_N	output	Read enable. Asserted low by the Stream Processor to enable a read transaction for the selected GIO device.
GIO_CTL[3]	WE_N	output	Write enable. Asserted low by the Stream Processor to enable a write transaction for the selected GIO device.
GIO_RDY	RDY_N	input	Device Ready. Asserted low by the selected GIO device to indicate that a read or write operation has completed. The Stream Processor observes this signal only if the generic I/O interface is configured to honor the data handshake for the selected device. Otherwise, the configured timing values are used to control the length of the transaction's the data phase.

The timing diagrams indicate the relationships between the timing parameters that control the transition of signals sourced by the Stream Processor. Each timing parameter is relative to one of several possible reference points, depending upon the signal and GIO device's mode of operation.

- The transaction timing starts with the first transition on the address lines.
- The first transition on each other signals is timed from the start of the transaction.
- Except for handshaking, subsequent transitions are timed from the previous transition of

the signal.

- In the case of handshaking, some transitions are timed from the assertion of RDY_N.

The reference points employed for each transaction type are defined in the timing diagrams.

The use of the timing parameters for the example protocol is shown in Table 64. These parameters are for a representative GIO device. The device in this example is attached to GIO interface 0. For clarity, the *m* and *n* subscripts in Table 62 are omitted from the timing parameter names in the example application timing diagrams.

Table 64: Example GIO Application Parameters

GIO Parameter	Application	Description
t_{A01}, t_{A02}, t_{A03}, t_{A04}, t_{D01}, t_{D02}	t_{A1}, t_{A2}, t_{A3}, t_{A4}, t_{D1}, t_{D2}	Address and data timing. Used as described in Table 62.
t_{RS0}	t_{RS}	Delay for the sampling of read data.
t_{SEL01}	t_{SEL1}	Delay for the assertion of SEL_N[0].
t_{SEL02}	t_{SEL2}	Delay for the de-assertion of SEL_N[0].
t_{CTL001}	t_{OE1}	Delay for the assertion of OE_N.
t_{CTL002}	t_{OE1}	Delay for the assertion of OE_N.
t_{CTL003}	-	Disabled.
t_{CTL011}	t_{ALE1}	Delay for the assertion of ALE.
t_{CTL012}	t_{ALE2}	Delay for the de-assertion of ALE.
t_{CTL013}	-	Disabled.
t_{CTL021}	t_{RE1}	Delay for the assertion of RE_N.
t_{CTL022}	t_{RE2}	Delay for the de-assertion of RE_N.
t_{CTL023}	-	Disabled.
t_{CTL031}	t_{WE1}	Delay for the first assertion of WE_N.
t_{CTL032}	t_{WE2}	Delay for the de-assertion of WE_N.
t_{CTL033}	t_{WE3}	If a multiple beat transaction, delay for the second and subsequent assertions of WE_N. Not used for single beat transactions.

WARNING It is possible to set timing parameters that do not make sense or might even damage the components that are connected to the GIO interface. No checks are performed by the GIO interface to prevent such problems. It is the responsibility of the designer to choose reasonable timing parameters.

.

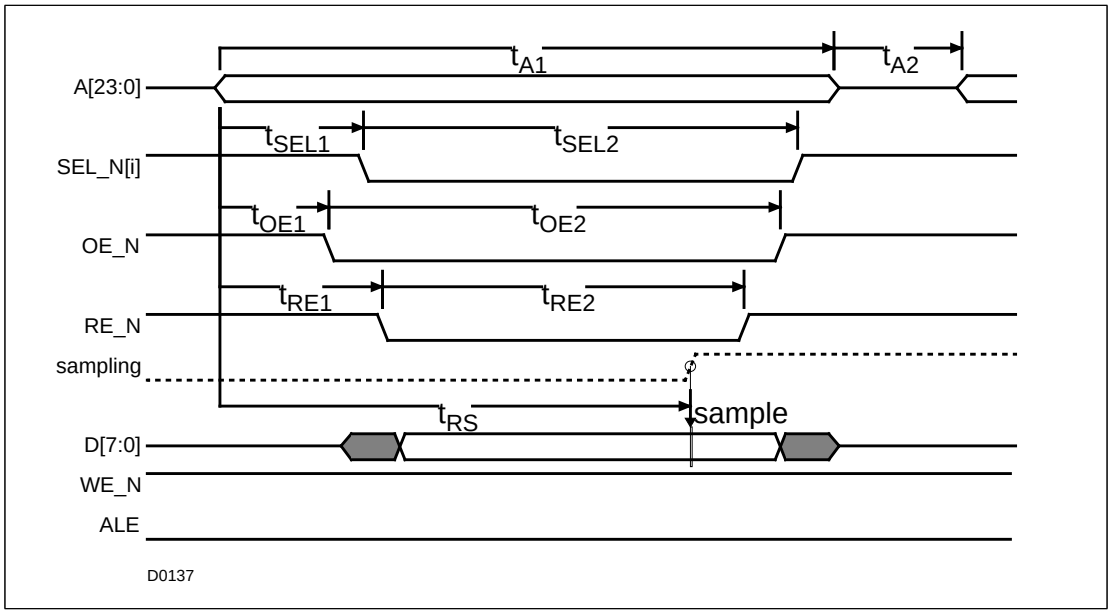

Figure 33: Generic I/O Simple Read

.

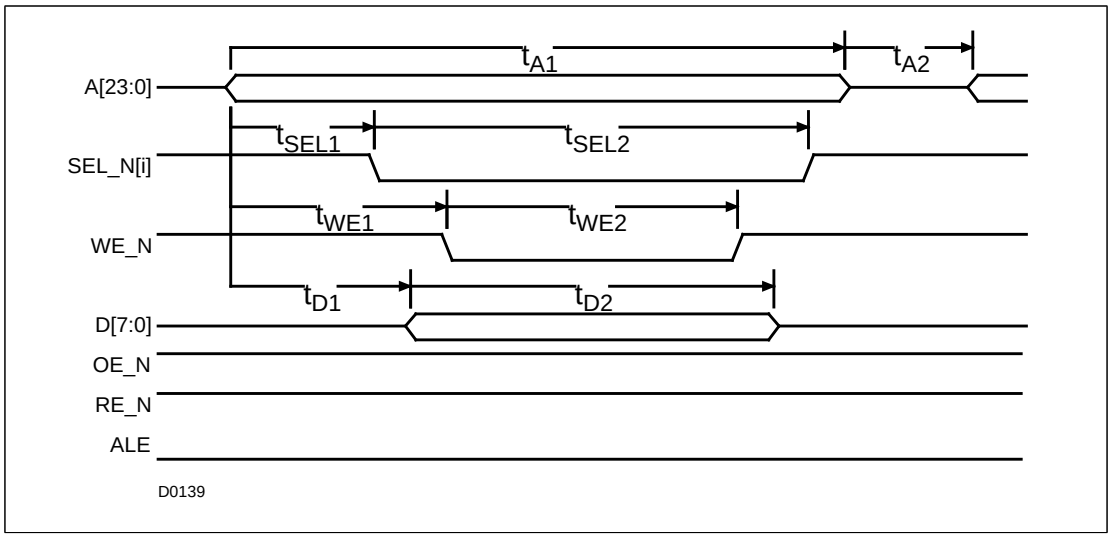

Figure 34: Generic I/O Simple Write

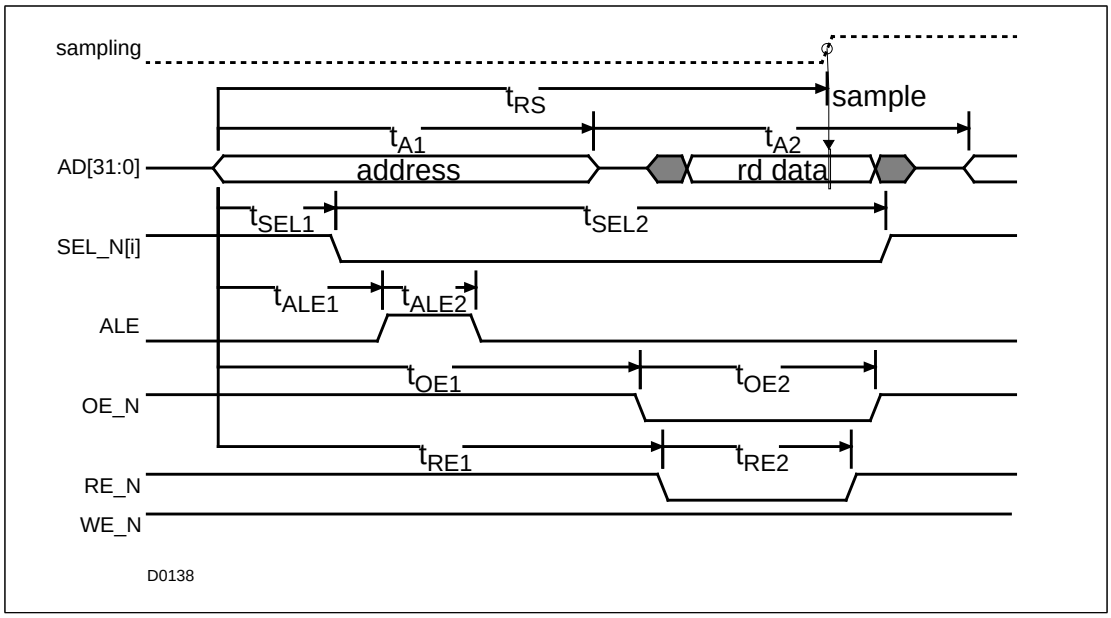

Figure 35: Generic I/O Multiplexed Read

.

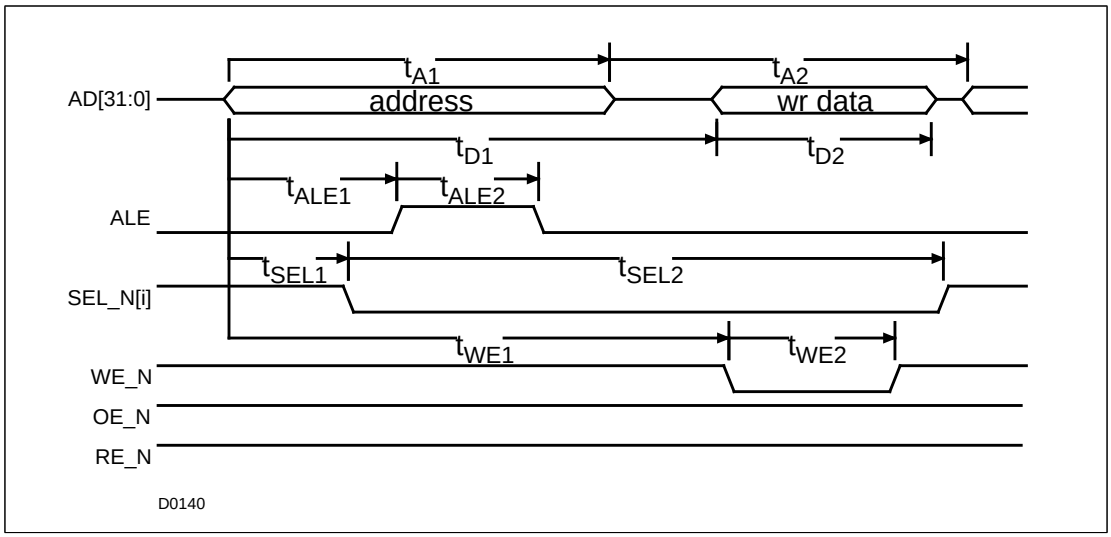

Figure 36: Generic I/O Multiplexed Write

.

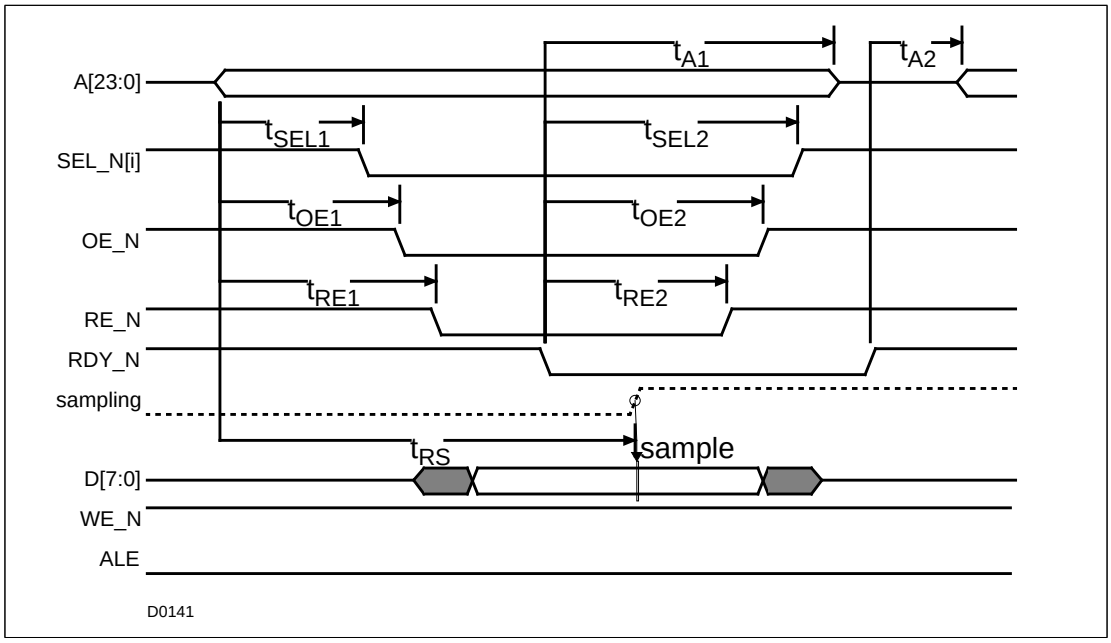

Figure 37: Generic I/O Read with Data Handshake

.

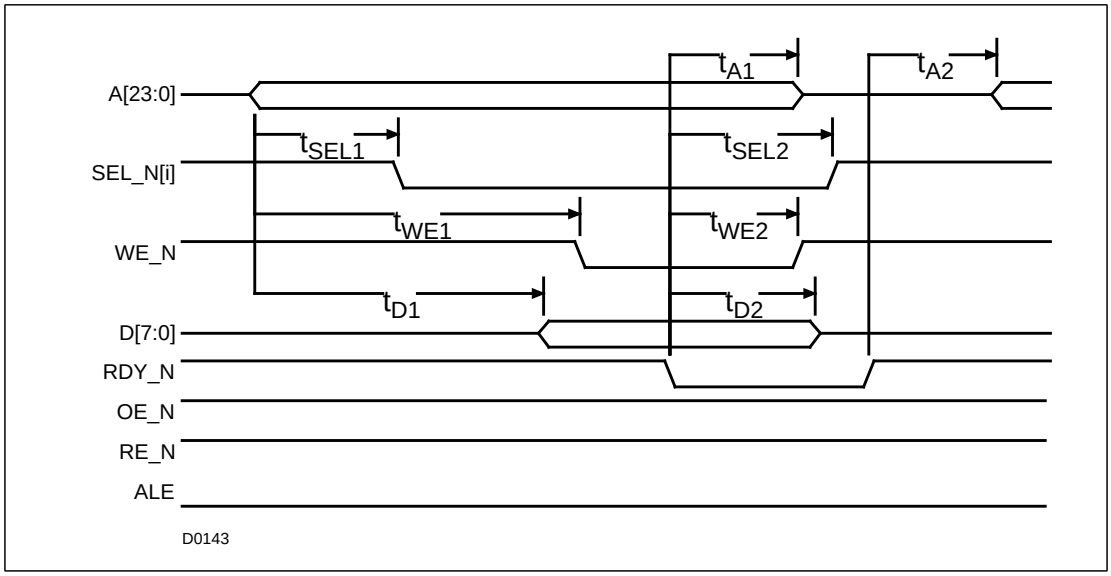

Figure 38: Generic I/O Write with Data Handshake

.

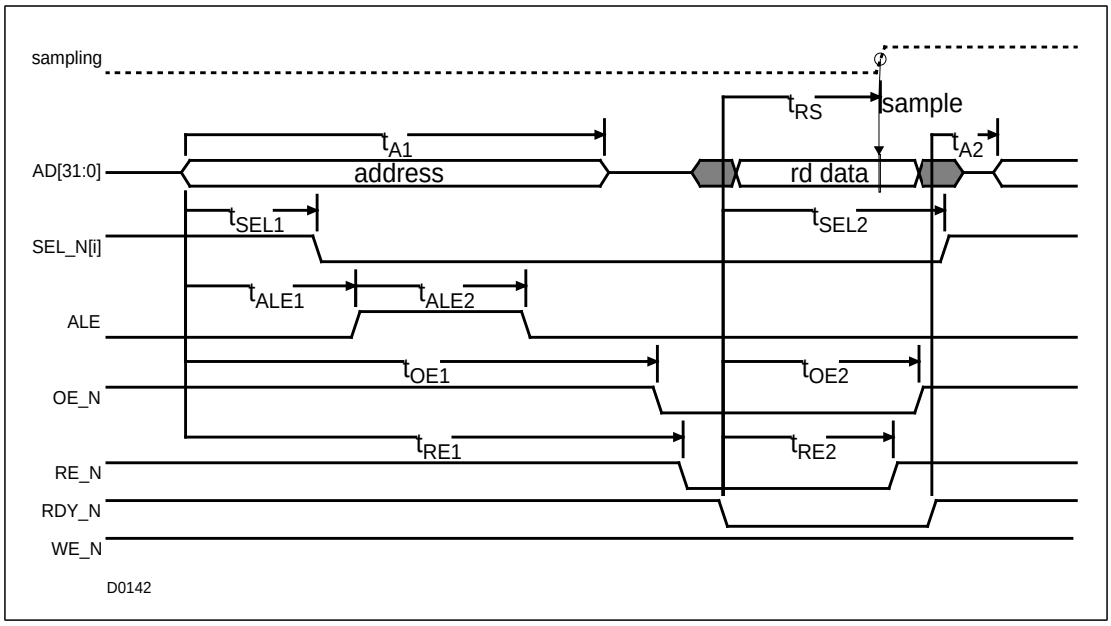

Figure 39: Generic I/O Multiplexed Read with Data Handshake

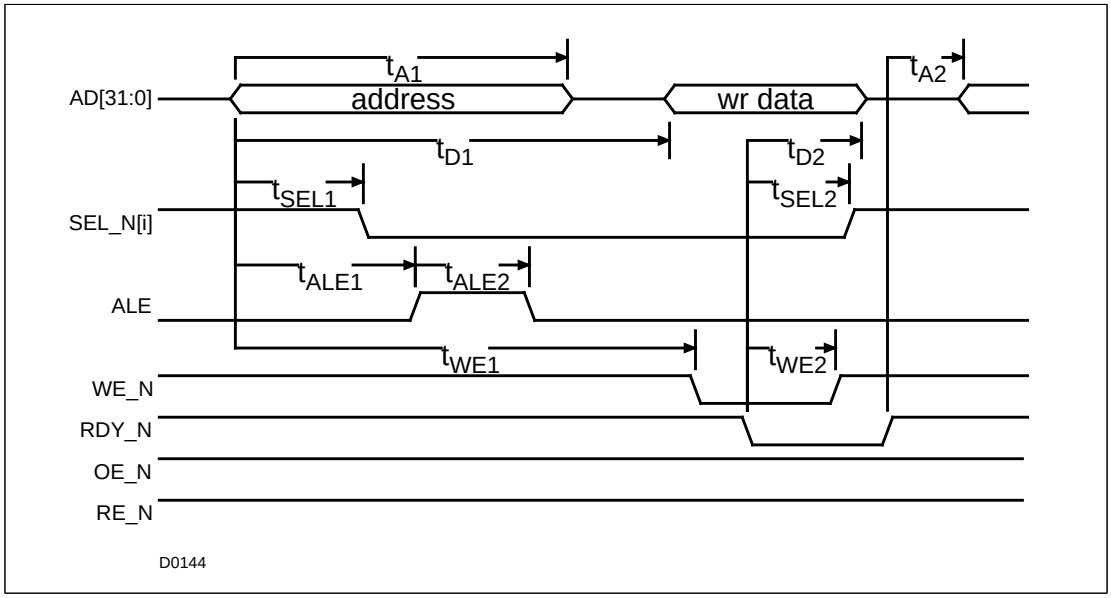

Figure 40: Generic I/O Multiplexed Write with Data Handshake

.

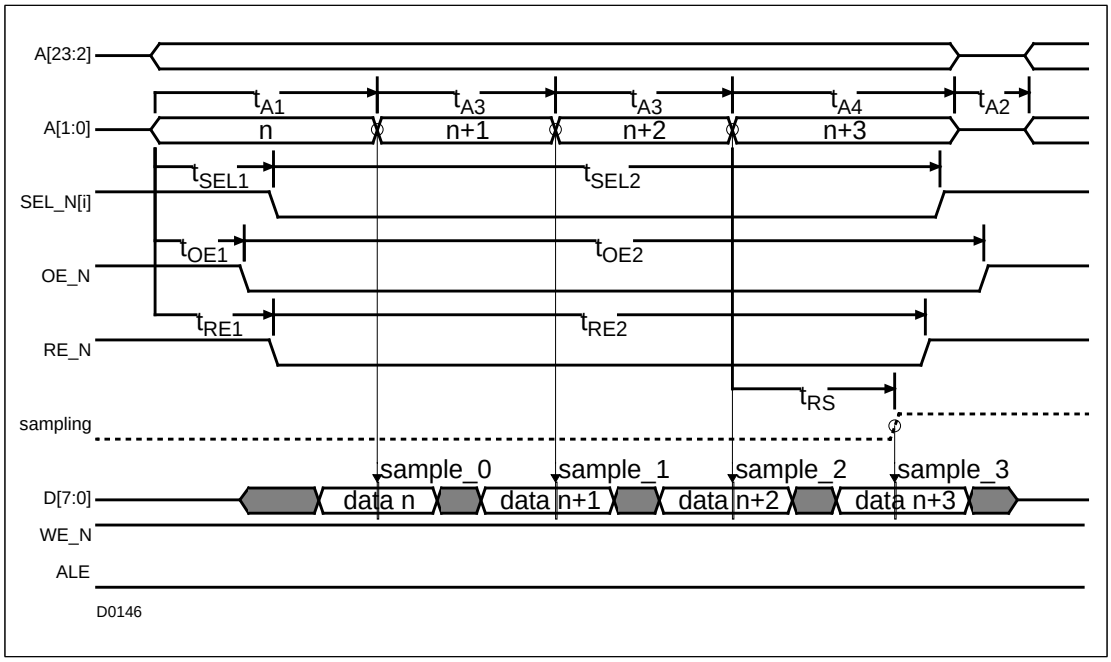

Figure 41: Generic I/O Burst Read

.

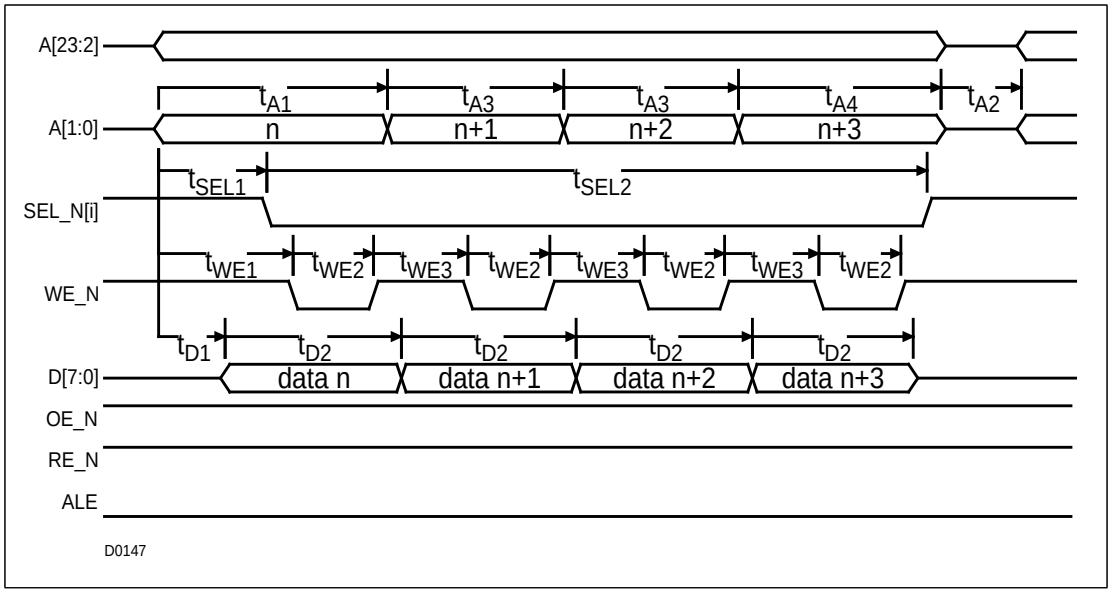

Figure 42: Generic I/O Burst Write

14.6. Errors and Error Reporting

The Generic I/O Interface responds with a bus error message if it receives a tri-byte access or an access that is narrower than the configured data width of the device. The Generic I/O Interface also responds with a bus error message if an access is made to an address that is not within the configured range of any of the devices or does not match a memory mapped configuration register.

14.7. Generic I/O Configuration Registers

The address space is used to access GIO devices is configured via registers in the CPU crossbar interface. See Sections 8.7.6 and 8.7.7.

The Generic I/O Interface also includes configuration registers for each device, as described in Table 65.

Table 65: Summary of GIO Config Registers for Each Device

Register name	Relative Address	Description
GIO_Cfg*m*	0x00	Master device configuration fields
GIO_t_{Am1}	0x04	First transaction address assertion
GIO_t_{Am2}	0x08	Transaction address de-assertion
GIO_t_{Am3}	0x0C	Second transaction beat address assertion
GIO_t_{Am4}	0x10	Last transaction beat address assertion
GIO_t_{Dm1}	0x14	Transaction data delay
GIO_t_{Dm2}	0x18	Transaction data assertion
GIO_t_{RSm}	0x1C	Transaction data read sampling
GIO_t_{SELm1}	0x20	Device select delay
GIO_t_{SELm2}	0x24	Device select duration
GIO_t_{CTLm01}	0x28	Device control signal 0 delay
GIO_t_{CTLm02}	0x2C	Device control signal 0 assertion
GIO_t_{CTLm03}	0x30	Device control signal 0 de-assertion
GIO_t_{CTLm11}	0x34	Device control signal 1 delay
GIO_t_{CTLm12}	0x38	Device control signal 1 assertion
GIO_t_{CTLm13}	0x3C	Device control signal 1 de-assertion
GIO_t_{CTLm21}	0x40	Device control signal 2 delay
GIO_t_{CTLm22}	0x44	Device control signal 2 assertion

Table 65: Summary of GIO Config Registers for Each Device (Continued)

Register name	Relative Address	Description
GIO_t_{CTLm23}	0x48	Device control signal 2 de-assertion
GIO_t_{CTLm31}	0x4C	Device control signal 3 delay
GIO_t_{CTLm32}	0x50	Device control signal 3 assertion
GIO_t_{CTLm33}	0x54	Device control signal 3 de-assertion

Name: GIO_Cfg*m*.
Size: 32 bits.
Address: GIOBase + 0x58**m*.

31-9	8	7	6-5	4-2	1	0
0	RDY Sense	Enable	Width	Burst	Handshake	Multiplexed

Field	Bits	Description	R/W	Reset
RDY Sense	8	Active state of RDY from device *m*. This field is only valid for handshaking devices.		
Enable	7	GIO device interface *m* is enabled.	R/W	1 for device 0 (*m*=0) 0 for others
Width	6-5	The data width of the connected device 00 = 8 bits 01 = 16 bits 10 = 32 bits 11 = illegal This field is only valid for multiplexed devices.	R/W	00
Burst	4-2	Maximum burst size supported by the connected device 000 = single cycle transactions only 001 = 2 beat bursts 010 = 4 beat bursts 011 = 8 beat bursts 100 = 16 beat bursts 101 = 32 beat bursts 110 = 64 beat bursts 111 = unlimited bursts The GIO will never perform larger than a 64 beat burst since 64 beats from an 8-bit wide device completes an entire cache line, the largest transaction transferred over the crossbar. This field is only valid for non-multiplexed devices.	R/W	000
Handshake	1	1 = Wait for a data ready (RDY) handshake from the connected device. 0 = Rely only on timing for signal transitions.	R/W	0

Multiplexed	0	1 = Address and data are multiplexed on the AD bus. 0 = The address is asserted on the lower 24 bits of the AD bus and the data is asserted on the upper 8 bits.	R/W	0

Name: GIO_t_{Am1}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x04.

31-8	7-0
0	t_{Am1}

Field	Bits	Description	R/W	Reset
t_{Am1}	7-0	The number of crossbar clock cycles for which the first address of a transaction is driven for device *m*.	R/W	0xff

Name: GIO_t_{Am2}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x08.

31-8	7-0
0	t_{Am2}

Field	Bits	Description	R/W	Reset
t_{Am2}	7-0	The number of crossbar clock cycles for which the address is not driven between multiple device transactions to device *m*.	R/W	0xff

Name: GIO_t_{Am3}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x0C.

31-8	7-0
0	t_{Am3}

Field	Bits	Description	R/W	Reset
t_{Am3}	7-0	The number of crossbar clock cycles for which the address is driven for the second and subsequent beats of a many beat transaction to device *m*. Does not apply to the last beat of a transaction	R/W	0xff

Name: GIO_t_{Am4}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x10.

31-8	7-0
0	t_{Am4}

Field	Bits	Description	R/W	Reset
t_{Am4}	7-0	The number of crossbar clock cycles for which the address is driven for the last beat of a multi-beat transaction to device *m*. Does not apply to single beat transactions.	R/W	oxff

Name: GIO_t_{Dm1}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x14.

31-8	7-0
0	t_{Dm1}

Field	Bits	Description	R/W	Reset
t_{Dm1}	7-0	The number of crossbar clock cycles from the start of a transaction until the first write data beat is driven to device *m*.	R/W	0x55

Name: GIO_t_{Dm2}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x18.

31-8	7-0
0	t_{Dm2}

Field	Bits	Description	R/W	Reset
t_{Dm2}	7-0	The number of crossbar clock cycles for which each write data beat is driven to device *m*.	R/W	’0x55

Name: GIO_t_{RSm}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x1C.

31-8	7-0
0	t_{RSm}

Field	Bits	Description	R/W	Reset
t_{RSm}	7-0	The number of crossbar clock cycles after the start of a read transaction until the read data is sampled from device *m*. For burst transactions, the number of crossbar clock cycles after the last address transition until the last read data is sampled from device *m*.	R/W	’0x55

Name: GIO_t_{SELm1}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x20.

31	30-8	7-0
Active	0	t_{SELm1}

Field	Bits	Description	R/W	Reset
Active	31	Active state of t_{SELn1} 0 = active low 1 = active high	R/W	0
t_{SELm1}	7-0	The number of crossbar clock cycles from the start of the transaction until the assertion of GIO_SEL[*m*]	R/W	’0x2A

Name: GIO_t_{SELn2}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x24.

31-8	7-0
0	t_{SELm2}

Field	Bits	Description	R/W	Reset
t_{SELm2}	7-0	The number of crossbar clock cycles for which GIO_SEL[m] is asserted.	R/W	'0xAA

Name: GIO_t_{CTLmn1}.
Size: 32 bits.
Address: GIOBase + 0x58*m + 0x28 + 0xC*n.

31	30	29	28-8	7-0
Polarity	Read	Write	0	t_{CTLmn1}

Field	Bits	Description	R/W	Reset
Polarity	31	Active state of t_{CTLmn1} 0 = active low 1 = active high	R/W	0
Read	30	Indicates if control line is active for read operations. 0 = not active for reads 1 = active for reads	R/W	0
Write	29	Indicates if control line is active for write operations. 0 = not active for writes 1 = active for writes	R/W	0
t_{CTLmn1}	7-0	The number of crossbar clock cycles from the start of the transaction until the assertion of GIO_CTLm[n]	R/W	0x3F

Name: GIO_t_{CTLmn2}.
Size: 32 bits.
Address: GIOBase + 0x58**m* + 0x2C + 0xC**n*.

31-8	7-0
0	t_{CTLmn2}

Field	Bits	Description	R/W	Reset
t_{CTLmn2}	7-0	The number of crossbar clock cycles for which GIO_CTL*m*[*n*] is asserted.	R/W	0x6B

Name: GIO_t_{CTLmn3}.
Size: 32 bits.
Address: GIOBase + 0x58**m* + 0x30 + 0xC**n*.

31-8	7-0
0	t_{CTLmn3}

Field	Bits	Description	R/W	Reset
t_{CTLmn3}	7-0	The number of crossbar clock cycles between consecutive assertions of GIO_CTL*m*[*n*] in multi-beat transactions. Does not apply to single beat transactions.	R/W	0x5A

The Stream Processor 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 Stream Processor.

SP-1 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.

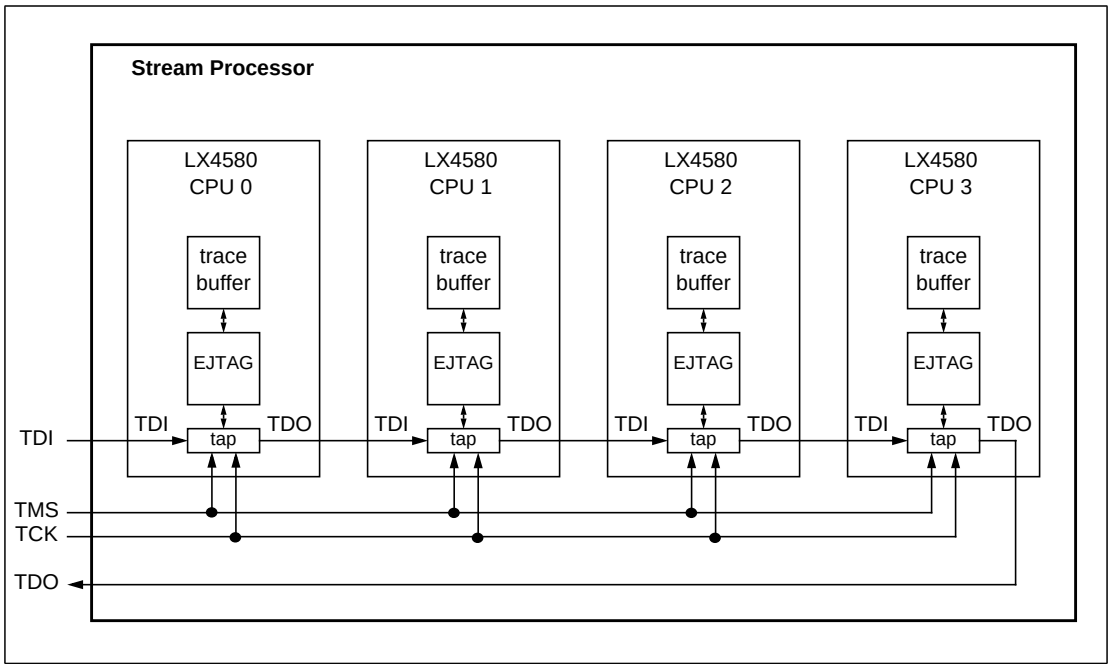

Figure 43: SP-1 ETJAG Organization

THE REMAINDER OF THIS CHAPTER IS FOR INTERNAL LEXRA USE.

15.1. 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.

15.1.1. EJTAG TAP Registers

Table 66: 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 66: 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

15.1.2. 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 67: 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	0x210 +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 67: EJTAG DRSeg Registers (Continued)

NAME	Offset	Field	O/D	Implementation Specific Information	Reset
DBVn	0x208 +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 68: 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

15.2. Description of LX4580 CPU Specific EJTAG features

15.2.1. 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.

15.2.2. 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.

15.2.3. 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.

15.2.4. 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.

15.2.5. Precise Data Breaks

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

15.2.6. Data Value Break Loads

Data value breaks on loads are not supported.

15.2.7. EJTAG_ADDR (36-bit)

As the Stream Processor 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.

15.2.8. PC Trace Buffer & TAC

15.2.8.1. 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.

15.2.8.2. CPU EJTAG Block Diagram

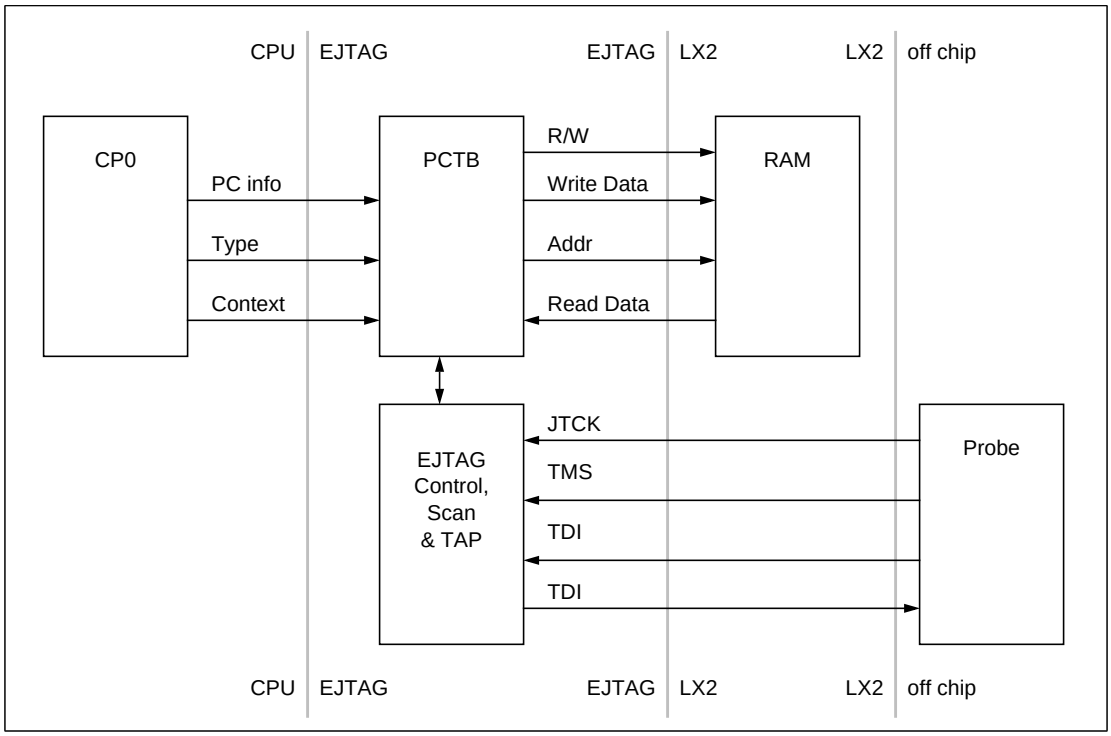

Figure 44: CPU EJTAG Block Diagram

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.

15.2.8.3. 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.

15.2.8.4. 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.

15.2.8.5. 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.

15.2.9. 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.

15.2.10. 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).

15.2.11. 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.

15.2.12. 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 Stream Processor normally and assert/de-assert cold reset (CRESET_N)

Connect the EJTAG probe to the Stream Processor 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.

15.2.13. The Lexra Probe

Contained in the Stream Processor is the Lexra Probe which sits between the RS232 UARTE block and the JTAG pins inside the design. ASCII characters can be sent to UARTE which cause instructions to be sent to CPUs TAP controllers via their JTAG pins.

On power-up the JTAG pins are connected to the JTAG pins of the Stream Processor, but if a character is received on UARTE the JTAG pins of the LX4580 will be switched to be connected to the Lexra Probe instead. Only a reset will change the mulitplexer back to the JTAG pins of the Stream Processor.

16.1. Performance Counter Overview

The Stream Processor's hardware performance counters allow applications to be analyzed and tuned to get the most performance possible. Software may select from a range of events to monitor and count during normal operation. This chapter summarizes the performance counters and provides cross-references to the detailed information for each counter.

The performance counters have the following characteristics:

- Accessible through CP0 or memory-mapped registers (depends on counter).
- Low software overhead.
- No impact on operation.
- Multiple 32-bit event counters.
- Selectable events per counter.
- TBD software supplied by Lexra.

16.2. Performance Counter Architecture

The performance counters are distributed throughout Stream Processor, with each counter contained within the functions that are counted. Counters have the structure shown in Figure 45, consisting of a 32-bit event counter and counter specific event selection logic. The event counters and selection logic are readable and writable under CPU program control.

The counters for some functions provide multiple event selection blocks and event counters, allowing more than one event type to be counted simultaneously. For example, to determine a cache miss ratio, the total number references and the number of misses may be counted simultaneously.

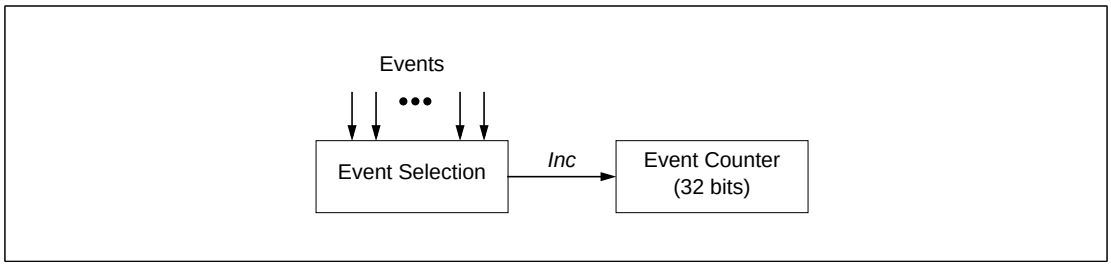

Figure 45: Performance Counter Structure

Counters within a CPU are accessed with the MFC0 and MTC0 instructions. Counters throughout the rest of the Stream Processor are accessed with load and store instructions, LW and SW.

16.3. Performance Counter Operation

An operating system may extend counter access rights to processes or device drivers. Only one device driver or process should write to a given counter's event selection logic or counter for initialization purposes. Any number of device drivers or processes can safely read the current counter settings.

All counters consist of a 32-bit value. There is no hardware provision for handling the rollover condition of the counters. At the worst-case rate of one increment per cycle, the minimum rollover period for any counter is several seconds, e.g. 8 seconds for 500 MHz operation. Software can easily detect a rollover condition and compensate for it by polling a counter more frequently than this period.

16.4. Summary of Performance Counters

Table 69 summarizes the Stream Processor performance counters, and includes a cross-reference to more detailed information within this data sheet. For replicated functions, e.g. multiple CPUs, the number of counters shown in the table is for each instance of the function.

Table 69: Summary of Performance Counters

Function	Counter Events	Counters	See Sec.
CPU	See MIPS32 chapter.	4	2.10
Memory Subsystem	See memory subsystem chapter	2	9.14
DMA Controller	The following events may be counted for the DMA controller as a whole or for a selected channel. Non-data refers to a transaction associated with DMA overhead such as reading a transfer descriptor. data line read data line write non-data line read non-data line write data sub-line write data sub-line read non-data sub-line write non-data sub-line read	2	XREF to be supplied.

Some functional blocks include performance monitoring features other than the standard counter type. These features listed in Table 70. Additional details on their structure and use are found in the cross-referenced section.

Table 70: Additional Performance Monitoring Features

Function	Performance Monitoring Feature	See Sec.
DMA	The following values may be sampled for each queue. current number of entries held in queue maximum number of entries held in queue	XREF to be supplied.

17.1. Error Detection and Reporting Overview

The Stream Processor incorporates error detection and reporting mechanisms required for robust software and hardware environments. This chapter summarizes these features and provides cross references to more detailed information throughout this data sheet.

The following error types are detected within the Stream Processor:

- Bus errors:
  - Access to an undefined location in the system address space.
  - Access to a defined location in the system address space, but with incorrect size.
- Hardware errors:
  - Crossbar transaction time-out.
  - Byte parity protection for all internal SRAMs.
  - ECC (single error correct and multi error detect) for external SDRAM.
  - Error detection for external interfaces, as defined by the interface standards.
  - INTERNAL USE. State consistency checks (e.g. L1 vs. L1 tag copies).
  - INTERNAL USE. Cmd/state consistency checks (e.g. command vs. L1/L2 state).
  - INTERNAL USE. Unexpected crossbar replies.

Exceptional conditions that may arise in a device, such as a DMA queue overflow, are not considered in this chapter. They are described in the applicable chapters.

The reporting of bus errors and hardware errors are handled differently.

Bus errors always cause a Bus Error exception to be taken by the CPU context that initiated the access.

Hardware errors generally cannot be attributed to a specific thread. When an error is detected, applicable information is recorded within the detecting module for later diagnosis. The Stream Processor is configurable to report hardware errors so that software can analyze the details and take appropriate action. The following reporting options are available:

- No report.
- Report error via a global level sensitive interrupt.
- Assert panic output pins for use by system level logic.

17.2. Bus Error Reporting

A bus error arises when a CPU context attempts an invalid access to the Stream Processor's address space. A bus error causes an Bus Error exception for the CPU context that attempts the invalid access. It is not possible to mask bus errors. They are always reported unconditionally to the applicable CPU context.

If a bus error is detected within the CPU's crossbar interface, no other modules are involved in the error signalling.

If the bus error is detected within a device, the device sends a crossbar Bus Error (BE) message to the CPU context that issued the offending request. If the offending request would normally have corresponding reply, the BE message is issued in place of the reply.

The CPU's crossbar interface converts the BE transaction into a Bus Error exception. The CPU also releases any internal resources that would otherwise be held up until the receipt of the normal reply for the request that caused the BE reply.

17.3. Hardware Error Reporting

Hardware error detection and reporting blocks are distributed throughout the Stream Processor's logic modules. Each module includes the general features shown in Figure 46.

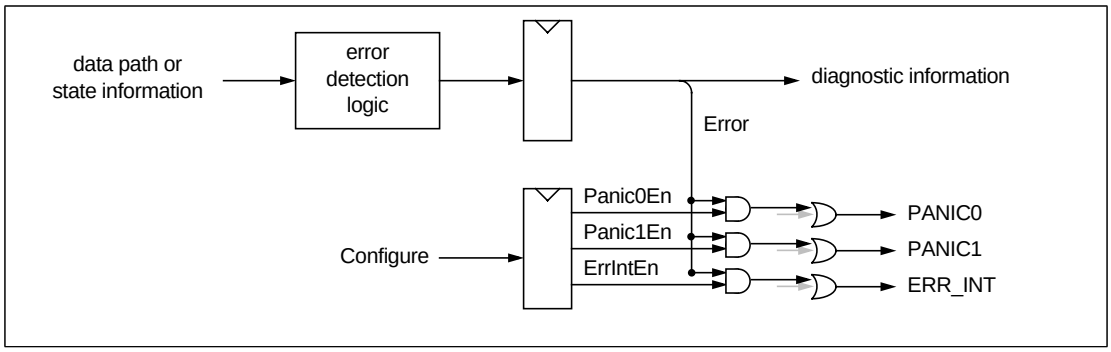

Figure 46: Error Detection Block Architecture

Error detection logic within the block monitors the signals from a data path or other state within the Stream Processor. The detection logic provides three error output signals and additional diagnostic information that is captured in a register. The diagnostic information is captured upon the detection of the first error within the module and is held until it is cleared by software or a hardware reset.

Each block of error logic includes a configuration register that determines how its error(s) are to be reported through the module's three output signals.

- Panic0En - assert the Stream Processor's PANIC0 output pin.
- Panic1En - assert the Stream Processor's PANIC1 output pin.
- ErrIntEn - assert modules error interrupt signal.

The three reporting options for each detectable error can be set independently. Any combination of reporting options listed above can be enabled for each detectable error. As a result of reset, all hardware error reporting options are disabled. Software can enable any combination of reporting options for a module by setting the appropriate bits in the modules's configuration registers.

The error flags from the detection logic are ANDed with the enable terms from the configuration register. The PANIC0 and PANIC1 outputs of all modules are reduction ORed to produce the Stream Processor's PANIC0 and PANIC1 output signals. The ERR_INT output from each module is passed to the Interrupt Router and Reflector (IRR), The IRR retains an indication of the first module that reports an error and sources a global error interrupt signal. The (See Section 4.3.3.)

The Stream Processor reports hardware errors in the most timely manner possible without compromising the clock speed of the hardware. However, the number of cycles that elapse between an error condition and the reporting through a panic or interrupt condition is not defined. The global interrupt used to signal a hardware error to the CPU is by definition imprecise.

17.4. Error Detection Configuration Registers

This section summarizes error detection and reporting registers and provides cross-references to more detailed information within this data sheet.

TABLE TO BE SUPPLIED.

17.5. Error Detection and Reporting Pins

This section describes the Stream Processor IC pins related to error detection and reporting.

Signal Name	Direction	Description
PANIC0	output	Asserted (high) if one or more errors are detected and the corresponding Panic0En configuration term is set to 1. Otherwise, de-asserted (low). The meaning of PANIC0 and the system's response to its assertion is application defined.
PANIC1	output	Asserted (high) if one or more internal Stream Processor errors are detected and the corresponding Panic1En configuration term is set to 1. Otherwise, de-asserted (low). The meaning of PANIC1 and the system's response to its assertion is application defined.

18.1. Interfaces

Table 71: Interface Summary

Name	Pins	Direction	Description
Clocks and Reset (4 pins)			
CPUCLK	1	input	CPU core clock.
CPUCLK_N	1	input	CPU core clock differential input.
CRESET_N	1	input	Cold Reset.
DMACLK	1	input	DMA core clock.
DMACLK_N	1	input	DMA core clock differential input.
JTAG_RST_N	1	input	Connection from the EJTAG probe.
Interrupts (4 pins)			
INT0_N	1	input	External Interrupt Line 0. Active low.
INT1_N	1	input	External Interrupt Line 1. Active low.
INT2_N	1	input	External Interrupt Line 2. Active low.
INT3_N	1	input	External Interrupt Line 3. Active low.
Chip Test and Software Debug (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.
Error Detection (2 pins)			
PANIC0	1	output	Hardware error detection panic output 0.
PANIC1	1	output	Hardware error detection panic output 1.
DDR SDRAM (2 interfaces; n=0,1; total of 230 pins)			
MC*n*_RAS_N	1	output	Row address strobe
MC*n*_CAS_N	1	output	Column address strobe
MC*n*_WE_N	1	output	Write enable
MC*n*_S_N[3:0]	4	output	Chip select
MC*n*_BA[1:0]	2	output	Bank address
MC*n*_A[13:0]	14	output	Address
MC*n*_DQ[71:0]	72	inout	Data bus
MC*n*_DQS[8:0]	9	inout	Data strobe
MC*n*_DQM[8:0]	9	output	Data mask

Table 71: Interface Summary

Name	Pins	Direction	Description
MC*n*_CKE	1	output	Clock enable
MC_MEMCLK	1	input	Memory clock
GMII (3 interfaces; n=0,1,2; total of 81 pins)			
GMII*n*_MDC	1	output	Management clock.
GMII*n*_MDIO	1	inout	Management data to/from PHY.
GMI*n*_GTX_CLK	1	input	Transmit reference clock, 2.5, 25 or 125 MHz.
GMII*n*_TX_CLK	1	output	Transmit clock to PHY.
GMII*n*_TXD[7:0]	8	output	Transmit data to PHY.
GMII*n*_TX_EN	1	output	Transmit data enable to PHY.
GMII*n*_TX_ER	1	output	Transmit error control to PHY.
GMII*n*_CRS	1	input	Carrier sense from PHY.
GMII*n*_COL	1	input	Collision detect from PHY.
GMII*n*_RX_CLK	1	input	Receive clock from PHY.
GMII*n*_RXD[7:0]	8	input	Receive data from PHY.
GMII*n*_RX_DV	1	input	Receive data valid from PHY.
GMII*n*_RX_ER	1	input	Receive error control from PHY.
PCI-X (1 Interface, total of 56 pins)			
PCIX_CLK	1	input	Clock
PCIX_PAR	1	inout	Parity.
PCIX_RST_N	1	input	Reset.
PCIX_AD[31:0]	32	inout	Address or data.
PCIX_CBE_N[3:0]	4	inout	Command and byte enable flags.
PCIX_FRAME_N	1	inout	Frame.
PCIX_IRDY_N	1	inout	Initiator ready.
PCIX_TRDY_N	1	inout	Target ready.
PCIX_STOP_N	1	inout	Stop.
PCIX_DEVSEL_N	1	inout	Device select.
PCIX_IDSEL	1	input	Initialization device select.
PCIX_LOCK_N	1	inout	Lock target.
PCIX_PERR_N	1	inout	Parity error.
PCIX_SERR_N	1	inout	System error.
PCIX_REQ_N	1	output	Stream Processor's request to external arbiter.

Table 71: Interface Summary

Name	Pins	Direction	Description
PCIX_GNT_N	1	input	Grant from external arbiter.
PCIX_REQIN_N[2:0]	3	input	Requests from external PCI devices to Stream Processor's internal arbiter.
PCIX_GNTOUT_N[2:0]	3	output	Grants to external PCI devices to Stream Processor's internal arbiter.
I^2C (1 Interface, total of 2 pins)			
I2C_SCL	1	inout	Serial clock.
I2C_SDA	1	inout	Serial data.
General Purpose Input/Output (1 Interface, total of 41 pins)			
GIO_AD[31:0]	32	inout	Address (output only) or data (inout).
GIO_SEL_N[3:0]	4	output	Device select.
GIO_CTL[3:0]	4	output	Controls.
GIO_RDY_N	1	input	Device ready.
UART (1 Interface, total of 4 pins)			
UART_DTR	1	output	Data terminal ready.
UART_DSR	1	input	Data set ready.
UART_RXD	1	input	Receive data.
UART_TXD	1	output	Transmit data.

Index