

Lanai X

REVISION HISTORY

REVISION	COMMENT
1.0	Initial document.
1.1	<p>Chip Restrictions: Page 20 - Send interface restrictions.</p> <p>Specification Corrections: Page 9 - Length of the copy segment must not be equal to 0. Page 11, 12 - Worst-case computation for the minimum STOP_SIZE modified. Page 60, 67 - Receive buffers resized in the message-passing example, to conform with the above. Page 15 - All incoming packets directed to the alternate receive channel and dropped for lack of buffer space are counted, including InfiniBand Link Packets.</p> <p>Specification Clarifications: Page 11 - X-Port flow control <i>vs.</i> the alternate receive channel. Page 12 - Accounting of dropped incoming packets.</p>

TABLE OF CONTENTS

1.	INTRODUCTION	3
2.	ABOUT THIS DOCUMENT	5
3.	HIGHLIGHTS OF LANAI X–LANAI 9 DIFFERENCES	6
3.1.	Software	6
3.2.	Hardware	6
4.	RELEVANT STANDARDS	6
5.	MEMORY ARCHITECTURE	7
6.	SPECIAL REGISTERS	7
6.1.	Port Registers	7
6.2.	Saturating Counters	7
7.	MISCELLANEOUS SPECIAL REGISTERS	8
8.	COPY/CRC32 ENGINE	9
9.	PACKET-INTERFACE RECEIVER	10
9.1.	SAN Port	11
9.2.	X Port	11
9.3.	X Port — Myrinet	11
9.4.	X Port — Gigabit Ethernet	12
9.5.	X Port — InfiniBand	12
9.6.	Received-Packet Descriptor (RPD)	13
9.7.	Receiver Special Registers	14
10.	PACKET-INTERFACE SENDER	19
10.1.	Cut-Through Packet Forwarding	19
10.2.	Send-Packet-Segment Descriptor (SPD)	20
10.3.	Cut-Through SPD	21
10.4.	Sender Special Registers	22
11.	PCI INTERFACE	24
11.1.	PCI Configuration Space	25
11.2.	PCI Interface Initialization	27
12.	PCI DMA INTERFACE	28
12.1.	PCI DMA-Request Descriptor (DRD)	29
12.2.	PCI-DMA Special Registers	31
13.	COMMUNICATION PORTS	33
13.1.	Port-Access Special Registers	33
13.2.	Configuration Port Registers	34
13.3.	Shared Port Registers	41
13.4.	Myrinet-Specific Port Registers	41
13.5.	Gigabit-Ethernet-Specific Port Registers	42
13.6.	InfiniBand-Specific Port Registers	44
14.	INTERFACE-STATUS REGISTERS	45
15.	EVENT-DRIVEN CODE DISPATCH	50
16.	MEMORY ARBITRATION	52
17.	JTAG INTERFACE, EEPROM INTERFACE	53
A.	SUMMARY OF SPECIAL REGISTERS	54
B.	SUMMARY OF COMMUNICATION-PORT REGISTERS	58
C.	MESSAGE-PASSING EXAMPLE	60
D.	PCI-DMA EXAMPLE	73

1. INTRODUCTION

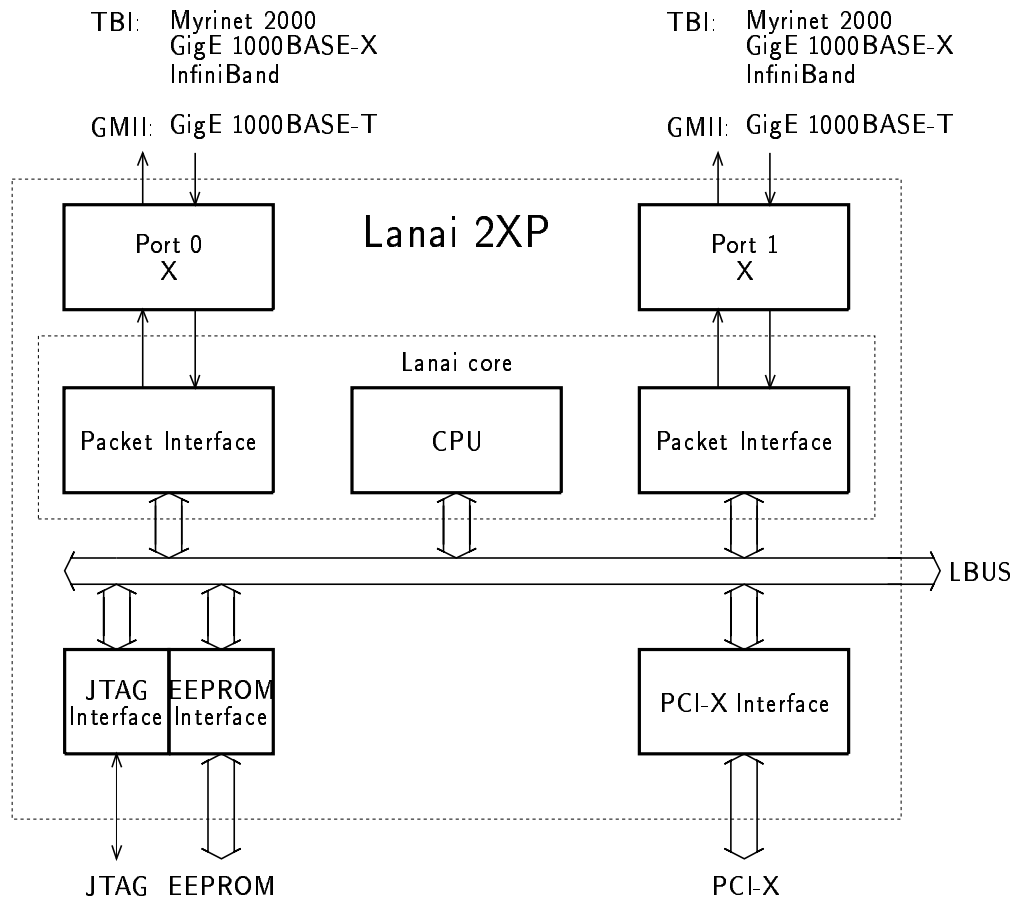
The Lanai X is a series of programmable devices with at least one multi-protocol communication port, referred to as the X-port. Each X-port can be configured to operate in Myrinet, Gigabit Ethernet, or InfiniBand mode.

There are currently three variants of Lanai X chips, which differ in the number and type of interfaces that they support:

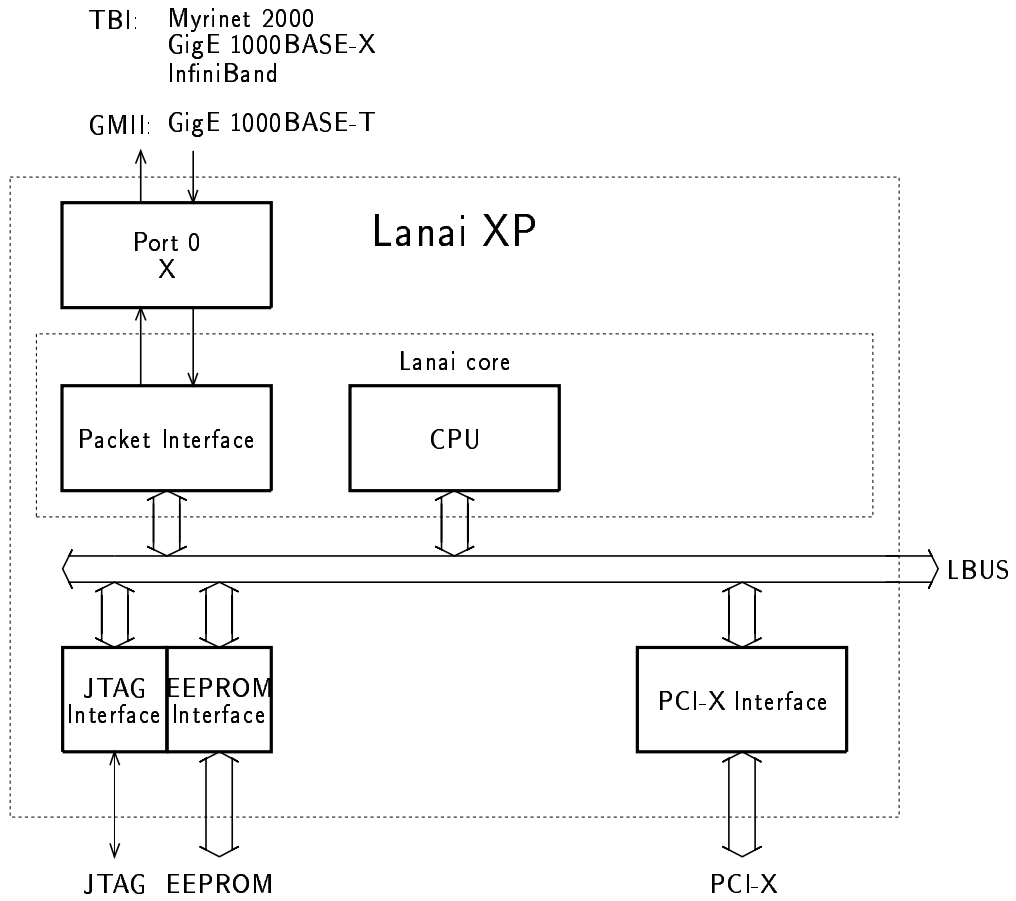
Lanai type	Interfaces Supported		
	X ports	SAN ports	PCI-X
Lanai XM	1	1	0
Lanai XP	1	0	1
Lanai 2XP	2	0	1

The fully-configured Lanai 2XP is illustrated below; it consists of the Lanai core (CPU with two packet interfaces), two X-ports, JTAG interface, EEPROM interface, and PCI-X I/O interface.

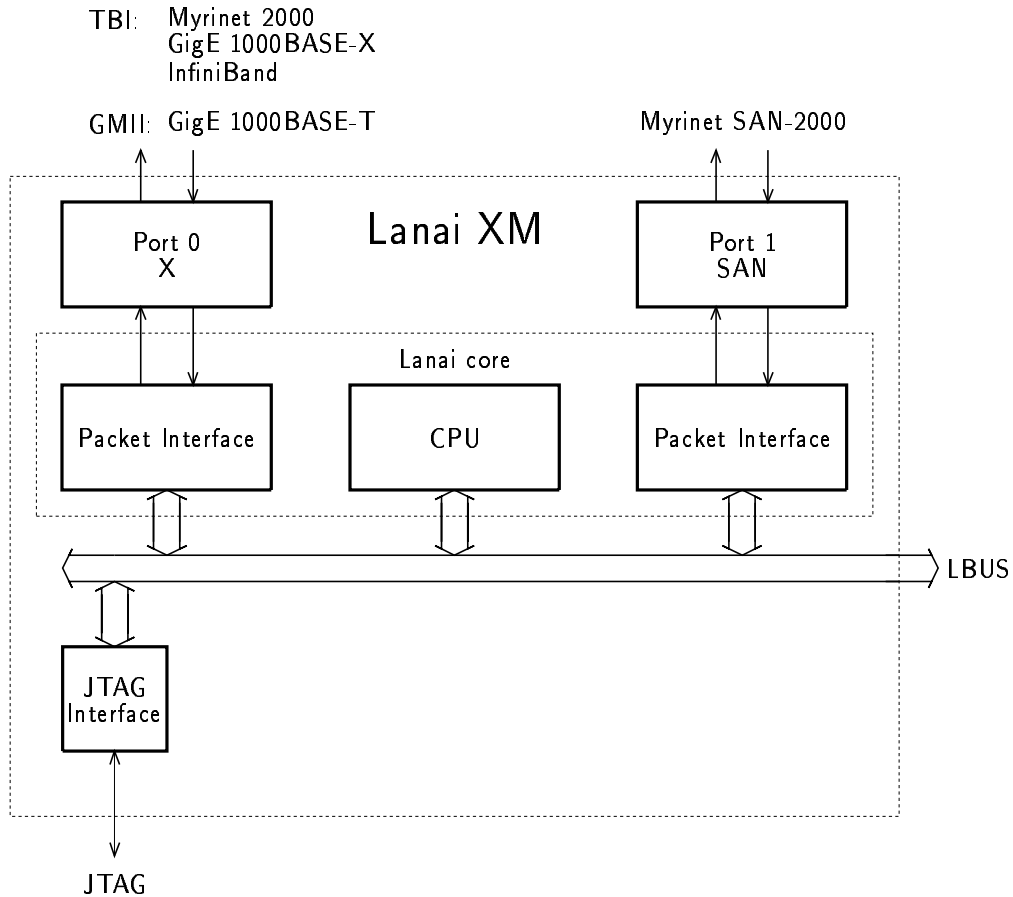
The Local Bus (LBUS) is an interface to fast, synchronous, static SRAM.



The Lanai XP has a single X port.



The Lanai XM is a version of the chip without the EEPROM and PCI-X interface. It also replaces one X port with the SAN port, an interface to the Myrinet system-area network (Myrinet SAN-2000); its intended use is as a Myrinet-SAN to Myrinet/Gigabit-Ethernet/InfiniBand protocol converter.



2. ABOUT THIS DOCUMENT

This document is a reference manual, not a tutorial, and the amount of information it contains can be overwhelming. However, the great majority of details is useful only for configuration; the Lanai code that is required to control the various DMA engines, once configured, is fairly simple.

For a novice, the information on pages 10, 11, 19, and 28 should be enough to grasp the big picture. This should be followed by becoming familiar with the status registers, on pages 45 through 49, and the event-dispatch mechanism on page 50. After that, one should follow the examples in Appendices C and D; they present simple but complete examples of the Lanai code required to initialize and use the network interfaces and the PCI DMA engine.

3. HIGHLIGHTS OF LANAI X-LANAI 9 DIFFERENCES

The architecture of the Lanai X chips is a significant departure from the previous Lanai versions. Only the CPU and the SAN port have not changed compared to the Lanai 9. Modifications in the packet interface and the host interface typically require non-trivial modifications of both Lanai and device-driver code.

3.1. Software

- The packet interface supports multiple pending packet buffers and arbitrary byte-alignment of DMA-blocks, for both send (page 19) and receive (page 10).
- Lanai X provides hardware support for event-driven code dispatch (page 50). This mechanism does not have to be used, but it offers a fast solution with a minimal memory-resident dispatch table.
- There is a memory-copy engine, and a CRC-32 compute engine. The input byte stream for the CRC-32 engine can be provided by the copy engine or by program I/O (page 9).
- The Lanai 9 EBUS interface has been replaced with the PCI-X I/O interface. Most of the functionality of the Myricom's PCIDMA chip has been incorporated into the Lanai X, including the support for multiple pending host DMAs.

3.2. Hardware

- The Lanai 9 EBUS interface has been replaced with the PCI-X interface (except in the Lanai XM, which has no host interface).
- The Lanai X chip contains a JTAG interface, and an EEPROM interface, enabling hostless initialization.
- The X port supports the TBI interface, for direct connection to a SerDes chip, and the GMII interface (including Management Interface), for direct connection to a Gigabit-Ethernet PHY chip.
- Clocking of the Lanai core is completely independent from the rest of the chip (no multiple-of-EBUS-clock requirement).
- The LBUS memory bus arbitration policy is configurable, and guarantees a fraction of the total LBUS bandwidth and a maximum response time to the bus contenders (page 52).

4. RELEVANT STANDARDS

ANSI/VITA 26-1998 — referred to as ANSI Myrinet.

<http://www.myri.com/open-specs/serial.pdf> — referred to as Serial Myrinet.

IEEE Std 802.3, 2000 Edition — referred to as IEEE 802.3.

InfiniBand Architecture Specification, Release 1.0.a — referred to as IAS.

PCI Local Bus Specification, Revision 2.2 — referred to as PCI.

PCI-X Addendum to the PCI Local Bus Specification, Revision 1.0 — referred to as PCI-X.

IEEE Std 1149.1-1990 — referred to as JTAG.

5. MEMORY ARCHITECTURE

In the remainder of this specification, we shall refer to 8-bit data units as bytes, to 16-bit units as half-words, to 32-bit units as words, and to 64-bit units as double-words. Although the internals of the Lanai X support 32-bit addresses, PCI-related system limitations restrict the LBUS memory to a maximum of 8MB. The top half of the address space (with the most significant address bit equal to 1) is reserved for the on-chip special registers described below.

The local memory is the main communication medium between the Lanai X subsystems, and all data streams pass through it. With two full-duplex high-bandwidth communication ports, PCI-X interface, memory-copy engine, and CPU, all competing for LBUS cycles, the most precious resource in a system with the Lanai X chip is typically the LBUS bandwidth. The LBUS memory bus arbitration policy is configurable, and it guarantees a fraction of the total LBUS bandwidth and a maximum response time to the bus contenders (page 52).

The LBUS addresses are byte addresses, and the byte order is big-endian (the most-significant byte is stored at the lowest byte address). The 2-, 4-, and 8-byte memory accesses on the LBUS must be aligned; any least-significant bits of an address that would make a memory access non-aligned are ignored.

The Lanai provides a rudimentary memory-protection mechanism that can detect write accesses outside of a configurable memory segment (page 8).

6. SPECIAL REGISTERS

Operation of the on-chip interfaces is controlled by a set of status/control registers referred to as *special registers* (specials). See page 54 for a summary.

Unless specified otherwise:

- specials are mapped into the top half of the LBUS memory space (the most significant address bit equal to 1). They can be accessed by the CPU, through the JTAG interface (page 53), and, except for a few special registers with read side-effects (page 54), through the PCI-X interface (page 24);
- specials are read-write registers;
- the specials with up to 8 bits should be accessed as bytes, the ones with 9-16 bits as half-words, and the ones with more than 16 bits as words, with addresses specified on page 54.

6.1. Port Registers

The communication ports operate in clock domains that are independent from the Lanai-core clock. As a result, the control/status registers associated with the port circuitry (see page 58 for a summary) cannot be accessed using the standard, memory-mapped, special-register-access mechanism.

Instead, the port registers are accessed through the two port-access special registers described on page 33.

6.2. Saturating Counters

Some of the special and/or port registers are described as *read-only*, *reset-on-read*, *saturating counters*. A saturating counter does not overflow; it does not increment past the all-ones state. These registers are used to resolve the mutual exclusion between the CPU and the interfaces. An interface "produces" an event by incrementing a counter, the CPU "consumes" a number of events by reading the counter (the number equal to the value read). The chip guarantees that no events are lost (unless the counter saturates).

7. MISCELLANEOUS SPECIAL REGISTERS

REGISTER	DESCRIPTION												
CPUC[31:0]	CPU Clock: This counter is incremented every CPU clock.												
RTC[31:0]	Real-Time Clock: This counter is incremented every $1/2\mu\text{s}$.												
IT0[31:0] IT1[31:0] IT2[31:0]	<p>Interval Timers: These counters are decremented every $1/2\mu\text{s}$. Whenever an interval timer makes a transition from 0x00000000 to 0xFFFFFFFF, the corresponding TIME_INT bit of the special register ISR is set (page 45). Writing an interval timer clears the corresponding TIME_INT bit, with the maximum delay of 1 clock cycle.</p> <p>If the ENABLE_PAUSE_TIMER bit of the special register channel_RPL_CONFIG (page 16) is set, when a valid Gigabit Ethernet MAC Control PAUSE packet is received on Port 0, IT0 is set to 0 (Port 1 sets IT1 to 0).</p>												
LED[3:0]	LED Register: The value of this register is driven to the LED[3:0] output pins.												
MDI[2:0]	<p>GMII Management Data Interface:</p> <table border="1"> <thead> <tr> <th>BITS</th> <th>NAME</th> <th>DESCRIPTION</th> </tr> </thead> <tbody> <tr> <td>2</td> <td>MDC</td> <td>The value of MDC bit is driven to the MDC output pin.</td> </tr> <tr> <td>1</td> <td>MDEN</td> <td>This bit controls the direction of the MDIO pin. If MDEN bit is equal to 1, the MDIO bit is driven to the MDIO pin; if it is equal to 0, the MDIO pin is not driven.</td> </tr> <tr> <td>0</td> <td>MDIO</td> <td>If MDEN bit is equal to 1, the MDIO bit is driven to the MDIO pin. Reading the MDI register returns the value of the MDIO pin in this bit.</td> </tr> </tbody> </table> <p>This register is typically used to implement the GMII Management Data Interface, as specified in IEEE 802.3 §22.2.4.5. When one or both X ports operate in GMII mode, typically used in conjunction with off-chip Gigabit Ethernet 1000BASE-T devices, this interface can be used to access the devices' GMII registers.</p>	BITS	NAME	DESCRIPTION	2	MDC	The value of MDC bit is driven to the MDC output pin.	1	MDEN	This bit controls the direction of the MDIO pin. If MDEN bit is equal to 1, the MDIO bit is driven to the MDIO pin; if it is equal to 0, the MDIO pin is not driven.	0	MDIO	If MDEN bit is equal to 1, the MDIO bit is driven to the MDIO pin. Reading the MDI register returns the value of the MDIO pin in this bit.
BITS	NAME	DESCRIPTION											
2	MDC	The value of MDC bit is driven to the MDC output pin.											
1	MDEN	This bit controls the direction of the MDIO pin. If MDEN bit is equal to 1, the MDIO bit is driven to the MDIO pin; if it is equal to 0, the MDIO pin is not driven.											
0	MDIO	If MDEN bit is equal to 1, the MDIO bit is driven to the MDIO pin. Reading the MDI register returns the value of the MDIO pin in this bit.											
MP_LOWER[30:3] MP_UPPER[31:3]	<p>Memory Protection: The Lanai X chip provides a rudimentary memory-protection mechanism that can detect write accesses outside of a configurable memory segment. A memory write to an address ADDR is allowed if $\text{MP_LOWER} \leq \text{ADDR} < \text{MP_UPPER}$. When a memory write outside of this segment is detected, the MEMORY_INT bit of the special register ISR is set (page 45).</p> <p>Typically, MP_LOWER points to the beginning of the program data segment, and MP_UPPER is equal to the size of the LBUS SRAM.</p> <p>Upon reset, $\text{MP_LOWER} = 0$, and $\text{MP_UPPER} = 0x80000000$, allowing writes to any memory location.</p>												
PLL[31:0]	<p>Clocking Configuration: Communication ports of the Lanai X chip operate in clock domains that are independent from the CPU/memory clock; the receive clocks are derived from the input data streams, the transmit clocks from the 125MHz reference clock on the REF input pin.</p> <p>After the chip is out of reset, the PLL and SYNC registers must be initialized with Myricom-supplied configuration, which is a part of the Lanai-programming toolkit targeted for each Lanai application. See the initialize_PLLs() function in Appendix C for an illustration. After these two registers have been written, the communication ports must not be enabled for at least 10ms, to allow the on-chip PLLs to stabilize.</p>												
SYNC[31:0]	<p>Synchronizer Configuration: See the above description of special register PLL. The top 6 bits of the SYNC register are used only for hardware debugging. They select 1 of 64 internal signals to be output on the WIN pin, for timing observation.</p>												

8. COPY/CRC32 ENGINE

The following special registers control the memory-copy and CRC-32 compute engine:

REGISTER	DESCRIPTION						
COPY[31:0]	<p>Memory Copy: This write-only register should be written three times to initiate a memory-copy operation: the first write specifies the source address, the second specifies the length, and the third specifies the destination address. The source and destination are arbitrary byte addresses with their most-significant bit equal to 0. If the most-significant bit of an address is equal to 1 (the special register address space), the resulting behavior is undefined. The second, length word consists of the following bits:</p> <table border="1" data-bbox="841 632 1182 737"> <thead> <tr> <th data-bbox="841 632 943 667">BITS</th> <th data-bbox="943 632 1182 667">NAME</th> </tr> </thead> <tbody> <tr> <td data-bbox="841 667 943 703">31</td> <td data-bbox="943 667 1182 703">COPY_TO_CRC32</td> </tr> <tr> <td data-bbox="841 703 943 737">30-0</td> <td data-bbox="943 703 1182 737">COPY_LENGTH</td> </tr> </tbody> </table> <p>COPY_LENGTH is the number of bytes to be copied; if it is equal to 0, the resulting behavior is undefined. If COPY_TO_CRC32 bit is equal to 1, the data stream is directed to the CRC32 register, and the destination address must not be written. Upon the first write into the COPY register, the COPY_BUSY bit of the special register ISR is set (page 45), and it is cleared when the copy operation has completed. If the COPY register is written while a COPY operation is in progress (before COPY_BUSY is cleared), the resulting behavior is undefined.</p>	BITS	NAME	31	COPY_TO_CRC32	30-0	COPY_LENGTH
BITS	NAME						
31	COPY_TO_CRC32						
30-0	COPY_LENGTH						
COPY_ABORT[0]	<p>Memory Copy Abort: Writing this write-only register aborts the current copy operation (if any).</p>						
CRC32_CONFIG[2:1]	<p>CRC-32 Configuration: Myrinet, Gigabit Ethernet, and InfiniBand, all use the same generating polynomial for the channel CRC-32: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$. However, the bit order (within a byte) for channel CRC-32 computation in Gigabit Ethernet and InfiniBand is least-significant bit first, and in Myrinet it is most-significant bit first. When replicating the channel CRC-32 computation in the CRC-32 engine, one should set the CRC32_CONFIG bits to 00 for Myrinet, and to 11 for Gigabit Ethernet and InfiniBand.</p>						
CRC32[31:0]	<p>CRC-32: We shall define the CRC-32 computation as a function of 32 state bits and 8 input bits, producing 32 next-state bits: $next_state = crc32(state, input)$. This register provides the state bits for this computation. To replicate the channel CRC-32 computation in the CRC-32 engine, one should initialize the CRC32 register to 0xFFFFFFFF, apply the byte stream either by the copy engine or by the single-access registers (described next), and complement the final value of CRC32. If a memory copy with the CRC32 register as its target is in progress (COPY_BUSY equal to 1), writing any of the CRC32 registers results in undefined behavior. This register should not be read during the clock cycle immediately following the write to CRC32_BYTE/CRC32_HALF/CRC32_WORD.</p>						
CRC32_BYTE[7:0]	<p>CRC-32 Byte: When a BYTE is written to this write-only register, the CRC32 register is assigned the value of $crc32(CRC32, BYTE)$.</p>						
CRC32_HALF[15:0]	<p>CRC-32 Half-Word: When a half-word is written to this write-only register, the CRC32 computation is performed on the two bytes, most-significant byte first.</p>						
CRC32_WORD[31:0]	<p>CRC-32 Word: When a word is written to this write-only register, the CRC32 computation is performed on the four bytes, most-significant byte first.</p>						

9. PACKET-INTERFACE RECEIVER

There is one receive-DMA engine (*receiver*) on the chip for each network port. A packet arriving from the network is written to memory by the corresponding receiver.

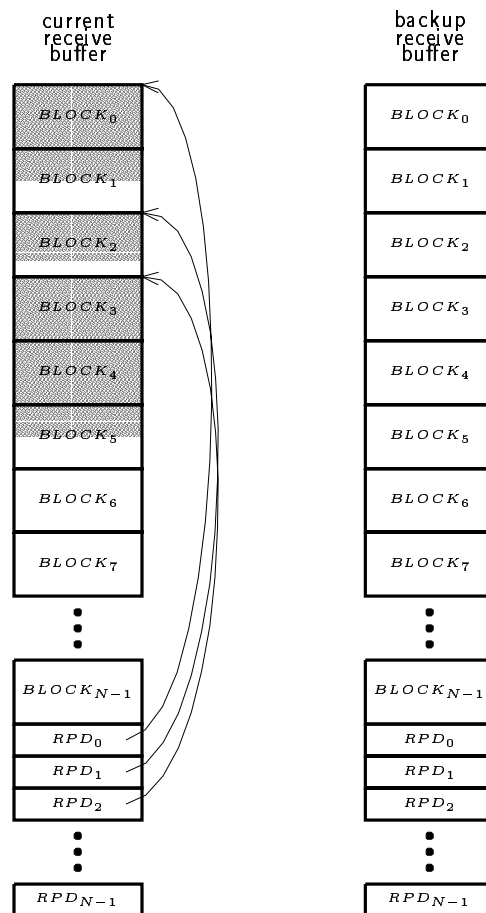
A set of receive buffers and the associated control/status registers is called a receive *channel*. Each port has two receive channels, *regular* and *alternate* (*A*), and can be configured to separate the incoming packets into the two channels, based on the incoming-packet header (page 17).

The unit of receive-buffer allocation is a *buffer*; buffers are aligned on double-word boundary, each buffer contains a fixed number of *blocks*, N , and each block contains a fixed number of bytes, B . Both N and B are configurable (page 18).

There are at most two receive buffers associated with a channel at any time: the *current receive buffer*, and the *backup receive buffer*.

Incoming packets are written into the current receive buffer, always starting at a block boundary. A packet can span several blocks, up to the *maximum packet length*, which is also configurable (page 18). When the receiver determines that the current receive buffer does not have sufficient space for a packet of maximum length, it switches to the backup receive buffer, thereby ensuring that packets are never fragmented.

At the end of each receive buffer is the *Received-Packet List*, (RPL) for that buffer. When a complete packet has been written into the current receive buffer, a *Received-Packet Descriptor* (RPD, page 13), consisting of a pointer to the start of the packet, the packet length, and some additional status information, is written into RPL. The maximum number of packets in a receive buffer is equal to the number of blocks in the buffer, so RPL must have room for N RPD entries.



A Receive Channel

How to handle the receive-buffers is one of the most important design decisions in a system using the Lanai X chip, and it should be addressed with careful consideration. It depends on the flow-control mechanism of the protocol used on the port at the time, on memory-arbitration policy, and on consumption of received packets by the CPU.

Regardless of the mode a port is operating in, the flow-control mechanism is affected only by the amount of buffering available in the regular channel, not the alternate channel. When there are no receive buffers in the alternate channel, all packets directed to it are dropped.

9.1. SAN Port

Handling of the data stream from the SAN port to the corresponding receiver is fairly simple. This interface uses a byte-to-byte flow-control; when the receiver runs out of receive buffers, the input data stream is stopped, and it resumes when a receive buffer is available. The SAN interface contains a slack-buffer sufficient for the longest-possible, 3-meter, SAN cable, so it, too, can arbitrarily stop the data flow.

Packet consumption still has to be guaranteed, and if the Lanai fails to consume a packet, it will get dropped, but the time constants for the watchdog timer (page 37) are on the order of seconds.

This behavior is consistent with SAN ports in all previous Lanai versions.

9.2. X Port

Handling of the data stream from the X port to the corresponding receiver has drastically more stringent requirements, since the packet-interface receive buffers also serve as the cable slack buffer. In absence of a receive buffer, or of sufficient memory cycles, the incoming data is dropped. We present here a discussion on some of the design trade-offs, and suggest an approach to receive-buffer management for each X-port mode of operation: Myrinet, Gigabit Ethernet, and InfiniBand. These approaches do not have to be adhered to, but they should be consulted to gain an insight into the issues involved.

9.3. X Port — Myrinet

The bit-serial, 8b/10b-encoded Myrinet links use STOP and GO control symbols for implementing the flow control. These control symbols can be emitted at any time, including in the middle of a packet, and implement a byte-level flow control. The standard Myrinet fiber links are up to 200-meters long, and the round-trip delay in such a cable is approximately $2.6\mu\text{s}$. At 250MB/s data rate, this corresponds to approximately 650 bytes of data in flight. In other words, once a port sends a STOP symbol to the port on the other end of the link, it must be ready to receive up to 650 more bytes before the STOP takes effect. Myricom's products use approximately 1KB of buffer space for this purpose, for an extra safety margin.

To configure a receive channel, the programmer must select the block size, buffer size, maximum packet length, and "stop size" (see below).

When selecting the block size, one tradeoff is that the smaller block size implies less wasted buffer space between packets, but it requires a larger RPL (at the limit, when using 8-byte block size, half of the space in the receive buffers is used for RPLs).

An additional consideration is that the STOP flow-control symbol must be sent assuming worst-case buffer utilization. The worst-case utilization occurs when all incoming packets are of the size that wastes the largest fraction of buffer space between packets; in addition, the receiver will stop using a receive buffer when a maximum-length packet cannot fit in it.

For example, if the minimum valid packet size is 20B, the following is the worst-case buffer utilization for several block sizes:

block size	worst-case buffer utilization	minimum stop size (guaranteed 1KB of buffer space)
8B	25/32	1.28KB + maximum packet length
16B	20/32	1.6KB + maximum packet length
32B	20/32	1.6KB + maximum packet length
64B	20/64	3.2KB + maximum packet length
128B	20/128	6.4KB + maximum packet length

The most conservative approach would be to use the minimum valid packet size of 5B (the shortest valid Myrinet-packet, as defined in ANSI Myrinet), since all packets up to 4B are dropped by the port itself (page 41):

block size	worst-case buffer utilization	minimum stop size (guaranteed 1KB of buffer space)
8B	5/8	1.6KB + maximum packet length
16B	5/16	3.2KB + maximum packet length
32B	5/32	6.4KB + maximum packet length
64B	5/64	12.8KB + maximum packet length
128B	5/128	25.6KB + maximum packet length

The buffer size must be larger than the stop size (page 18).

9.4. X Port — Gigabit Ethernet

The Gigabit Ethernet link-level flow-control mechanism is specified in IEEE 802.3 §31B. This mechanism uses PAUSE, a dedicated MAC Control packet type (Length/Type field equal to 88-08, Mac Control Opcode equal to 00-01).

The Lanai X can be configured to detect incoming packets with Length/Type/Opcode of 88-08-00-01, and, when such a packet is received, to stop sending packets and reset the IT timer (page 23, 8). Upon expiration of the pause timeout, packet sending must be explicitly re-enabled by the programmer (page 23).

Handling of all received packets (including PAUSE packets, whether or not they cause the packet sending to stop), is the responsibility of the programmer.

9.5. X Port — InfiniBand

The InfiniBand link-level flow-control mechanism is specified in IAS §v1.7.9. This mechanism uses 64-byte blocks and an absolute-credit-based scheme that can be implemented using the Lanai X channels.

The flow-control credits are exchanged using InfiniBand link packets, which can be diverted to the alternate channel (page 17) for special attention.

The InfiniBand VL15 packets do not use link-level flow-control, and they, too, can be diverted to the alternate channel.

The Lanai X does not provide any hardware support for more than one data virtual-lane (optional feature, IAS §v1.7.6); the additional lanes can be implemented in software.

Handling of all received packets (including flow-control and VL15 packets, whether or not they are diverted to the alternate channel) is the responsibility of the programmer.

9.4. X Port — Dropping Incoming Packets

Any packet arriving on the X-port is accounted for, packets are never silently dropped, and one of the special registers listed below will account for each dropped packet:

- Because of the 8b/10b encoding used on the serial links (Serial Myrinet), link bit errors appear as 8b/10b code errors. The code errors may cause a single incoming packet to appear as several packets or no packets at all. In any case, all code errors are counted by the 8B10B.ERROR register (page 41). If this register is not incrementing, the port is receiving properly formatted packets.
- If a packet is shorter than 5 bytes, it is dropped and the SHORT_DROP register (page 41) is incremented.
- If there is no space in the receive buffer, the packet is dropped and the BUFFER_DROP register (page 15) is incremented.
- If there are not enough memory cycles to write the packet into the memory, the packet may be dropped or corrupted and the MEMORY_DROP register (page 15) is incremented. If the packet is corrupted (not dropped), the corresponding packet descriptor will have its DESC_INVALID (page 13) bit set.

9.6. Received-Packet Descriptor (RPD)

The received-packet descriptors are always aligned on a double-word boundary, and consist of the following 64 bits.

BITS	NAME	DESCRIPTION
63-32	DESC_POINTER	Points to the beginning of the packet. Every packet starts at a receive-buffer-block boundary.
31	DESC_INVALID	This bit is equal to 1 if the packet has been corrupted because of lack of available memory cycles (page 52). The special registers port_MEMORY_DROP (page 15) count these packets. This bit should never be equal to 1 during regular operation; it is an indication of improperly-initialized memory-bus arbitration, or of numerous exceedingly-small packets arriving on the port. See page 52 for a discussion on memory arbitration and how it affects the packet interface.
30	DESC_LINK	If the port operates in Myrinet or in Gigabit Ethernet mode, this bit is never equal to 1. When the port operates in InfiniBand mode, this bit is equal to 1 if the packet is a Link Packet. These packets are typically diverted to the alternate channel (page 17).
29	DESC_MARKED_BAD	If the port operates in Myrinet mode, this bit is never equal to 1. If the port operates in Gigabit Ethernet mode, this bit is equal to 1 if the packet is marked with ERROR or EXTEND_ERROR (IEEE 802.3 §35.2.2). These packets should be treated by the MAC layer as if they had arrived with a CRC-32 error(IEEE 802.3 §35.2.1.5). If the port operates in InfiniBand mode, this bit is equal to 1 when the packet is marked bad by the port circuitry. This occurs if a packet ends with EBP, or if a packet-format error is detected after a portion of the packet has already been forwarded to the receive-DMA. Either the InfiniBand Remote Error Counter (page 44) or the Local Error Counter (page 44) is incremented by the port circuitry, and these packets should be silently dropped.
28	DESC_LAST	This bit is equal to 1 if the packet is the last packet in a receive buffer.
27-25	DESC_ZEROES	This field specifies how many trailing bytes (0-7) of the packet are equal to 0. It can be used to speed-up incoming-packet CRC verification, as described below. The communication protocols supported by the Lanai X chip (Myrinet, Gigabit Ethernet, InfiniBand) use a combination of CRC-32, CRC-16, and CRC-8, to protect data integrity. Configuring the port (page 38) selects the appropriate CRC computation(s). Regardless of the protocol used, the CRC bytes are always trailing packet bytes. When a packet is written into the receive buffer, the CRC(s) embedded in the packet are XOR-ed with the CRC(s) computed by the receiver. Therefore, every passing CRC check results in zero-bytes written into the receive buffer instead of the original CRC. Any failing CRC check results in less than the required number of zero-bytes at the end of the packet.
24-0	DESC_LENGTH	Length of the received packet in bytes (including all CRC bytes).

9.7. Receiver Special Registers

The following special registers control the packet-interface receivers.

If a register is named `port_NAME`, it represents two independent registers, `P0_NAME` and `P1_NAME`.

If a register is named `channel_NAME`, it represents four independent registers: `P0_NAME`, `P0_A_NAME`, `P1_NAME`, and `P1_A_NAME`.

A port is in one of four modes (Myrinet, Gigabit Ethernet, InfiniBand, GMII), as specified by the port special register `port_MODE` (page 34), except for Port 1 in the Lanai XM, which is always in Myrinet mode.

If a bit within a register is not applicable to some port/channel/mode, writing such a bit has no effect, and reading it returns 0.

REGISTER	DESCRIPTION						
channel_RB[31:0]	<p>Receive Buffer: Writing a receive-buffer pointer into this register issues the buffer to a receive channel.</p> <p>A receive channel has up to two receive buffers, the current receive buffer, and the backup receive buffer. The bits channel_NO_BUFFER and channel_NO_BACKUP in special registers ISR and AISR specify how many buffers the channel has. When the available space in the current buffer drops below maximum packet length (page 18), the backup buffer (if any) becomes the current buffer, and channel_NO_BACKUP becomes 1. If the channel_RB register is written when the channel_NO_BACKUP bit is equal to 0, the resulting behavior is undefined.</p> <p>Reading channel_RB is typically not required; the information on the received packets is available in RPL. However, if a program is reacting to a channel_PACKET_HEAD bit (page 17, 46), while the receive DMA is possibly still going on, reading channel_RB is the way to get a pointer to the packet currently being received:</p> <table border="1" data-bbox="841 709 1193 814"> <thead> <tr> <th data-bbox="847 718 945 745">BITS</th> <th data-bbox="945 718 1187 745">NAME</th> </tr> </thead> <tbody> <tr> <td data-bbox="847 745 945 779">31</td> <td data-bbox="945 745 1187 779">DESC_COMPLETE</td> </tr> <tr> <td data-bbox="847 779 945 812">30-0</td> <td data-bbox="945 779 1187 812">DESC_POINTER</td> </tr> </tbody> </table> <p>The DESC_COMPLETE bit mirrors the state of the channel_PACKET_RCVD (page 46) bit of ISR:</p> <p>If DESC_COMPLETE is equal to 1, the packet that caused channel_PACKET_HEAD has already been received, DESC_POINTER is pointing to some later packet, and the original-packet pointer must be fetched from RPL.</p> <p>If DESC_COMPLETE is equal to 0, DESC_POINTER points to the (still-incomplete) packet that caused channel_PACKET_HEAD.</p>	BITS	NAME	31	DESC_COMPLETE	30-0	DESC_POINTER
BITS	NAME						
31	DESC_COMPLETE						
30-0	DESC_POINTER						
channel_RPL_INDEX[15:0]	<p>Received-Packet-List Index: This read-only register is the index (0 through $N - 1$) into the current RPL. It is incremented every time an RPD is written into RPL, and it is reset to 0 when the receiver switches to a new receive buffer.</p>						
channel_RECV_COUNT[15:0]	<p>Received-Packet-List Counter: This read-only register counts the number of packets in all RPLs. It is incremented every time the receiver writes an RPD to the current RPL. Writing an arbitrary value to this register decrements it; it should be written every time an RPD is consumed from RPL.</p> <p>The programmer must assure that the channel_RECV_COUNT does not overflow, by controlling the rate at which the receive buffers are issued and consumed.</p>						
channel_RPL_SNAPSHOT[31:0]	<p>Received-Packet-List Snapshot: channel_RPL_INDEX is the most-significant half of this register, and channel_RECV_COUNT is its least-significant half. Reading channel_RPL_SNAPSHOT returns a consistent state of its two constituents.</p>						
channel_BUFFER_DROP[15:0]	<p>Packets Dropped for Lack of Receive Buffers Counter: This register is a read-only, reset-on-read, saturating counter (page 7). It is incremented every time a packet is dropped for lack of a receive buffer (ENABLE_BUFFER_DROP, page 17). See page 11 for a discussion on buffer-allocation policy.</p>						
port_MEMORY_DROP[15:0]	<p>Packets Dropped for Lack of Memory Cycles Counter: This register is a read-only, reset-on-read, saturating counter (page 7). It is incremented every time a packet is dropped or corrupted (INVALID, page 13) for lack of memory cycles. See page 52 for a discussion on memory arbitration and how it affects the packet interface. If the register corresponds to a SAN port (Port 1 in Lanai XM), it is never incremented.</p>						

REGISTER	DESCRIPTION		
channel_RPL_CONFIG[31:0]	Configuration: This register configures a receive channel:		
	BITS	NAME	DESCRIPTION
	31	PORT_ENABLE	This bit exists only in regular-channel registers. When the PORT_ENABLE bit is equal to 0 for a minimum of 10 clock cycles, the port is reset. Setting this bit to 1 enables the port (send channel and both receive channels).
	30-29		Reserved.
28	ENABLE_PAUSE_TIMER	This bit is applicable only to regular channels. When a valid Gigabit Ethernet MAC Control PAUSE packet is received, the port_PAUSE_COUNT is incremented (page 23). In addition, if the ENABLE_PAUSE_TIMER bit is equal to 1, the IT0 timer (for Port 0) or IT1 timer (for Port 1) is set to 0. This feature can be used to implement the flow-control when the port is operating in Gigabit-Ethernet mode.	
Continues on the next page.			

REGISTER	DESCRIPTION		
channel_RPL_CONFIG[31:0]	Continued from the previous page:		
	BITS	NAME	DESCRIPTION
	27	ENABLE_BUFFER_DROP	<p>This bit exists only in regular-channel registers, and applies to both channels of the port.</p> <p>In case of an X port, this bit must be set to 1, specifying that when there are no receive buffers, the incoming packets are to be dropped.</p> <p>In case of a SAN port, this bit must be set to 0, specifying that when there are no receive buffers, the incoming packets should be blocked in the network.</p>
	26	MATCH_LINK	<p>This bit is applicable only to regular channels. If it is equal to 1, the InfiniBand link packets are diverted to the alternate channel.</p>
	25	MATCH_OR	<p>This bit is applicable only to regular channels. If it is equal to 1, packets whose first byte has at least one of its four most-significant bits set are diverted to the alternate channel.</p> <p>This match does not apply to any of the currently-supported communication protocols.</p>
	24	MATCH_VL15	<p>This bit is applicable only to regular channels. If it is equal to 1, packets whose first byte has all four most-significant bits set are diverted to the alternate channel.</p> <p>This match can be used to select InfiniBand VL15 packets.</p>
	23-16	HEADER_LENGTH	<p>This field is applicable only to regular channels. It specifies the packet-header length in double-words. The only use of HEADER_LENGTH is as an early warning of an incoming packet, while the receive DMA is still going on.</p> <p>When the receiver has written HEADER_LENGTH double-words of an incoming packet into the current receive buffer, the channel_PACKET_HEAD bit of the special register ISR is set. This bit is reset when the entire packet has been written.</p> <p>If HEADER_LENGTH is equal to 0, the channel_PACKET_HEAD bit is always 0.</p>
Continues on the next page.			

REGISTER	DESCRIPTION		
channel_LRPL_CONFIG[31:0]	Continued from the previous page:		
	BITS	NAME	DESCRIPTION
	15-12	STOP_SIZE	<p>This field is applicable only to regular channels of X ports operating in Myrinet mode. In this mode, the STOP/GO flow-control symbols are generated by the hardware, without CPU assistance (page 11).</p> <p>The field specifies SIZE according to the size table below. If the available buffer space in the current receive buffer drops below SIZE, and there is no backup buffer, a STOP control symbol is generated and the port_MYRI_STOP bit of the special register AISR is set (page 47). When the next receive buffer is provided, a GO control symbol is generated, and the port_MYRI_STOP bit is cleared.</p> <p>If BUFFER_SIZE is smaller than STOP_SIZE, the resulting behavior is undefined.</p>
	11-8	MAX_PACKET	<p>This field specifies LENGTH according to the size table below. Each received packet longer than LENGTH is truncated; only the first LENGTH bytes are written into the receive buffer.</p>
	7-4	BLOCK_SIZE	<p>This field specifies B, <i>i.e.</i>, the size of receive-buffer blocks, according to the size table below.</p> <p>If the selected B is greater than the selected $N \times B$, the resulting behavior is undefined.</p>
3-0	BUFFER_SIZE	<p>This field specifies $N \times B$, <i>i.e.</i>, the size of receive buffers, according to the size table below.</p> <p>If the selected B is greater than the selected $N \times B$, the resulting behavior is undefined.</p>	

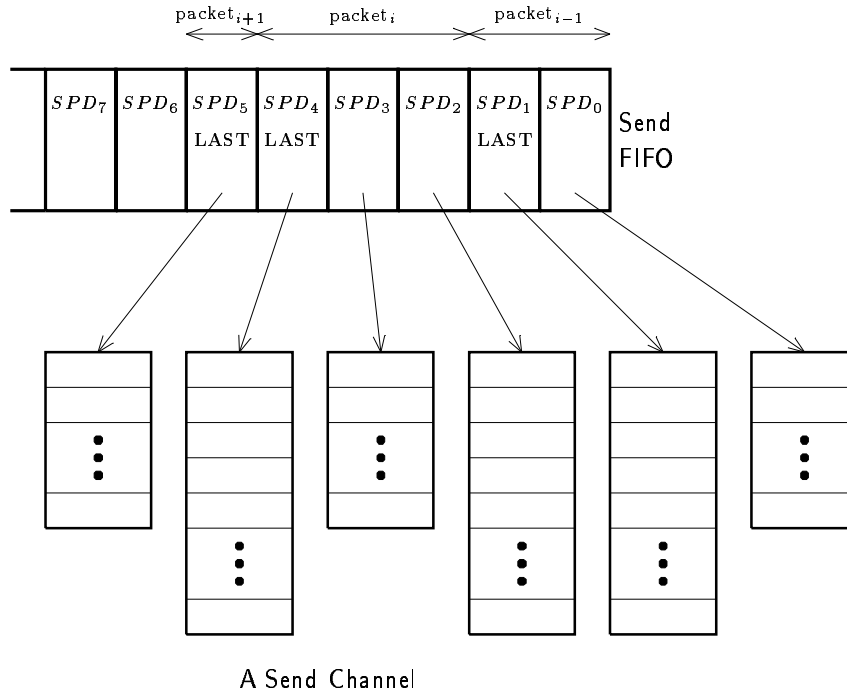
STOP_SIZE MAX_PACKET BUFFER_SIZE				BLOCK_SIZE			
Value	Size	Value	Size	Value	Size	Value	Size
0	512B	8	128KB	0	8B	8	2KB
1	1KB	9	256KB	1	16B	9	4KB
2	2KB	10	512KB	2	32B	10	8KB
3	4KB	11	1MB	3	64B	11	16KB
4	8KB	12	2MB	4	128B	12	32KB
5	16KB	13	4MB	5	256B	13	64KB
6	32KB	14	8MB	6	512B	14	128KB
7	64KB	15	16MB	7	1KB	15	256KB

10. PACKET-INTERFACE SENDER

There is one send-DMA engine (*sender*) on the chip for each network port. A packet is read from memory and emitted into the network by the corresponding sender.

Each sender maintains an 8-deep FIFO of Send-Packet-segment Descriptors (SPDs). An SPD consists of a pointer to the start of a packet segment, the segment length, and some additional flags, including a flag that marks the last segment of a packet (page 20).

A sender FIFO with its associated control/status registers is called a send *channel*.



A packet is specified by writing a set of SPDs into the channel send FIFO (page 22). If the FIFO is written when it is full (as reported by the special register port_SEND_FREE_COUNT, page 22), the resulting behavior is undefined.

The packet is emitted into the network only after the last SPD of a packet has been written, and, if sufficient memory bandwidth has been allocated to the channel (page 52), the packet is guaranteed to be contiguous (Gigabit Ethernet and InfiniBand require that packets be contiguous, Myrinet does not).

Depending on the mode the port is in, and on the state of the configuration register (page 40), the appropriate CRC values will be computed and appended to the packet.

10.1. Cut-Through Packet Forwarding

The senders provide hardware support for cut-through packet forwarding. To forward a packet, the last SPD of the packet is marked to indicate that the block of data it describes is potentially still being written into the memory. An additional, special SPD is provided, instructing the sender to verify the length, CRCs, and some additional flags of the incoming packet, and, if the verification fails, to invalidate the outgoing packet (page 39).

10.2. Send-Packet-Segment Descriptor (SPD)

The send-packet-segment descriptors consist of the following 64 bits.

BITS	NAME	DESCRIPTION
63-32	DESC_POINTER	Points to the beginning of the packet segment.
31	DESC_CUT_THROUGH	This bit is applicable only for the last SPD of a packet, and, when it is equal to 1, this SPD describes a memory buffer that is potentially still being written into the memory by a receiver. An additional, special, cut-through SPD (page 21) must follow this SPD, instructing the sender how to verify the still-incoming packet, and to invalidate the outgoing packet, if it fails the verification.
30	DESC_ROUTE/ DESC_LINK	When the port is in Myrinet mode, if this bit is equal to 1, the packet segment specified by this SPD contains a Myrinet route that will be stripped by the switches, and is not to be included in the CRC-32 computation. When the port is in InfiniBand mode, and only in the first SPD of a packet, if this bit is equal to 1, the outgoing packet is the InfiniBand Link packet. When the port is in Gigabit Ethernet mode, this bit must be equal to 0.
29	DESC_INVALIDATE	This bit is applicable only for the last SPD of a packet. If it is equal to 1, it requests that the outgoing packet be invalidated, as specified by the port_SEND_CONFIG register (page 39).
28	DESC_LAST	This bit marks the last segment of a packet.
27-25		Reserved.
24-0	DESC_LENGTH	Length of the send packet segment in bytes. If DESC_LENGTH is equal to 0, the resulting behavior is undefined.

NOTE:

Firmware for all Lanai X chips should adhere to the following restrictions:

- DESC_POINTER must point to an 8-byte-aligned address.
- A valid packet can be specified by a maximum of one SPD with its DESC_ROUTE set, followed by any number of SPDs with their DESC_ROUTE not set. The DESC_LENGTH of each of the latter SPDs must be a multiple of 8 bytes; the sum of DESC_LENGTHs of all of the latter SPDs must be at least 16.

In some cases, it is possible to relax the above restrictions, but the detailed description is beyond the scope of this document.

10.3. Cut-Through SPD

If the CUT_THROUGH bit of the last SPD of a packet is equal to 1, this special SPD must also be written, instructing the sender where the *matching* received-packet descriptor (RPD) of the potentially still-incoming packet is, and how to check it to verify the incoming packet. If the incoming-packet verification fails, the outgoing packet will be invalidated, as specified on page 39.

BITS	NAME	DESCRIPTION
63-32	DESC_POINTER	Points to the length word of the matching RPD. By the time sender has to complete the outgoing packet (and possibly invalidate it), the information on the incoming packet must be in the matching RPD. If both the receive and the send channel used in a cut-through operation have sufficient memory bandwidth (page 52), this condition will be satisfied, except when a possibly corrupted incoming packet is longer than specified in its header. To deal with this case, the length word of all RPDs that might be used as matching RPDs should be pre-invalidated (page 13).
31	DESC_INVALID	If this bit is equal to 1, the sender must verify that the INVALID bit of the matching RPD is equal to 0.
30	DESC_LINK	If this bit is equal to 1, the sender must verify that the LINK bit of the matching RPD is equal to 0.
29	DESC_MARKED_BAD	If this bit is equal to 1, the sender must verify that the MARKED_BAD bit of the matching RPD is equal to 0.
28		Reserved.
27-25	DESC_ZEROES	This field specifies what the minimum value of the ZEROES field in the matching RPD should be, to be verified by the sender.
24-0	DESC_LENGTH	This field specifies what the value of the LENGTH field in the matching RPD should be, to be verified by the sender.

10.4. Sender Special Registers

The following special registers control the packet-interface senders.

If a register is named `port_NAME`, it represents two independent registers, `P0_NAME` and `P1_NAME`.

A port is in one of four modes (Myrinet, Gigabit Ethernet, InfiniBand, or GMII), as specified by the port special register `port_MODE` (page 34), except for Port 1 in the Lanai XM, which is always in Myrinet mode.

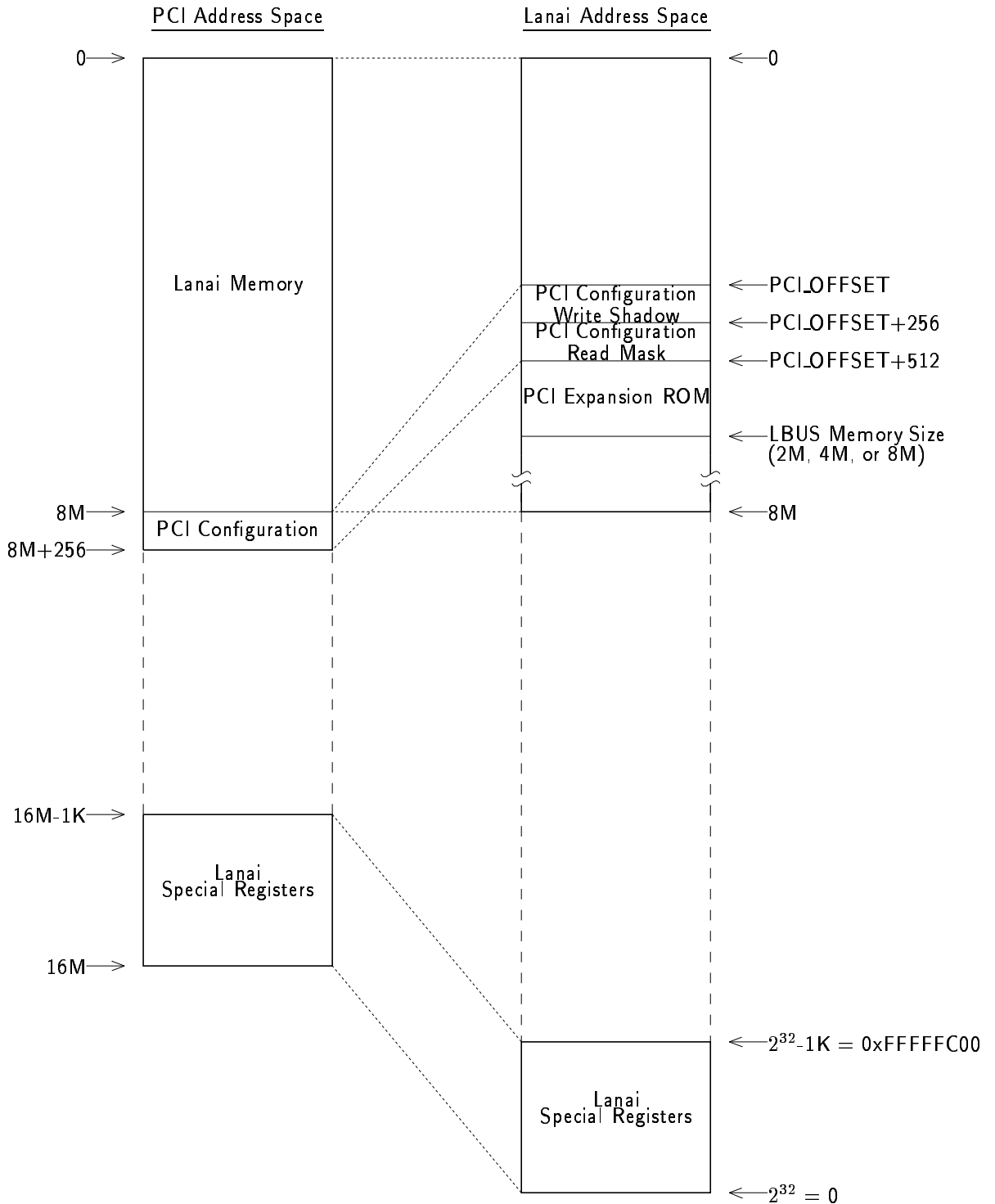
REGISTER	DESCRIPTION
<code>port_SEND[31:0]</code>	Port Send FIFO: This is a write-only register. The send-packet descriptors (SPDs) (page 20) are written into this register to specify the outgoing packet; the pointer word of an SPD must be written first, followed by the length/flags word. If <code>port_SEND</code> is written when the <code>port_SEND_FREE_COUNT</code> is equal to 0, the resulting behavior is undefined.
<code>port_SEND_FREE_COUNT[4:0]</code>	Port Send-FIFO Free Counter: This read-only register specifies how many words (0-16) may be written into the <code>port_SEND</code> register (an SPD is a double-word). If <code>port_SEND</code> is written when the <code>port_SEND_FREE_COUNT</code> is equal to 0, the resulting behavior is undefined.
<code>port_SEND_FREE_LIMIT[3:0]</code>	Port Send-FIFO Free Limit: This register has its least significant bit hard-wired to 0, and it specifies the threshold (0-14) for setting the <code>port_SEND_READY</code> bit of the special register ISR. When $\text{port_SEND_FREE_COUNT} \geq \text{port_SEND_FREE_LIMIT} + 2$, <code>port_SEND_READY</code> bit is equal to 1.
<code>port_SEND_COUNT[7:0]</code>	Port Send-Packet Counter: This register counts the number of packets sent. It is incremented every time the sender emits the data requested by the last descriptor of a packet (including the cut-through descriptor, if applicable). Writing an arbitrary value to this register decrements it; it should be written every time a set of SPDs corresponding to a single packet is reclaimed from the sender, to assure that it does not overflow.

REGISTER	DESCRIPTION										
port_PAUSE_COUNT[7:0]	<p>Port Pause Counter: This register is incremented every time a valid Gigabit Ethernet MAC Control PAUSE packet is received. In this context, a valid packet means a packet with Length/Type field equal to 88-08, Mac Control Opcode equal to 00-01, correct CRC-32, not INVALID, and not MARKED_BAD (page 13).</p> <p>When port_PAUSE_COUNT is not equal to 0, the port_PAUSE_RCVD bit of the special register AISR is set (page 47), and, depending on the state of the port_SEND_CONTROL register (described next), the packet sending may be stopped. This counter saturates at 0xFF; it does not overflow. Writing an arbitrary value to this register decrements it; it should be written every time a PAUSE packet has been consumed.</p>										
port_SEND_CONTROL[1:0]	<p>Port Send Control: The value of this register controls the packet transmission:</p> <table border="1" data-bbox="539 655 1479 1465"> <thead> <tr> <th data-bbox="539 655 646 688">Value</th> <th data-bbox="646 655 1479 688">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="539 688 646 793">0</td> <td data-bbox="646 688 1479 793">SEND_ON: Packet sending is not affected by the port_PAUSE_COUNT. This is the typical mode of operation when the chip is in Myrinet or InfiniBand mode.</td> </tr> <tr> <td data-bbox="539 793 646 1060">1</td> <td data-bbox="646 793 1479 1060">SEND_PAUSE: If port_PAUSE_COUNT is equal to 0, packet sending is not affected; if it is non-zero, the current packet is completed (if any), and no more packets are sent. This mode of operation can be used to implement flow-control in full-duplex Gigabit Ethernet mode: Arrival of a valid MAC Control PAUSE packet stops packet transmission (except for the current packet, if any), and, once the pause timeout expires, the transmission should be re-enabled by decrementing the port_PAUSE_COUNT.</td> </tr> <tr> <td data-bbox="539 1060 646 1138">2</td> <td data-bbox="646 1060 1479 1138">SEND_OFF: The current packet is completed (if any), and no more packets are sent.</td> </tr> <tr> <td data-bbox="539 1138 646 1465">3</td> <td data-bbox="646 1138 1479 1465">SEND_FLUSH: The current packet is completed (if any), all other packets are flushed from the send queue, and no more packets are sent. This feature can be used when operating in full-duplex Gigabit Ethernet mode. The flow control does not apply to MAC Control packets: if a request is posted to send such a packet while the transmission is disabled, the send queue should be flushed (see P0_SEND_STOPPED on page 47 for flush-completion detection), the transmission enabled (by writing SEND_ON into this register), and the MAC Control packet sent. Resending the flushed packets upon expiration of the pause timeout is the responsibility of the programmer.</td> </tr> </tbody> </table>	Value	Description	0	SEND_ON: Packet sending is not affected by the port_PAUSE_COUNT. This is the typical mode of operation when the chip is in Myrinet or InfiniBand mode.	1	SEND_PAUSE: If port_PAUSE_COUNT is equal to 0, packet sending is not affected; if it is non-zero, the current packet is completed (if any), and no more packets are sent. This mode of operation can be used to implement flow-control in full-duplex Gigabit Ethernet mode: Arrival of a valid MAC Control PAUSE packet stops packet transmission (except for the current packet, if any), and, once the pause timeout expires, the transmission should be re-enabled by decrementing the port_PAUSE_COUNT.	2	SEND_OFF: The current packet is completed (if any), and no more packets are sent.	3	SEND_FLUSH: The current packet is completed (if any), all other packets are flushed from the send queue, and no more packets are sent. This feature can be used when operating in full-duplex Gigabit Ethernet mode. The flow control does not apply to MAC Control packets: if a request is posted to send such a packet while the transmission is disabled, the send queue should be flushed (see P0_SEND_STOPPED on page 47 for flush-completion detection), the transmission enabled (by writing SEND_ON into this register), and the MAC Control packet sent. Resending the flushed packets upon expiration of the pause timeout is the responsibility of the programmer.
Value	Description										
0	SEND_ON: Packet sending is not affected by the port_PAUSE_COUNT. This is the typical mode of operation when the chip is in Myrinet or InfiniBand mode.										
1	SEND_PAUSE: If port_PAUSE_COUNT is equal to 0, packet sending is not affected; if it is non-zero, the current packet is completed (if any), and no more packets are sent. This mode of operation can be used to implement flow-control in full-duplex Gigabit Ethernet mode: Arrival of a valid MAC Control PAUSE packet stops packet transmission (except for the current packet, if any), and, once the pause timeout expires, the transmission should be re-enabled by decrementing the port_PAUSE_COUNT.										
2	SEND_OFF: The current packet is completed (if any), and no more packets are sent.										
3	SEND_FLUSH: The current packet is completed (if any), all other packets are flushed from the send queue, and no more packets are sent. This feature can be used when operating in full-duplex Gigabit Ethernet mode. The flow control does not apply to MAC Control packets: if a request is posted to send such a packet while the transmission is disabled, the send queue should be flushed (see P0_SEND_STOPPED on page 47 for flush-completion detection), the transmission enabled (by writing SEND_ON into this register), and the MAC Control packet sent. Resending the flushed packets upon expiration of the pause timeout is the responsibility of the programmer.										

11. PCI INTERFACE

The Lanai XP and 2XP chips include a PCI-X interface. The Lanai memory and special registers, PCI Configuration Registers, and PCI Expansion ROM are all mapped into 16MB of regular, PCI Memory Space. The PCI I/O Space is not used.

As illustrated below, the first 8MB are mapped into the Lanai memory, the next 256 bytes into the PCI Configuration Space, and the last 1KB into the Lanai special registers (page 7). The PCI Expansion ROM can be accessed through the Lanai memory. See page 32 for a description of PCIOFFSET.



Only a fraction of the PCI Configuration Space state bits are implemented in hardware (page 25); the rest reside in the PCI Configuration Read Mask region of the Lanai memory (page 24). In typical configuration, a PCI Configuration Read access returns the value that is the bit-wise OR of the hardware-implemented bits and the corresponding value in the Read Mask in memory. This memory is typically initialized by the Lanai processor upon PCI reset, by copying it from EEPROM (page 27).

PCI Configuration Write accesses have a side effect of writing into the PCI Configuration Write Shadow region in the Lanai memory (page 24).

Note that the Lanai CPU is big-endian, and the PCI Configuration Space is little-endian (PCI §6.1).

The Base Address Registers (PCI §6.2.5) are configured such that each Lanai-based interface requests 16MB of memory, located anywhere in 64-bit address space. If necessary, for compatibility with some non-conforming PCI implementations, the Base Address Register can be forced into the 32-bit-only mode; contact Myricom for details.

The PCI Expansion ROM Base Address Register (PCI §6.2.5.2) maps 512KB of Expansion ROM (PCI §6.3).

The only Lanai-specific Configuration Space register is at byte addresses 43–40, defined below. Writing 0 into any of its bits has no effect on the Lanai operation.

BITS	NAME	DESCRIPTION
31-24		Bits 31-24 are arranged in <code>_ON/_OFF</code> pairs, each pair corresponding to a hardware-control signal. Writing a 1 into an <code>_ON</code> bit sets the corresponding signal, writing a 1 into an <code>_OFF</code> bit clears it. If 1 is written into both bits, the <code>_OFF</code> bit has precedence. Upon PCI reset, all control signals are OFF.
31 30 29 28	<code>PCI_CPU_RESET_ON</code> <code>PCI_CPU_RESET_OFF</code> <code>PCI_IO_RESET_ON</code> <code>PCI_IO_RESET_OFF</code>	There are two internal reset signals that control the operation of a Lanai chip: <code>CPU_RESET</code> , which resets the CPU, and <code>IO_RESET</code> , which resets most of the remaining non-PCI Lanai circuitry. <code>CPU_RESET</code> = <code>PCI_CPU_RESET</code> or <code>JTAG_CPU_RESET</code> <code>IO_RESET</code> = <code>PCI_IO_RESET</code> or <code>JTAG_IO_RESET</code> Upon hardware reset (controlled by the <code>RST</code> pin), <code>PCI_RESET</code> signals are OFF, enabling the JTAG interface to control the chip. Once a Lanai XP/2XP-based interface has been properly configured, the <code>JTAG_RESET</code> signals are OFF, and the chip can be controlled through the PCI interface. Consult the Initialization section on page 27 for additional details.
27 26	<code>PCI_INT_ENABLE_ON</code> <code>PCI_INT_ENABLE_OFF</code>	The <code>REQ_ACK[0]</code> bit of the special register <code>ISR</code> (page 45) is used as a PCI-interrupt request bit. The <code>PCI_INT_ENABLE</code> signal determines if the interrupt-request bit causes a PCI interrupt.
25 24	<code>PCI_MASK_DISABLE_ON</code> <code>PCI_MASK_DISABLE_OFF</code>	During normal operation, the <code>PCI_MASK_DISABLE</code> signal must be OFF, and PCI Configuration Read access returns the value that is the bit-wise OR of the hardware-implemented bits, and the corresponding value in the Read Mask in memory. The <code>PCI_MASK_DISABLE</code> bit may be ON only for debugging purposes; when it is ON, the Read Mask in memory is not used during for PCI Configuration Read accesses.
23-0	<code>PCI_JTAG_MASTER</code>	The details of the JTAG interface are beyond the scope of this document; contact Myricom for details.

11.2. PCI-Interface Initialization

This section describes the typical initialization sequence of the chips with the PCI/PCI-X interface.

The main chip reset is the \overline{RST} pin, which should be connected to PCI RST# signal. The \overline{RST} pin must be asserted for a minimum of 32 PCI clocks to reset the chip.

When the \overline{RST} pin is deasserted, the EEPROM interface section of the chip (page 53) starts reading the attached EEPROM and executing the pseudo-code it contains. Typically, the pseudo code loads the initial CPU program into the Lanai memory (through the JTAG interface), and then releases JTAG_IO_RESET and JTAG_CPU_RESET. While this is going on, the PCI section of the chip is RETRYing any PCI accesses that may be requested.

The initial CPU code uses the special register PCI_CLOCK (page 31) to determine the clock frequency of the PCI bus, and configures the synchronization by setting the PCISYNCH and PCIDELAY fields of the special register DMA_CONFIG (page 32). It also determines the available size of the LBUS memory, sets the PCI_OFFSET field of DMA_CONFIG as shown on page 24, and initializes the PCI Configuration Read Mask region and the PCI Expansion ROM region of the Lanai memory.

Having completed the required PCI Interface configuration, the CPU enables it through the JTAG interface (page 53), after which the PCI Interface starts responding to system-configuration requests. This initialization process must complete in less than 2^{25} PCI clocks (PCI §4.2.3.2).

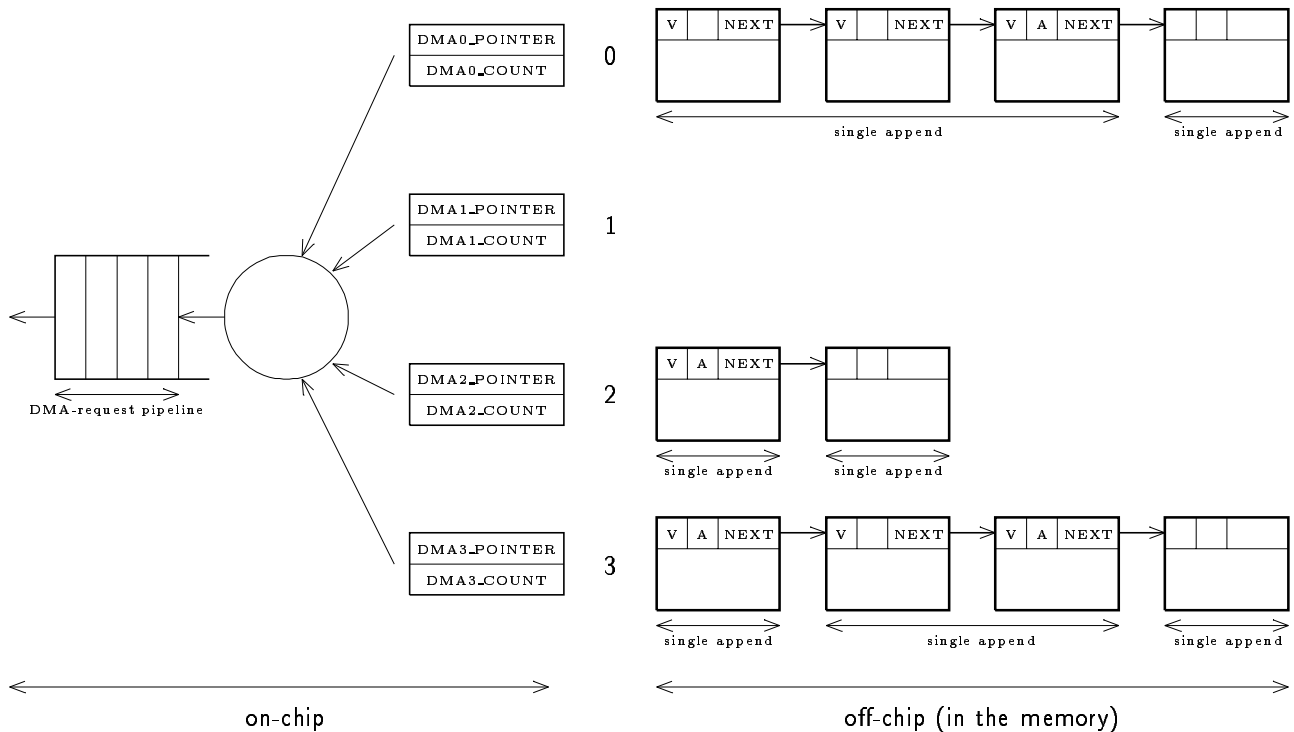
12. PCI DMA INTERFACE

The PCI-DMA engine has four DMA channels, $i = 0, 1, 2, 3$, each controlled by a linked list of DMA-request descriptors.

After a new descriptor (or a linked-list of descriptors) has been appended to a channel's linked list in memory, the pointer to the descriptor (only to the first one, if a linked list was appended) must be written to the corresponding special register DMA $_i$ _POINTER.

See page 29 for a detailed description of DMA-request descriptors. For purposes of this section, we will discuss only the enqueueing/dequeueing issues. Each descriptor contains the NEXT pointer and its two associated bits: VALID and APPEND. Note that the NEXT, VALID, and APPEND bits are all parts of the same 32-bit word, and are therefore accessed simultaneously; the VALID and APPEND bits refer to the NEXT pointer, not the state of the descriptor. The VALID bit indicates that its NEXT pointer is valid, *i.e.*, that this is not the last descriptor in the list. The APPEND bit indicates that its NEXT pointer has been (or will be) written to the DMA $_i$ _POINTER; any pre-linked NEXT pointers (the ones not written to DMA $_i$ _POINTER) must not have this bit set.

Note that an empty linked list is a valid configuration; when a DRD is added, the DMA engine will be notified by writing the DMA $_i$ _POINTER.



The completion of a DMA request on channel i is signaled by incrementing the special register DMA $_i$ _COUNT, but only for the DMA descriptors marked with the END bit (not shown in the above diagram). Writing any value into DMA $_i$ _COUNT decrements it; it should be written every time a PCI-DMA descriptor is reclaimed from the channel link list.

There are four DMA $_i$ _DONE bits in the special register ISR (page 45), and each of these bits is equal to 1 when the corresponding DMA $_i$ _COUNT registers is non-zero.

The four DMA channels can have four priority levels, they can be served in round-robin fashion, or a combination of the two (page 32). If two or more channels are configured to be served in round-robin fashion, the DMA engine will switch to the next channel with the same priority level only upon servicing a descriptor with its END bit set.

The PCI-DMA engine can be configured to pipeline the DMA requests, with the depth of the pipeline configurable from 1 to 4 (1 corresponds to no pipelining). The channel requests are prioritized at the pipeline entrance, so although increasing the pipeline depth typically increases the I/O bus utilization, it also increases the maximum delay a higher-priority request may encounter because of the lower-priority requests already in the pipeline.

12.1. PCI DMA-Request Descriptor (DRD)

PCI-DMA descriptors consist of three double-words, and must be double-word (*i.e.* 8-byte) aligned.

DOUBLE-WORD 0		
BITS	NAME	DESCRIPTION
63	DMA_READ	If this bit is equal to 1, the direction of the DMA transfer is from some other device on the PCI bus to the Lanai.
62	DMA_END	If this bit is equal to 1, upon completion of the PCI-DMA transfer, the DMA _i _COUNT register for the corresponding channel is incremented. If two or more DMA channels are at the same priority level (page 32), the DMA engine will switch to the next channel with the same priority level only upon servicing a descriptor with its DMA_END bit set.
61	DMA_FLUSH	If this bit is equal to 1, the PCI-DMA engine will wait until all outstanding DMAs (including this one) have been completed before continuing to load the next DMA request. This feature can be used, for example, if there are data dependencies between two DMAs.
60	DMA_NO_SNOOP	Setting this bit to 1 enables the PCI-X "no-snoop" feature. Experimental.
59	DMA_NO_CACHE	Setting this bit to 1 enables the PCI-X "no-cache" feature. Experimental.
58-56		Reserved.
55-32	DMA_LENGTH	Length of the DMA transfer in bytes. If DMA_LENGTH is zero, the DMA engine completes all the requests that have already been inserted into the pipeline, then completes the zero-length request, too (without any activity on the PCI bus). The checksum (page 30) is not written.
31-0	DMA_PCI_ADDR	The least significant 32 bits of the DMA-transfer PCI address.

DOUBLE-WORD 1		
BITS	NAME	DESCRIPTION
63-60		Reserved.
59-32	DMA_LANAI_ADDR	The DMA-transfer Lanai address.
31-0	DMA_PCI64_ADDR	The most significant 32 bits of the DMA-transfer PCI address. If the system is using 32-bit PCI addresses, this word must be set to 0.

DOUBLE-WORD 2		
BITS	NAME	DESCRIPTION
63-62		Reserved.
61	DMA_APPEND	This bit is associated with the DMA_NEXT pointer described below. It indicates that the DMA_NEXT pointer has been (or will be) written to the DMA _i _POINTER; any pre-linked DMA_NEXT pointers (the ones not written to DMA _i _POINTER) must not have this bit set.
60	DMA_VALID	This bit is associated with the DMA_NEXT pointer described below. It indicates that the DMA_NEXT pointer is valid, <i>i.e.</i> , that this is not the last descriptor in the list.
59-32	DMA_NEXT	Pointer to the next DMA-request descriptor. Since the descriptors must be aligned on an 8-byte boundary, bits 34-32 are ignored. Note that 0 is a valid pointer value.
31-16	DMA_OFFSET	A side effect of all PCI-DMA transfers is that the 16-bit 1's-complement sum is computed on the transferred byte stream, starting with the byte at offset DMA_OFFSET (relative to the start of the DMA transfer), and ending with the last transferred byte. The result is stored into the DMA_CHECKSUM field of the DMA-request descriptor (described next). If $\text{DMA_OFFSET} \geq \text{DMA_LENGTH}$, the resulting checksum is 0.
15-0	DMA_CHECKSUM	See the above definition of DMA_OFFSET. Upon completion of a PCI-DMA request, the computed value is written into the DMA_CHECKSUM field of the requesting DRD (except when DMA_LENGTH is equal to 0, as described on page 29).

12.2. PCI-DMA Special Registers

The following special registers control the PCI-DMA engine.

REGISTER	DESCRIPTION
DMA3_POINTER[31:0] DMA2_POINTER[31:0] DMA1_POINTER[31:0] DMA0_POINTER[31:0]	<p>PCI DMA Pointers: These registers correspond to the 4 PCI-DMA channels. When a DMA request (or a pre-linked list of DMA requests) has been appended to the channel linked list, the address of the first (and possibly the only) DMA request must be written to the DMA_i_POINTER.</p>
DMA3_COUNT[7:0] DMA2_COUNT[7:0] DMA1_COUNT[7:0] DMA0_COUNT[7:0]	<p>PCI DMA-Done Counters: These registers correspond to the 4 PCI-DMA channels. When a DMA request has completed, if its DMA_END bit is set, the DMA_i_COUNT register is incremented. Writing any value to this register decrements it; it should be written every time a PCI-DMA descriptor marked with DMA_END is reclaimed from the channel link list. If this register is greater or equal to 128, the corresponding DMA channel is disabled. During normal operation, this counter can reach up to 132 due to pipelining of DMA requests; once the register is decremented to less than 128, the DMA channel is re-enabled.</p>
PCLCLOCK[15:0]	<p>PCI Clock Counter: This register is used to determine the ratio of CPU to PCI clock frequencies. It counts PCI clocks during 2^{13} CPU clock cycles. The counter is activated when it is written, or when PCI reset is deasserted. The most-significant bit of this register is equal to 1 while the counting is going on, and 0 when the counting completes.</p>

REGISTER	DESCRIPTION		
DMA_CONFIG[31:0]	PCI Configuration: This register configures the PCI-DMA engine:		
	BITS	NAME	DESCRIPTION
	31	DMA_ENABLE	This read-only bit indicates that the PCI interface has enabled the DMA. This bit is the same as the "Bus Master" bit in the "Command" register of the PCI Configuration Space (page 25), synchronized to the CPU clock.
	30-28	DMA_PRIORITY	These bits set the priority of the 4 PCI-DMA channels. By default, these bits are equal to 0, and the four channels are prioritized from 0 (the highest) to 3 (the lowest). If bit 28 is set, channels 0 and 1 have the same priority. If bit 29 is set, channels 1 and 2 have the same priority. If bit 30 is set, channels 2 and 3 have the same priority. Note that setting all three bits results in the round-robin scheduling of the four DMA channels. If two or more channels are at the same priority level, the DMA engine will switch to the next channel with the same priority level only upon servicing a descriptor with its DMA_END bit set.
	27-10	PCI_OFFSET	These are bits 27-10 of the address of the PCI-configuration data in the Lanai LBUS memory (the rest of the bits are 0). See page 24 for details.
	9-8	DMA_PIPELINE	These bits configure the PCI-DMA engine pipeline depth. The DMA_PIPELINE is the number of DMA requests that may be issued before completing the original DMA request. A value of 0 implies no pipelining, <i>i.e.</i> , each DMA request must be completed before the next one is issued. The channel requests are prioritized at the pipeline entrance, so although increasing the pipeline depth typically increases the I/O bus utilization, it also increases the maximum delay a higher-priority request may encounter because of the lower-priority requests already in the pipeline.
	7-4	PCI_DELAY	Myricom-supplied configuration. This field is initialized upon PCI reset (page 27), and must not be modified by the programmer.
3-0	PCI_SYNC	Myricom-supplied configuration. This field is initialized upon PCI reset (page 27), and must not be modified by the programmer.	

13. COMMUNICATION PORTS

The communication ports operate in clock domains that are independent from the Lanai-core clock. As a result, the control/status registers associated with the port circuitry cannot be accessed using the standard, memory-mapped, special-register-access mechanism (page 7).

Instead, the port registers are accessed through the two port-access special registers described next.

13.1. Port-Access Special Registers

REGISTER	DESCRIPTION								
PORT_ADDR[15:0]	<p>Port Address: This register specifies the port-register address and the type of port-register access:</p> <table border="1" data-bbox="867 726 1154 867"> <thead> <tr> <th>BITS</th> <th>NAME</th> </tr> </thead> <tbody> <tr> <td>15</td> <td>PORT_READ</td> </tr> <tr> <td>14-6</td> <td>Reserved</td> </tr> <tr> <td>5-0</td> <td>PORT_REG</td> </tr> </tbody> </table> <p>To be able to accommodate the worst-case synchronization delay between the CPU and the port circuitry, we shall define MIN_DELAY to be 6 CPU cycles if the accessed port is in Myrinet or InfiniBand mode, and 12 CPU cycles if it is in Gigabit Ethernet mode. When PORT_ADDR is written with PORT_READ bit equal to 1, the port register specified by PORT_REG will be copied into the special register PORT_DATA after a maximum of (6 + MIN_DELAY) CPU cycles. When PORT_ADDR is written with PORT_READ bit equal to 0, no action is performed on any port register. After a minimum of MIN_DELAY CPU cycles, the PORT_DATA register may be written, and it will be copied into the port register specified by PORT_REG. After writing PORT_DATA, the PORT_ADDR must not be written for at least MIN_DELAY cycles.</p>	BITS	NAME	15	PORT_READ	14-6	Reserved	5-0	PORT_REG
BITS	NAME								
15	PORT_READ								
14-6	Reserved								
5-0	PORT_REG								
PORT_DATA[15:0]	<p>Port Data: This register is used to transfer data in and out of the port registers. When this register is read, the returned value is the value of the most-recently-read port register. When a value is written into this register, that value will be written into the port register specified by PORT_ADDR. See PORT_ADDR for timing restrictions.</p>								

The following port registers control the communication ports.

If a register is named port_NAME, it represents two independent registers, P0_NAME and P1_NAME.

A port is in one of four modes (Myrinet, Gigabit Ethernet, InfiniBand, or GMII), as specified by the port special register port_MODE described below, except for Port 1 in the Lanai XM, which is always in Myrinet mode.

If a bit within a register is not applicable to some port/mode, writing such a bit has no effect, and reading it returns 0.

13.2. Configuration Port Registers

PORT REGISTER	DESCRIPTION															
port_MODE[3:0]	Port Mode: This register selects the operating mode of the X port.															
	<table border="1"> <thead> <tr> <th>BITS</th> <th>NAME</th> <th>DESCRIPTION</th> </tr> </thead> <tbody> <tr> <td>3</td> <td>MODE_GMII</td> <td>If this bit is equal to 1, the X port operates in GMII mode, typically used in conjunction with an off-chip Gigabit Ethernet 1000BASE-T device.</td> </tr> <tr> <td>2</td> <td>MODE_GEX</td> <td>If this bit is equal to 1, the X port operates in Gigabit Ethernet 1000BASE-X mode.</td> </tr> <tr> <td>1</td> <td>MODE_IB</td> <td>If this bit is equal to 1, the X port operates in InfiniBand mode.</td> </tr> <tr> <td>0</td> <td>MODE_M</td> <td>If this bit is equal to 1, the X port operates in Myrinet mode.</td> </tr> </tbody> </table>	BITS	NAME	DESCRIPTION	3	MODE_GMII	If this bit is equal to 1, the X port operates in GMII mode, typically used in conjunction with an off-chip Gigabit Ethernet 1000BASE-T device.	2	MODE_GEX	If this bit is equal to 1, the X port operates in Gigabit Ethernet 1000BASE-X mode.	1	MODE_IB	If this bit is equal to 1, the X port operates in InfiniBand mode.	0	MODE_M	If this bit is equal to 1, the X port operates in Myrinet mode.
	BITS	NAME	DESCRIPTION													
	3	MODE_GMII	If this bit is equal to 1, the X port operates in GMII mode, typically used in conjunction with an off-chip Gigabit Ethernet 1000BASE-T device.													
	2	MODE_GEX	If this bit is equal to 1, the X port operates in Gigabit Ethernet 1000BASE-X mode.													
	1	MODE_IB	If this bit is equal to 1, the X port operates in InfiniBand mode.													
0	MODE_M	If this bit is equal to 1, the X port operates in Myrinet mode.														
It is not possible to write a value with more than a single bit equal to 1 into this register; only the least significant 1 will be written, all other bits will be cleared. If all the bits are equal to 0, the port is disabled.																

When port_MODE[3]=0, exactly one of port_MODE[2:0] bits is equal to 1, and port_TEST[4:0]=0 (page 35), we define CLSM to be the Currently-active Link-synchronization State Machine: Myrinet 2000 (Serial Myrinet), Gigabit Ethernet 1000BASE-X (IEEE 802.3 §37.3), or InfiniBand 1X (IAS §v2.5.7.4).

Myrinet 2000		Gigabit Ethernet 1000BASE-X		InfiniBand 1X	
STATE NAME	CODE	STATE NAME	CODE	STATE NAME	CODE
WE_LOST_SYNC	0	AN_ENABLE	0	Sleeping.Delay	1
THEY_LOST_SYNC	16	AN_RESTART	1	Sleeping.Quiet	9
REGAIN_SYNC	5	AN_DISABLE_LINK_OK	2	Polling.Active	2
REGAIN_SYNC_GAP	6	ABILITY_DETECT	3	Polling.Quiet	10
REGAIN_SYNC_STOP	13	ACKNOWLEDGE_DETECT	4	Disabled	3
REGAIN_SYNC_GO	9	COMPLETE_ACKNOWLEDGE	5	Config.Debounce	4
IN_SYNC_GO	10	IDLE_DETECT	6	Config.RcvrCfg	12
IN_SYNC_STOP	14	LINK_OK	7	Config.WaitRmt	20
		NEXT_PAGE_WAIT	8	Config.Idle	28
				LinkUp	5
				Recovery.ReTrain	6
				Recovery.WaitRmt	14
				Recovery.Idle	22

PORT REGISTER	DESCRIPTION																		
port_TEST[4:0]	<p>Port Test: This register is provided only for test measurements, and must not be accessed during regular operation. The register consists of the following bits:</p> <table border="1" data-bbox="532 365 1487 982"> <thead> <tr> <th data-bbox="532 365 659 415">BITS</th> <th data-bbox="659 365 980 415">NAME</th> <th data-bbox="980 365 1487 415">DESCRIPTION</th> </tr> </thead> <tbody> <tr> <td data-bbox="532 415 659 558">4</td> <td data-bbox="659 415 980 558">TEST_DEBUG</td> <td data-bbox="980 415 1487 558">If this bit is equal to 1, the link timeouts in InfiniBand and in Gigabit Ethernet mode are approximately 1,000 times shorter than specified.</td> </tr> <tr> <td data-bbox="532 558 659 663">3</td> <td data-bbox="659 558 980 663">TEST_DISCDET</td> <td data-bbox="980 558 1487 663">If this bit is equal to 1, the X port disables comma-detection on the incoming 8b/10b-encoded symbol stream.</td> </tr> <tr> <td data-bbox="532 663 659 768">2</td> <td data-bbox="659 663 980 768">TEST_MIX</td> <td data-bbox="980 663 1487 768">If this bit is equal to 1, the X port emits a stream of K28.5 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.3.</td> </tr> <tr> <td data-bbox="532 768 659 873">1</td> <td data-bbox="659 768 980 873">TEST_LOW</td> <td data-bbox="980 768 1487 873">If this bit is equal to 1, the X port emits a stream of K28.7 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.2.</td> </tr> <tr> <td data-bbox="532 873 659 982">0</td> <td data-bbox="659 873 980 982">TEST_HIGH</td> <td data-bbox="980 873 1487 982">If this bit is equal to 1, the X port emits a stream of D21.5 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.1.</td> </tr> </tbody> </table> <p>If more than one of the port_TEST[2:0] bits is equal to 1, the resulting behavior is undefined.</p>	BITS	NAME	DESCRIPTION	4	TEST_DEBUG	If this bit is equal to 1, the link timeouts in InfiniBand and in Gigabit Ethernet mode are approximately 1,000 times shorter than specified.	3	TEST_DISCDET	If this bit is equal to 1, the X port disables comma-detection on the incoming 8b/10b-encoded symbol stream.	2	TEST_MIX	If this bit is equal to 1, the X port emits a stream of K28.5 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.3.	1	TEST_LOW	If this bit is equal to 1, the X port emits a stream of K28.7 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.2.	0	TEST_HIGH	If this bit is equal to 1, the X port emits a stream of D21.5 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.1.
BITS	NAME	DESCRIPTION																	
4	TEST_DEBUG	If this bit is equal to 1, the link timeouts in InfiniBand and in Gigabit Ethernet mode are approximately 1,000 times shorter than specified.																	
3	TEST_DISCDET	If this bit is equal to 1, the X port disables comma-detection on the incoming 8b/10b-encoded symbol stream.																	
2	TEST_MIX	If this bit is equal to 1, the X port emits a stream of K28.5 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.3.																	
1	TEST_LOW	If this bit is equal to 1, the X port emits a stream of K28.7 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.2.																	
0	TEST_HIGH	If this bit is equal to 1, the X port emits a stream of D21.5 8b/10b-encoded symbols, as defined in IEEE 802.3 §36A.1.																	
port_STATE[5:0]	<p>Port State: The register consists of the following bits:</p> <table border="1" data-bbox="857 1129 1162 1234"> <thead> <tr> <th data-bbox="857 1129 964 1163">BITS</th> <th data-bbox="964 1129 1162 1163">NAME</th> </tr> </thead> <tbody> <tr> <td data-bbox="857 1163 964 1197">5</td> <td data-bbox="964 1163 1162 1197">FORCE_STATE</td> </tr> <tr> <td data-bbox="857 1197 964 1234">4-0</td> <td data-bbox="964 1197 1162 1234">CLSM_STATE</td> </tr> </tbody> </table> <p>Writing this register is for test-purposes only, it must not be written during regular operation (except when the X port operates in InfiniBand mode, as defined in IAS §v2.5.7.4). Writing this register assigns CLSM_STATE to state bits of CLSM (page 34). If FORCE_STATE=1, the state bits of CLSM are forced to CLSM_STATE and cannot be modified by the state machine itself. If FORCE_STATE=0, the state bits of CLSM may change according to its state diagram. Reading this register returns the current state of CLSM in its CLSM_STATE bits.</p>	BITS	NAME	5	FORCE_STATE	4-0	CLSM_STATE												
BITS	NAME																		
5	FORCE_STATE																		
4-0	CLSM_STATE																		

PORT REGISTER	DESCRIPTION		
port_RECV_CONFIG[13:0]	Port Receive Configuration: This register configures the port receive section.		
	BITS	NAME	DESCRIPTION
	13	SAN_INCLUDE	<p>This bit is applicable only to the SAN port in Lanai XM. It affects the XLED output pin, the X-port activity indicator. If this bit is equal to 0, the XLED pin reflects the state of the X port only: 0 when the port is down, 1 when the port is up, blinking when there is traffic.</p> <p>In some application of the Lanai XM chip, it is required that the XLED reflect the state of both the X port and the SAN port, and the SAN_INCLUDE should be set to 1. In this case, the XLED pin is 0 when either port is down, 1 when both ports are up, blinking when both ports are up and there is traffic on either one.</p>
12	SAN_DEBUG	<p>This bit is applicable only to the SAN port in Lanai XM.</p> <p>If this bit is equal to 1, the link timeouts (SAN_BEAT_ENABLE, SAN_ILGL_ENABLE, SAN_TIMEOUT) are approximately 8,000 times shorter than specified.</p> <p>This bit is provided only for test purposes, and must not be set during regular operation.</p>	
Continues on the next page.			

PORT REGISTER	DESCRIPTION											
port_RECV_CONFIG[13:0]	Continued from the previous page:											
	BITS	NAME	DESCRIPTION									
	11	SAN_BEAT_ENABLE	<p>This bit is applicable only to the SAN port in Lanai XM. When no BEAT control symbols are received for 30μs, the SAN port sets P1_MYRLNO_BEAT_INT in AISR (page 49).</p> <p>In addition, if SAN_BEAT_ENABLE is equal to 1, the P1_LINK_READY bit in ISR is reset (page 46), and the port shuts itself off until there are no missed BEATs (if SAN_BEAT_ENABLE is equal to 1) and no illegal control symbols (if SAN_ILGL_ENABLE is equal to 1) for 1/4 second.</p>									
	10	SAN_ILGL_ENABLE	<p>This bit is applicable only to the SAN port in Lanai XM. When an illegal control symbol is received, the SAN port sets P1_MYRILGL_INT in AISR (page 49).</p> <p>In addition, if SAN_ILGL_ENABLE is equal to 1, the P1_LINK_READY bit in ISR is reset (page 46), and the port shuts itself off until there are no missed BEATs (if SAN_BEAT_ENABLE is equal to 1) and no illegal control symbols (if SAN_ILGL_ENABLE is equal to 1) for 1/4 second.</p>									
	9-8	SAN_WINDOW	<p>These bits are applicable only to the SAN port in Lanai XM. They select the width of the character-time window for the input section of the Myrinet SAN interface.</p>									
7-6	SAN_TIMEOUT	<p>These bits are applicable only to the SAN port in Lanai XM. They select the timeout period of the Lanai SAN port. See page 49 for details.</p> <table border="1" data-bbox="1062 1482 1406 1650"> <thead> <tr> <th>Value</th> <th>Timeout</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1/16 - 1/8 second</td> </tr> <tr> <td>1</td> <td>1/4 - 1/2 second</td> </tr> <tr> <td>2</td> <td>1 - 2 seconds</td> </tr> <tr> <td>3</td> <td>4 - 8 seconds</td> </tr> </tbody> </table>	Value	Timeout	0	1/16 - 1/8 second	1	1/4 - 1/2 second	2	1 - 2 seconds	3	4 - 8 seconds
Value	Timeout											
0	1/16 - 1/8 second											
1	1/4 - 1/2 second											
2	1 - 2 seconds											
3	4 - 8 seconds											
Continues on the next page.												

PORT REGISTER	DESCRIPTION		
port_RECV_CONFIG[13:0]	Continued from the previous page:		
	BITS	NAME	DESCRIPTION
	5	ENABLE_CRC8	<p>If this bit is equal to 1, the port receiver computes the CRC-8 of the incoming packets (up to, but not including the last packet byte), and replaces the last packet byte with the XOR of its original value and the computed CRC-8.</p> <p>This bit must be equal to 1 when the port operates in Myrinet mode, unless the Lanai X is used as a repeater.</p> <p>ENABLE_CRC8 and ENABLE_CRC16 must not both be 1.</p>
	4	ENABLE_CRC16	<p>If this bit is equal to 1, the port receiver computes the CRC-16 of the incoming packets (up to, but not including the last 2 bytes of a packet), and replaces the last 2 bytes with the XOR of their original value and the computed CRC-16.</p> <p>This bit must be equal to 1 when the port operates in InfiniBand mode, unless the Lanai X is used as a repeater.</p> <p>ENABLE_CRC8 and ENABLE_CRC16 must not both be 1.</p>
	3	ENABLE_CRC32	<p>If this bit is equal to 1, the port receiver computes the CRC-32 of the incoming packets. The CRC-32 bytes are the 4 bytes preceding the CRC-8 byte or the 2 CRC-16 bytes (if enabled). The 4 CRC-32 bytes are replaced with the XOR of their original value and the computed CRC-32 (if CRC32_INFINIBAND is equal to 1, the receiver applies the appropriate CRC-32 computation, as specified in IAS §7.8.1).</p> <p>This bit must be equal to 1 when the port operates in Myrinet, InfiniBand, or Gigabit Ethernet mode, unless the Lanai X is used as a repeater.</p>
	2-1	CRC32_BIT_ORDER	<p>This field selects the bit order of the CRC-32 bytes.</p> <p>It should be set to 00 in Myrinet mode, to 11 in Gigabit Ethernet and InfiniBand modes (page 9).</p>
0	CRC32_INFINIBAND	<p>This bit must be set to 1 when operating in InfiniBand mode, and to 0 otherwise. It configures the receiver to apply the appropriate CRC-32 computation, as specified in IAS §7.8.1.</p>	

PORT REGISTER	DESCRIPTION		
port_SEND_CONFIG[9:0]	Port Send Configuration: This register configures the port send section.		
	BITS	NAME	DESCRIPTION
	9	INVALIDATE_MARK_BAD	If this bit is equal to 1, when an outgoing packet is to be invalidated (page 20, 21), it is marked bad: in Myrinet mode, this bit has no effect; in Gigabit Ethernet mode, the packet is sent with EXTEND_ERROR (IEEE 802.3 §35.2.2); in InfiniBand mode, the packet is sent with EBP (IAS §v2.5.3.1).
	8	INVALIDATE_CRC8	If this bit is equal to 1, when an outgoing packet is to be invalidated, its CRC-8 is set to 1's complement of the computed value. If this bit is equal to 1, and the ENABLE_CRC8 is not, the resulting behavior is undefined.
	7	INVALIDATE_CRC16	If this bit is equal to 1, when an outgoing packet is to be invalidated, its CRC-16 is set to 1's complement of the computed value. If this bit is equal to 1, and the ENABLE_CRC16 is not, the resulting behavior is undefined.
6	INVALIDATE_CRC32	If this bit is equal to 1, when an outgoing packet is to be invalidated, its CRC-32 is set to 1's complement of the computed value. If this bit is equal to 1, and the ENABLE_CRC32 is not, the resulting behavior is undefined.	
Continues on the next page.			

PORT REGISTER	DESCRIPTION		
port_SEND_CONFIG[9:0]	Continued from the previous page:		
	BITS	NAME	DESCRIPTION
	5	ENABLE_CRC8	If this bit is equal to 1, the port receiver computes the CRC-8 of each outgoing packet (including the CRC-32 word, if any), and appends it to the packet. This bit must be equal to 1 when the port operates in Myrinet mode, unless the Lanai X is used as a repeater. ENABLE_CRC8 and ENABLE_CRC16 must not both be 1.
	4	ENABLE_CRC16	If this bit is equal to 1, the port receiver computes the CRC-16 of each outgoing packet (including the CRC-32 word, if any), and appends it to the packet. This bit must be equal to 1 when the port operates in InfiniBand mode, unless the Lanai X is used as a repeater. ENABLE_CRC8 and ENABLE_CRC16 must not both be 1.
	3	ENABLE_CRC32	If this bit is equal to 1, the port sender computes the CRC-32 of each outgoing packet, and appends it to the packet. If CRC32_INFINIBAND is equal to 1, the receiver applies the appropriate CRC-32 computation, as specified in IAS §7.8.1. This bit must be equal to 1 when the port operates in Myrinet, InfiniBand, or Gigabit Ethernet mode, unless the Lanai X is used as a repeater.
	2-1	CRC32_BIT_ORDER	This field selects the bit order of the CRC-32 bytes. It should be set to 00 in Myrinet mode, to 11 in Gigabit Ethernet and InfiniBand modes (page 9).
0	CRC32_INFINIBAND	This bit must be set to 1 when operating in InfiniBand mode, and to 0 otherwise. It configures the sender to apply the appropriate CRC-32 computation, as specified in IAS §7.8.1.	

13.3. Shared Port Registers

PORT REGISTER	DESCRIPTION
port_SHORT_DROP[15:0]	<p>Packets-Too-Short Dropped Counter: This register is a read-only, reset-on-read, saturating counter (page 7). Regardless of the mode the port operates in, this counter is incremented when a packet shorter than 5 bytes is detected and dropped.</p>
port_8B10B_ERROR[15:0]	<p>8b/10b Error Counter: This register is applicable only to X ports. It is a read-only, reset-on-read, saturating counter (page 7). The X ports use the industry-standard 8b/10b encoding for transferring data across serial links. This register counts the low-level link errors detected by the port receiver. When the port operates in Myrinet mode, this register counts invalid 10b-symbols, non-aligned commas, and unused special symbols. When the port operates in Gigabit Ethernet mode, this register counts invalid 10b-symbols and non-aligned commas (<i>cgbad</i>, IEEE 802.3 §36.2.5.1.3). When the port operates in InfiniBand mode, this register counts errors as required by the IAS §v2.5.8.2.</p>

13.4. Myrinet-Specific Port Registers

PORT REGISTER	DESCRIPTION												
port_MYRISEND[2:0]	<p>Myrinet Send Control Symbols: This register is used when the port operates in Myrinet mode.</p>												
	<table border="1"> <thead> <tr> <th data-bbox="521 1199 659 1243">BITS</th> <th data-bbox="659 1199 979 1243">NAME</th> <th data-bbox="979 1199 1500 1243">DESCRIPTION</th> </tr> </thead> <tbody> <tr> <td data-bbox="521 1243 659 1325">2</td> <td data-bbox="659 1243 979 1325">M_AUTO_BEAT</td> <td data-bbox="979 1243 1500 1325">When this bit is equal to 1, the port emits a BEAT symbol every 10µs.</td> </tr> <tr> <td data-bbox="521 1325 659 1430">1</td> <td data-bbox="659 1325 979 1430">M_SEND_BEAT</td> <td data-bbox="979 1325 1500 1430">This is a write-only bit. Writing 1 into this bit emits one BEAT control symbol on the port.</td> </tr> <tr> <td data-bbox="521 1430 659 1633">0</td> <td data-bbox="659 1430 979 1633">M_SEND_ILGL</td> <td data-bbox="979 1430 1500 1633">This is a write-only bit. Writing 1 into this bit emits one ILGL control symbol on the port. The ILGL control symbol should be used only for testing (unless the Lanai X is used as a repeater).</td> </tr> </tbody> </table>	BITS	NAME	DESCRIPTION	2	M_AUTO_BEAT	When this bit is equal to 1, the port emits a BEAT symbol every 10µs.	1	M_SEND_BEAT	This is a write-only bit. Writing 1 into this bit emits one BEAT control symbol on the port.	0	M_SEND_ILGL	This is a write-only bit. Writing 1 into this bit emits one ILGL control symbol on the port. The ILGL control symbol should be used only for testing (unless the Lanai X is used as a repeater).
	BITS	NAME	DESCRIPTION										
	2	M_AUTO_BEAT	When this bit is equal to 1, the port emits a BEAT symbol every 10µs.										
1	M_SEND_BEAT	This is a write-only bit. Writing 1 into this bit emits one BEAT control symbol on the port.											
0	M_SEND_ILGL	This is a write-only bit. Writing 1 into this bit emits one ILGL control symbol on the port. The ILGL control symbol should be used only for testing (unless the Lanai X is used as a repeater).											
<p>If 1 is written into M_SEND_BEAT and M_SEND_ILGL bits simultaneously, one BEAT and one ILGL control symbol are emitted, in arbitrary order.</p>													
<p>The BEAT and ILGL symbols are Myrinet control symbols, and the flow control does not apply to them (ANSI Myrinet §5.1). There is no guarantee that the relative order between writing port_MYRISEND and any other packet-send operation will be preserved in Myrinet.</p> <p>If two successive writes of this register are separated by less than 1µs, the resulting behavior is undefined.</p>													

13.5. Gigabit-Ethernet-Specific Port Registers

The port registers with names with prefix port_GEX apply only to the GEX mode of operation. The port registers with names with prefix port_GE apply to both GEX and GMII operation modes.

PORT REGISTER	DESCRIPTION		
port_GE_CONFIG[2:0]	Gigabit Ethernet Configuration Register:		
	BITS	NAME	DESCRIPTION
	2	GE_REPEATER_MODE	This bit controls the repeater mode, as defined in IEEE 802.3 §36.2.5.2.5.
	1	GE_BURST_MODE	This bit controls the burst mode, as defined in IEEE 802.3 §4.2.
0	GE_HALF_DUPLEX_MODE	This bit controls the full-/half-duplex mode of all on-chip functions of the Lanai X chip, with or without Auto-Negotiation, in GEX and in GMII mode. The IEEE 802.3 §22.2.4.1.8 specification states that, when Auto-Negotiation is disabled, bit 8 of the GMII Control Register (port_GEX_CONTROL in GEX mode, corresponding register of the off-chip device in GMII mode) determines if the port is configured in full- or half-duplex mode. The Lanai X programmer must set the GE_HALF_DUPLEX_MODE bit accordingly, either upon "resolving priority" in Auto-Negotiation mode, or according to the above spec when the Auto-Negotiation is disabled.	
port_GE_FALSE_CARRIER[15:0]	Gigabit Ethernet False-Carrier Counter: This register is a read-only, reset-on-read, saturating counter (page 7). It is incremented every time a false-carrier event is detected on the port (IEEE 802.3 §22.2.2.8).		

PORT REGISTER	DESCRIPTION
port_GEX_CONTROL[15:0]	Gigabit Ethernet 1000BASE-X Control Register: This register implements GMII Register 0, as defined in IEEE 802.3 §37.2.5.1.1, with one exception (see port_GE_CONFIG register, HALF_DUPLEX_MODE bit).
port_GEX_STATUS[15:0]	Gigabit Ethernet 1000BASE-X Status Register: This register implements GMII Register 1, as defined in IEEE 802.3 §37.2.5.1.2.
port_GEX_AN_ADVERTISEMENT[15:0]	Gigabit Ethernet 1000BASE-X AN Advertisement Register: This register implements GMII Register 4, as defined in IEEE 802.3 §37.2.5.1.3.
port_GEX_AN_LP_ABILITY_BP[15:0]	Gigabit Ethernet 1000BASE-X AN Link Partner Ability Base Page Register: This register implements GMII Register 5, as defined in IEEE 802.3 §37.2.5.1.4.
port_GEX_AN_EXPANSION[15:0]	Gigabit Ethernet 1000BASE-X AN Expansion Register: This register implements GMII Register 6, as defined in IEEE 802.3 §37.2.5.1.5.
port_GEX_AN_NP_TRANSMIT[15:0]	Gigabit Ethernet 1000BASE-X AN Next Page Transmit Register: This register implements GMII Register 7, as defined in IEEE 802.3 §37.2.5.1.6.
port_GEX_AN_LP_ABILITY_NP[15:0]	Gigabit Ethernet 1000BASE-X AN Link Partner Ability Next Page Register: This register implements GMII Register 8, as defined in IEEE 802.3 §37.2.5.1.7.
port_GEX_EXTENDED_STATUS[15:0]	Gigabit Ethernet 1000BASE-X AN Extended Status Register: This register implements GMII Register 15, as defined in IEEE 802.3 §37.2.5.1.8.

13.6. InfiniBand-Specific Port Registers

PORT REGISTER	DESCRIPTION															
port_IB_CONFIG[3:0]	<p>InfiniBand Configuration: This register configures the port when it operates in InfiniBand mode. It is provided in support of InfiniBand port configuration, as specified in IAS §v2.5.5.1 and §v1.14.2.5.6, and link retraining, as specified in §v2.5.7.4.</p>															
	<table border="1"> <thead> <tr> <th data-bbox="529 495 659 531">BITS</th> <th data-bbox="665 495 977 531">NAME</th> <th data-bbox="984 495 1500 531">DESCRIPTION</th> </tr> </thead> <tbody> <tr> <td data-bbox="529 539 659 604">3</td> <td data-bbox="665 539 977 604">IB_RETRAIN</td> <td data-bbox="984 539 1500 604">This is a write-only bit. Writing 1 into this bit initiates InfiniBand-link retraining.</td> </tr> <tr> <td data-bbox="529 613 659 741">2</td> <td data-bbox="665 613 977 741">IB_DEFAULT</td> <td data-bbox="984 613 1500 741">If this bit is equal to 0, the InfiniBand LinkDownDefaultState is "Polling." If it is equal to 1, the InfiniBand LinkDownDefaultState is "Sleeping."</td> </tr> <tr> <td data-bbox="529 749 659 814">1</td> <td data-bbox="665 749 977 814">IB_RATE</td> <td data-bbox="984 749 1500 814">When this bit is equal to 1, the InfiniBand 2.5Gb/s channel rate is enabled.</td> </tr> <tr> <td data-bbox="529 823 659 888">0</td> <td data-bbox="665 823 977 888">IB_WIDTH</td> <td data-bbox="984 823 1500 888">When this bit is equal to 1, the InfiniBand 1X lane configuration is enabled.</td> </tr> </tbody> </table>	BITS	NAME	DESCRIPTION	3	IB_RETRAIN	This is a write-only bit. Writing 1 into this bit initiates InfiniBand-link retraining.	2	IB_DEFAULT	If this bit is equal to 0, the InfiniBand LinkDownDefaultState is "Polling." If it is equal to 1, the InfiniBand LinkDownDefaultState is "Sleeping."	1	IB_RATE	When this bit is equal to 1, the InfiniBand 2.5Gb/s channel rate is enabled.	0	IB_WIDTH	When this bit is equal to 1, the InfiniBand 1X lane configuration is enabled.
	BITS	NAME	DESCRIPTION													
	3	IB_RETRAIN	This is a write-only bit. Writing 1 into this bit initiates InfiniBand-link retraining.													
	2	IB_DEFAULT	If this bit is equal to 0, the InfiniBand LinkDownDefaultState is "Polling." If it is equal to 1, the InfiniBand LinkDownDefaultState is "Sleeping."													
1	IB_RATE	When this bit is equal to 1, the InfiniBand 2.5Gb/s channel rate is enabled.														
0	IB_WIDTH	When this bit is equal to 1, the InfiniBand 1X lane configuration is enabled.														
3	IB_RETRAIN	This is a write-only bit. Writing 1 into this bit initiates InfiniBand-link retraining.														
2	IB_DEFAULT	If this bit is equal to 0, the InfiniBand LinkDownDefaultState is "Polling." If it is equal to 1, the InfiniBand LinkDownDefaultState is "Sleeping."														
1	IB_RATE	When this bit is equal to 1, the InfiniBand 2.5Gb/s channel rate is enabled.														
0	IB_WIDTH	When this bit is equal to 1, the InfiniBand 1X lane configuration is enabled.														
port_IB_LDC[7:0]	<p>InfiniBand Link Downed Counter: This register is a read-only, reset-on-read, saturating counter (page 7). When the port operates in InfiniBand mode, this register counts errors as required by the IAS §v2.5.5.3.</p>															
port_IB_LERC[7:0]	<p>InfiniBand Link Error Recovery Counter: This register is a read-only, reset-on-read, saturating counter (page 7). When the port operates in InfiniBand mode, this register counts errors as required by the IAS §v2.5.5.3.</p>															
port_IB_SEC[15:0]	<p>InfiniBand Symbol Error Counter: See port_8B10B_ERROR (page 41).</p>															
port_IB_LEC[15:0]	<p>InfiniBand Local Error Counter: This register is a read-only, reset-on-read, saturating counter (page 7). When the port operates in InfiniBand mode, this register counts the number of packets dropped due to local-link errors. This includes packets forwarded to the packet interface and marked bad (page 13), which should be silently discarded. This counter is provided in support of implementation of the InfiniBand PortRcvErrors counter, as defined in IAS §v1.16.1.3.5.</p>															
port_IB_REC[15:0]	<p>InfiniBand Remote Error Counter: This register is a read-only, reset-on-read, saturating counter (page 7). When the port operates in InfiniBand mode, this register counts the number of packets dropped due to remote-link errors (marked with EBP). These packets are forwarded to the packet interface and marked bad (page 13), and should be silently discarded. This counter is provided in support of implementation of the InfiniBand PortRcvRemotePhysicalErrors counter, as defined in IAS §v1.16.1.3.5.</p>															

14. INTERFACE-STATUS REGISTERS

REGISTER	DESCRIPTION
ISR[31:0]	Interface-Status Register: Essential chip-status information. The status bits in ISR are considered essential for efficient chip operation, and are used directly by the event-driven code dispatch (page 50).
AISR[31:0]	Alternate Interface-Status Register: Additional chip-status information. The status bits in AISR are not considered essential for efficient chip operation, and cannot be used directly by the event-driven code dispatch (page 50). Instead, AISR_ON and AISR_OFF registers are used as masks to compute the P0_EVENT and P1_EVENT bits of ISR: If an AISR bit is equal to 1, and the corresponding AISR_ON bit is equal to 1, or if an AISR bit is equal to 0, and the corresponding AISR_OFF bit is equal to 1, one of the two _EVENT bits in ISR is set. P0_EVENT is controlled by the Port-0-related AISR bits, and P1_EVENT by the Port-1-related AISR bits.
AISR_ON[31:0]	AISR Set Mask Register: Selects the AISR bits which, when equal to 1, set P0_EVENT or P1_EVENT bit of ISR.
AISR_OFF[31:0]	AISR Reset Mask Register: Selects the AISR bits which, when equal to 0, set P0_EVENT or P1_EVENT bit of ISR.

The bits of ISR and AISR with the _INT (interrupt) postfix are set by the hardware when their corresponding events occur. These bits are reset by writing a 1 into them.

All other bits are status flags, controlled by the hardware, and they cannot be modified by the software.

BIT	ISR BIT NAME	DESCRIPTION
31-30		Reserved.
29-28	REQ_ACK[1:0]	These bits are included only the Lanai chips with the PCI interface (XP, 2XP). A REQ_ACK bit is equal to 0 upon reset, and it is inverted every time a 1 is written into it. The intended use of these bits is for simple request/acknowledge protocols. For example, the Lanai can set a bit to post a request to the host; when the host detects the request and performs the required action, it clears the bit to notify the Lanai of completion. The REQ_ACK[0] bit is used as the PCI-interrupt request; if this bit and the PCI_INT_ENABLE bit (page 26) are set, a PCI interrupt is generated.
27-24	DMA_DONE[3:0]	A DMA_DONE[<i>i</i>] bit is equal to 1 if its corresponding PCI-DMA count register, DMA _{<i>i</i>} _COUNT (page 31), is non-zero.
23	DMA_ERROR_INT	This bit is set when the PCI-DMA engine runs out of LBUS memory bandwidth. If the PCI-DMA engine has been assigned sufficient memory bandwidth (page 52), this bit is never set. It is cleared by writing a 1 into it.
22	WAKE_INT	This bit is set when the WAKE input put is asserted. It is cleared by writing a 1 into it.
21	MEMORY_INT	This bit is set when a memory-protection violation is detected (page 8). It is cleared by writing a 1 into it.
20	PARITY_INT	This bit is set when a parity error is detected on the LBUS. It is cleared by writing a 1 into it. After a power-on reset, the entire LBUS memory should be initialized to avoid false parity-error warnings.
19-17	TIME_INT[2:0]	A TIME_INT[<i>i</i>] bit is set by the corresponding timer, IT _{<i>i</i>} (page 8), when it makes a transition from 0x00000000 to 0xFFFFFFFF. Each bit can be cleared directly — by writing a 1 into it, or indirectly — when the corresponding timer is written. Indirect clearing may occur with a maximum delay of 1 CPU clock cycle.
16	COPY_BUSY	This bit is equal to 1 if the memory-copy engine (page 9) is busy.

BIT	ISR BIT NAME	DESCRIPTION
15	P1_EVENT	Port-1 version of P0_EVENT.
14	P1_A_PACKET_RCVD	Port-1 version of P0_PACKET_RCVD.
13	P1_NO_BACKUP	Port-1 version of P0_NO_BACKUP.
12	P1_PACKET_RCVD	Port-1 version of P0_PACKET_RCVD.
11	P1_PACKET_HEAD	Port-1 version of P0_PACKET_HEAD.
10	P1_PACKET_SENT	Port-1 version of P0_PACKET_SENT.
9	P1_SEND_READY	Port-1 version of P0_SEND_READY.
8	P1_LINK_READY	<p>If the port is an X port, the bit is a Port-1 version of P0_LINK_READY.</p> <p>If the port is a SAN port, the status of this bit depends on the state of the P1_RECV_CONFIG register (page 36). If the SAN_BEAT_ENABLE and/or the SAN_ILGL_ENABLE bit are equal to 1, and the corresponding error condition has been detected, the SAN interface clears the P1_LINK_READY bit and drops all incoming and outgoing packets. It resorts back to the normal condition when there are no missed BEATs (if BEAT_ENABLE is equal to 1) and no illegal control symbols have been received (if ILGL_ENABLE is equal to 1) for at least 1/4 second.</p> <p>See Note on page 49.</p>
7	P0_EVENT	<p>This bit is computed from Port-0-related AISR bits, based on masks specified in AISR_ON and AISR_OFF.</p> <p>The bit is set if an AISR bit is equal to 1, and the corresponding AISR_ON bit is equal to 1, or if an AISR bit is equal to 0, and the corresponding AISR_OFF bit is equal to 1.</p>
6	P0_A_PACKET_RCVD	This bit is equal to 1 when P0_A_RECV_COUNT is greater than 0, indicating that there is at least one packet in the Received-Packet List of Port 0, Alternate Channel (page 10).
5	P0_NO_BACKUP	This bit is set if Port 0, Regular Channel has no backup receive buffer (page 10).
4	P0_PACKET_RCVD	This bit is equal to 1 when P0_RECV_COUNT is greater than 0, indicating that there is at least one packet in the Received-Packet List of Port 0, Regular Channel (page 10).
3	P0_PACKET_HEAD	This bit is set when Port 0, Regular Channel receiver writes the header of an incoming packet into the receive buffer (as specified by P0_HEADER_LENGTH, page 17); the bit is reset when the entire packet (and its corresponding descriptor) has been written (page 10).
2	P0_PACKET_SENT	This bit is equal to 1 when P0_SEND_COUNT (page 22) is greater than 0, indicating that at least one packet has been sent (page 19).
1	P0_SEND_READY	This bit is equal to 1 when $P0_SEND_FREE_COUNT \geq port_SEND_FREE_LIMIT + 2$ (page 19).
0	P0_LINK_READY	<p>This bit is equal to 1 when Port 0 indicates that the link has established low-level synchronization with the device on the other end of the link, and is ready to exchange data packets. It does not take into the account the state of the local or of the remote receive data buffer.</p> <p>The details depend on the mode the port operates in (page 34): Myrinet (IN_SYNC_GO, IN_SYNC_STOP), InfiniBand (LinkUp), Gigabit Ethernet 1000BASE-X (LINK_OK, AN_DISABLE_LINK_OK). In GMII mode, the P0_LINK_READY is always set, and the real link status must be obtained from the device used for the application at hand (page 8).</p>

The following bits of AISR control the P0_EVENT bit of ISR:

BIT	AISR BIT NAME	DESCRIPTION
12	P0_GEX_LATE_COLL_INT	This bit is applicable only in half-duplex Gigabit Ethernet mode. See P0_GEX_NO_COLL_INT for details.
11	P0_GEX_REG_COLL_INT	This bit is applicable only in half-duplex Gigabit Ethernet mode. See P0_GEX_NO_COLL_INT for details.
10	P0_GEX_NO_COLL_INT	This bit is applicable only in half-duplex Gigabit Ethernet mode. In this mode, when a packet is emitted into the network, there are three possible outcomes: no collision, regular collision, or late collision. In half-duplex mode, after writing the send-packet-segment descriptor for a single packet into the send FIFO (page 19), the CPU must wait until one of the three collision bits is set, and act according to CSMA/CD specification in IEEE 802.3 §4. These bits are cleared by writing a 1 into them.
9	P0_GEX_RESOLVE_INT	This bit is applicable only in Gigabit Ethernet 1000BASE-X mode. This bit is set during Auto-Negotiation, when the link partners have exchanged all required information, and the CPU is asked to "resolve priority", <i>i.e.</i> , find the largest common denominator of their capabilities, as specified in IEEE 802.3 §37.2.4.2, and to configure the port accordingly. This bit is cleared by writing a 1 into it.
8	P0_MYRI_STOP	This bit is applicable only in Myrinet mode. If the available buffer space in the current receive buffer (page 10) drops below size specified by STOP_SIZE, and there is no backup buffer, a STOP control symbol is generated (page 10) and the P0_MYRI_STOP bit is set. When the next receive buffer is provided, a GO control symbol is generated, and the port_MYRI_STOP bit is cleared.
7	P0_MYRI_ILGL_INT	This bit is applicable only in Myrinet mode, and it is set when the ILGL control symbol is received. It is cleared by writing a 1 into it.
6	P0_MYRI_BEAT_INT	This bit is applicable only in Myrinet mode, and it is set when the BEAT control symbol is received. It is cleared by writing a 1 into it.
5	P0_MYRI_NO_BEAT_INT	This bit is applicable only in Myrinet mode, and it is set when the BEAT control symbol is not received for 30 microseconds. It is cleared by writing a 1 into it.
4	P0_A_NO_BACKUP	This bit is equal to 1 if Port 0, Alternate Channel has no backup receive buffer (page 10).
3	P0_A_NO_BUFFER	This bit is equal to 1 if Port 0, Alternate Channel has no receive buffers (page 10).
2	P0_NO_BUFFER	This bit is equal to 1 if Port 0, Regular Channel has no receive buffers (page 10).
1	P0_SEND_STOPPED	Under certain conditions, defined at P0_SEND_CONTROL on page 23, the sender is instructed to complete the current packet (if any) and stop sending. When the current packet is completed (if any), and the sender stops, the P0_SEND_STOPPED is set. It is reset when the sending resumes (page 23).
0	P0_PAUSE_RCVD	This bit is equal to 1 if P0_PAUSE_COUNT is not equal to 0, indicating that a valid Gigabit Ethernet MAC Control PAUSE packet has been received (page 23).

The following bits of AISR control the P1_EVENT bit of ISR (not in Lanai XM):

BIT	AISR BIT NAME	DESCRIPTION
31-26		Reserved.
25	P1_GEX_LATE_COLL_INT	Port-1 version of P0_GEX_LATE_COLL_INT.
24	P1_GEX_REG_COLL_INT	Port-1 version of P0_GEX_COLL_INT.
23	P1_GEX_NO_COLL_INT	Port-1 version of P0_GEX_NO_COLL_INT.
22	P1_GEX_RESOLVE_INT	Port-1 version of P0_GEX_RESOLVE_INT.
21	P1_MYRI_STOP	Port-1 version of P0_MYRI_STOP.
20	P1_MYRI_ILGL_INT	Port-1 version of P0_MYRI_ILGL_INT.
19	P1_MYRI_BEAT_INT	Port-1 version of P0_MYRI_BEAT_INT.
18	P1_MYRI_NO_BEAT_INT	Port-1 version of P0_NO_BEAT_INT.
17	P1_A_NO_BACKUP	Port-1 version of P0_A_NO_BACKUP.
16	P1_A_NO_BUFFER	Port-1 version of P0_A_NO_BUFFER.
15	P1_NO_BUFFER	Port-1 version of P0_NO_BUFFER.
14	P1_SEND_STOPPED	Port-1 version of P0_SEND_STOPPED.
13	P1_PAUSE_RCVD	Port-1 version of P0_PAUSE_RCVD.

The following bits of AISR control the P1_EVENT bit of ISR (Lanai XM only):

BIT	AISR BIT NAME	DESCRIPTION
31-27		Reserved.
26	SAN_DATA_SENT_INT	This bit is set when a data symbol is emitted by the Myrinet SAN interface. It is cleared by writing a 1 into it.
25	SAN_DATA_RCVD_INT	This bit is set when a data symbol is received by the Myrinet SAN interface. It is cleared by writing a 1 into it.
24	SAN_TX_TOO_LONG_INT	This bit is set if an outgoing packet takes longer than the period specified by the SAN_TIMEOUT bits (page 37). This can happen if a packet is too long, or if it is blocked for too long. This bit is cleared by writing a 1 into it. See Note below.
23	SAN_RX_TOO_LONG_INT	This bit is set if an incoming packet takes longer than the period specified by the SAN_TIMEOUT bits (page 37). This can happen if a packet is too long, or if it is blocked for too long. This bit is cleared by writing a 1 into it. See Note below.
22	SAN_TX_BLOCKED_INT	This bit is set if an outgoing packet is blocked by the SAN-link flow control longer than the period specified by the SAN_TIMEOUT bits (page 37). It is cleared by writing a 1 into it. See Note below.
21	SAN_RX_BLOCKED_INT	This bit is set by the Myrinet-SAN interface whenever the Lanai chip fails to consume an incoming packet from the Myrinet network for the duration of the period specified by the SAN_TIMEOUT bits (page 37). It is cleared by writing a 1 into it. See Note below.
20	P1_MYRI_ILGL_INT	Port-1 version of P0_MYRI_ILGL_INT.
19	P1_MYRI_BEAT_INT	Port-1 version of P0_MYRI_BEAT_INT.
18	P1_MYRI_NO_BEAT_INT	Port-1 version of P0_MYRI_NO_BEAT_INT.
17	P1_A_NO_BACKUP	Port-1 version of P0_A_NO_BACKUP.
16	P1_A_NO_BUFFER	Port-1 version of P0_A_NO_BUFFER.
15	P1_NO_BUFFER	Port-1 version of P0_NO_BUFFER.
14	P1_SEND_STOPPED	Port-1 version of P0_SEND_STOPPED.
13	P1_PAUSE_RCVD	Port-1 version of P0_PAUSE_RCVD.

NOTE:

The Myrinet SAN was designed to operate in presence of network faults, including powering up and down of network components, plugging and unplugging of cables, *etc.* The SAN error-condition bits described above (P1_MYRI_ILGL_INT, P1_MYRI_NO_BEAT_INT, SAN_TX_BLOCKED_INT, SAN_TX_TOO_LONG_INT, SAN_RX_TOO_LONG_INT, SAN_RX_BLOCKED_INT) are merely an indication that a particular problem has occurred since its corresponding bit was reset; they are not an accurate representation of the current state of the SAN port.

The exact behavior of the SAN port is non-deterministic, and depends on timing of network faults, packet lengths, operating frequency, *etc.*

The only guarantee is that, if both ports of a SAN link are powered up, out of reset, functional, and not blocking the packets arriving from the Myrinet SAN, the link will start operating correctly within 1/2 second.

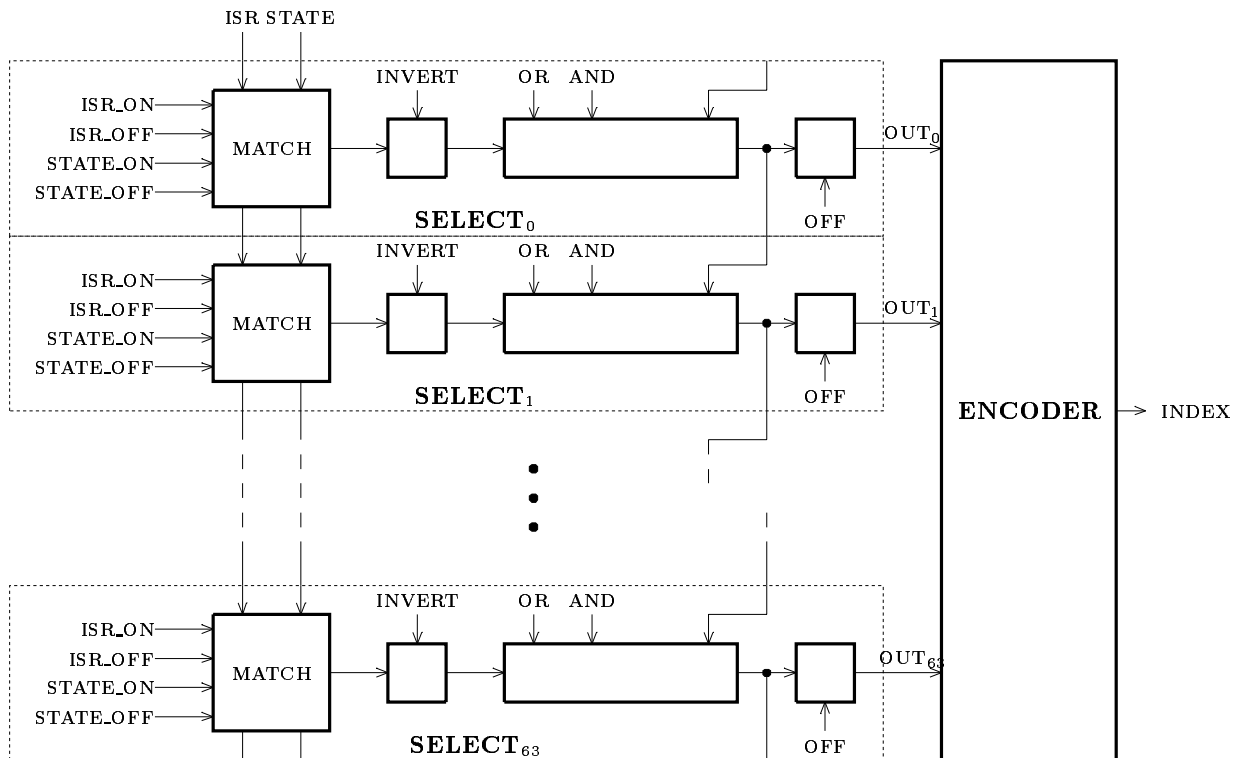
When the SAN port detects a problem in the outgoing packet stream (SAN_TX_BLOCKED_INT, SAN_TX_TOO_LONG_INT), it drops all outgoing packets until the problem goes away.

When the SAN port detects a problem in the incoming packet stream (P1_MYRI_ILGL_INT, P1_MYRI_NO_BEAT_INT, SAN_RX_TOO_LONG_INT, SAN_RX_BLOCKED_INT), it drops all incoming packets until the problem goes away. When the problem does go away, the port injects two tail flits into the incoming-packet channel, to complete any packets that may have been truncated. Consumption of these two, bogus packets is the responsibility of the programmer.

15. EVENT-DRIVEN CODE DISPATCH

The Lanai X provides hardware support for event-driven code dispatch. This mechanism does not have to be used, but it offers a fast solution that uses a small memory-resident dispatch table.

The mechanism uses the programmable-logic structure illustrated below. The programmer can specify up to 64 general product terms of all non-reserved ISR bits and 32 additional, software state bits. Once the programmable logic is configured, the intended use of this mechanism is to write the word containing the 32 software state bits into the special register DISPATCH_STATE, and then read the index of the highest-priority event handler out of the special register DISPATCH_INDEX.



The programmable-logic structure consists of 64 configurable select blocks and the encoder.

The output of the encoder is the smallest index (0-63) of all the select blocks whose OUT is equal to 1. If there are no such select blocks, the encoder output is equal to 64.

The configuration of a select block consists of the following registers: ISR_ON, ISR_OFF, STATE_ON, STATE_OFF, INVERT, OR, and AND.

ISR_ON register specifies which ISR bits must be equal to 1, ISR_OFF specifies which ISR bits must be equal to 0, STATE_ON specifies which STATE bits must be equal to 1, and STATE_OFF specifies which STATE bits must be equal to 0 for the output of the match block to be 1. Note that if all the bits of ISR_ON, ISR_OFF, STATE_ON, and STATE_OFF are equal to 0, the output of the match block is 1 (all the bits that must be of certain value, namely none, are).

This result can be optionally inverted (if INVERT is set), OR-ed with the partial output of the previous select block (if OR is set), or AND-ed with the partial output of the previous select block (if OR is not set, and AND is set). The OFF signals are defined as follows: $OFF_{i-1} = OR_i \mid AND_i$. In other words, if the partial output of a select block, $i - 1$, is used as an input of the following select block, i , its output is disabled (OUT_{i-1} is equal to 0).

AND₀, OR₀, and OFF₆₃ are hard-wired to 0.

The following special registers control the event-dispatch programmable logic:

REGISTER	DESCRIPTION										
DISPATCH_STATE[31:0]	Dispatch State: Writing this write-only register initiates the index computation.										
DISPATCH_INDEX[8:0]	Dispatch Index: This read-only register must not be read during the 2 clock cycles immediately following the write to DISPATCH_STATE. Reading it returns the value 0-256, equal to 4 times the index of the highest-priority dispatch event (the smallest index of all the select blocks whose OUT is equal to 1). If the DISPATCH_CONFIG register is written between writing DISPATCH_STATE and reading DISPATCH_INDEX, the resulting behavior is undefined.										
DISPATCH_CONFIG[31:0]	<p>Dispatch Configuration:</p> <table border="1"> <thead> <tr> <th>BITS</th> <th>NAME</th> </tr> </thead> <tbody> <tr> <td>8</td> <td>DISPATCH_AND</td> </tr> <tr> <td>7</td> <td>DISPATCH_OR</td> </tr> <tr> <td>6</td> <td>DISPATCH_INVERT</td> </tr> <tr> <td>5-0</td> <td>DISPATCH_SELECT</td> </tr> </tbody> </table> <p>When this write-only register is written, the select block DISPATCH_SELECT is configured with the values specified in DISPATCH_INVERT, DISPATCH_OR, DISPATCH_AND, DISPATCH_ISR_ON, DISPATCH_ISR_OFF, DISPATCH_STATE_ON, and DISPATCH_STATE_OFF.</p>	BITS	NAME	8	DISPATCH_AND	7	DISPATCH_OR	6	DISPATCH_INVERT	5-0	DISPATCH_SELECT
BITS	NAME										
8	DISPATCH_AND										
7	DISPATCH_OR										
6	DISPATCH_INVERT										
5-0	DISPATCH_SELECT										
DISPATCH_ISR_ON[31:0]	Dispatch Product Term: This write-only register specifies the ISR bits in a product term that must be equal to 1 (see DISPATCH_CONFIG). This register must not be written during the clock cycle immediately following the write to DISPATCH_CONFIG.										
DISPATCH_ISR_OFF[31:0]	Dispatch Product Term: This write-only register specifies the ISR bits in a product term that must be equal to 0 (see DISPATCH_CONFIG). This register must not be written during the clock cycle immediately following the write to DISPATCH_CONFIG.										
DISPATCH_STATE_ON[31:0]	Dispatch Product Term: This write-only register specifies the STATE bits in a product term that must be equal to 1 (see DISPATCH_CONFIG). This register must not be written during the clock cycle immediately following the write to DISPATCH_CONFIG.										
DISPATCH_STATE_OFF[31:0]	Dispatch Product Term: This write-only register specifies the STATE bits in a product term that must be equal to 0 (see DISPATCH_CONFIG). This register must not be written during the clock cycle immediately following the write to DISPATCH_CONFIG.										

The following special registers are used to configure the memory arbitration:

REGISTER	DESCRIPTION
ARBITER_SLOT[4:0]	Memory-Arbitration Slot: This register specifies the arbiter slot to be accessed through the ARBITER_CODE register.
ARBITER_CODE[3:0]	Memory-Arbitration Code: The value written into this register is written into the arbiter slot specified by ARBITER_SLOT. When this register is read, it returns the code from the arbiter slot specified by ARBITER_SLOT. If this register is read during the clock cycle immediately following the write to ARBITER_SLOT, the returned value is undefined.

17. JTAG INTERFACE, EEPROM INTERFACE

All three Lanai X variants (XM, XP, and 2XP) have a JTAG interface that can be used to access Lanai LBUS memory and special registers, to control LBUS memory timing and internal chip-reset signals.

There are two internal reset signals that control the operation of a Lanai chip: CPU_RESET, which resets the CPU, and IO_RESET, which resets most of the remaining non-PCI Lanai circuitry. These two signals are controlled through the JTAG interface, and, on XP/2XP chips, can also be controlled through the PCI interface:

CPU_RESET = PCI_CPU_RESET or JTAG_CPU_RESET

IO_RESET = PCI_IO_RESET or JTAG_IO_RESET

Upon hardware reset (controlled by the \overline{RST} pin), PCI_RESETs (if any) are cleared, enabling the JTAG interface to control the chip. Once a Lanai XP/2XP-based interface has been properly configured, the JTAG_RESET signals are cleared, and the chip can be controlled through the PCI interface.

Any system with a Lanai XM chip has to have some additional circuitry that can perform the minimal required initialization through the JTAG interface: configure the LBUS memory timing, load the initial Lanai code, then clear JTAG_IO_RESET, and clear JTAG_CPU_RESET.

The Lanai XP/2XP chips include an EEPROM interface that can control the JTAG, so these chips can perform the required initialization themselves (page 27).

The details of the JTAG interface are beyond the scope of this document; contact Myricom for details.

A. SUMMARY OF SPECIAL REGISTERS

All special registers are mapped into the LBUS memory space, and they can be accessed by the CPU, through the PCI-X interface, and through the JTAG interface.

To access a special register from the Lanai CPU one should use the address of 0xFFFFFC00 plus the offset of that special register. See page 24 for details on accessing special registers through PCI-X.

SPECIAL REGISTERS, Interface Status					
Register	Read	Write	Description	Offset	Page
ISR[31:0]	+	+	Interface Status	0x1F8	45
AISR[31:0]	+	+	Alternate Interface Status	0x1F0	45
AISR_ON[31:0]	+	+	AISR Set Mask	0x1E8	45
AISR_OFF[31:0]	+	+	AISR Reset Mask	0x1E0	45
SPECIAL REGISTERS, Event Dispatch					
Register	Read	Write	Description	Offset	Page
DISPATCH_STATE[31:0]		+	Initiate Dispatch	0x170	51
DISPATCH_INDEX[8:0]	+		Dispatch Result	0x170	51
DISPATCH_CONFIG[31:0]		+	Dispatch Configuration	0x198	51
DISPATCH_ISR_ON[31:0]		+	Dispatch Product Term, ISR ON bits	0x178	51
DISPATCH_ISR_OFF[31:0]		+	Dispatch Product Term, ISR OFF bits	0x180	51
DISPATCH_STATE_ON[31:0]		+	Dispatch Product Term, STATE ON bits	0x188	51
DISPATCH_STATE_OFF[31:0]		+	Dispatch Product Term, STATE OFF bits	0x190	51
SPECIAL REGISTERS, PCI DMA					
Register	Read	Write	Description	Offset	Page
DMA_CONFIG[31:0]	+	+	PCI DMA Configuration	0x280	32
DMA0_POINTER[31:0]		+	PCI DMA Pointer	0x2B0	31
DMA1_POINTER[31:0]		+	PCI DMA Pointer	0x2B8	31
DMA2_POINTER[31:0]		+	PCI DMA Pointer	0x2C0	31
DMA3_POINTER[31:0]		+	PCI DMA Pointer	0x2C8	31
DMA0_COUNT[7:0]	+	D*	PCI DMA-Done Counter	0x290	31
DMA1_COUNT[7:0]	+	D*	PCI DMA-Done Counter	0x299	31
DMA2_COUNT[7:0]	+	D*	PCI DMA-Done Counter	0x2A2	31
DMA3_COUNT[7:0]	+	D*	PCI DMA-Done Counter	0x2AB	31
PCI_CLOCK[15:0]	+	+	PCI Clock Counter	0x28A	31
SPECIAL REGISTERS, Copy/CRC32					
Register	Read	Write	Description	Offset	Page
COPY[31:0]		+	Memory Copy	0x0F0	9
COPY_ABORT[0]		+	Abort Memory Copy	0x0FA	9
CRC32_CONFIG[2:1]	+	+	CRC32 Configuration	0x102	9
CRC32[31:0]	+	+	CRC32 State	0x108	9
CRC32_BYTE[7:0]		+	CRC32 Compute	0x112	9
CRC32_HALF[15:0]		+	CRC32 Compute	0x118	9
CRC32_WORD[31:0]		+	CRC32 Compute	0x120	9
SPECIAL REGISTERS, Counters, Timers					
Register	Read	Write	Description	Offset	Page
CPUC[31:0]	+	+	CPU-Clock Counter	0x128	8
RTC[31:0]	+	+	Real-Time Clock	0x130	8
IT0[31:0]	+	+	Interval Timer	0x138	8
IT1[31:0]	+	+	Interval Timer	0x140	8
IT2[31:0]	+	+	Interval Timer	0x148	8

D* — these registers are decremented when written

SPECIAL REGISTERS, Programmable Input/Output Pins					
Register	Read	Write	Description	Offset	Page
LED[3:0]	+	+	LED Output Pins	0x150	8
MDI[2:0]	+	+	GMI Management Data Interface	0x158	8
SPECIAL REGISTERS, Memory Arbitration, Memory Protection					
Register	Read	Write	Description	Offset	Page
ARBITER_SLOT[4:0]	+	+	Memory-Arbitration Slot	0x1A0	53
ARBITER_CODE[3:0]	+	+	Memory-Arbitration Code	0x1A8	53
MP_LOWER[30:3]	+	+	Memory Protection, Lower Bound	0x1B0	8
MP_UPPER[31:3]	+	+	Memory Protection, Upper Bound	0x1B8	8
SPECIAL REGISTERS, Hardware Configuration					
Register	Read	Write	Description	Offset	Page
PLL[31:0]	+	+	PLL Configuration	0x168	8
SYNC[31:0]	+	+	Synchronizer Configuration	0x160	8

SPECIAL REGISTERS, Port 0

Send Special Registers					
Register	Read	Write	Description	Offset	Page
P0_SEND[31:0]		+	P0 Send FIFO	0x048	22
P0_SEND_FREE_COUNT[4:0]	+		P0 Send-FIFO-Free Counter	0x04B	22
P0_SEND_FREE_LIMIT[3:0]	+	+	P0 Send-FIFO-Free Limit	0x053	22
P0_SEND_COUNT[7:0]	+	D*	P0 Send Packet Counter	0x05B	22
P0_PAUSE_COUNT[7:0]	+	D*	P0 Pause Counter	0x063	23
P0_SEND_CONTROL[1:0]	+	+	P0 Send Control	0x06B	23
Receive Channel-Independent Special Registers					
Register	Read	Write	Description	Offset	Page
P0_MEMORY_DROP[15:0]	Z*		P0 No-Cycles Packet-Drop Counter	0x040	15
Regular Receive Channel Special Registers					
Register	Read	Write	Description	Offset	Page
P0_RB[31:0]	+	+	P0 Receive Buffer	0x008	15
P0_RPL_CONFIG[31:0]	+	+	P0 Receive-Channel Configuration	0x000	16
P0_RPL_INDEX[15:0]	+		P0 Received-Packet-List Offset	0x010	15
P0_RECV_COUNT[15:0]	+	D*	P0 Received-Packet-List Counter	0x012	15
P0_RPL_SNAPSHOT[31:0]	+		P0 Received-Packet-List Snapshot	0x010	15
P0_BUFFER_DROP[15:0]	Z*		P0 No-Buffers Packet-Drop Counter	0x032	15
Alternate Receive Channel Special Registers					
Register	Read	Write	Description	Offset	Page
P0_A_RB[31:0]	+	+	P0_A Receive Buffer	0x020	15
P0_A_RPL_CONFIG[31:0]	+	+	P0_A Receive-Channel Configuration	0x018	16
P0_A_RPL_INDEX[15:0]	+		P0_A Received-Packet-List Offset	0x028	15
P0_A_RECV_COUNT[15:0]	+	D*	P0_A Received-Packet-List Counter	0x02A	15
P0_A_RPL_SNAPSHOT[31:0]	+		P0_A Received-Packet-List Snapshot	0x028	15
P0_A_BUFFER_DROP[15:0]	Z*		P0_A No-Buffers Packet-Drop Counter	0x03A	15

Z* — these registers are reset to zero when read; because of read side-effects, they cannot be accessed through the PCI

D* — these registers are decremented when written

SPECIAL REGISTERS, Port 1

Send Special Registers					
Register	Read	Write	Description	Offset	Page
P1_SEND[31:0]		+	P1 Send FIFO	0x0C8	22
P1_SEND_FREE_COUNT[4:0]	+		P1 Send-FIFO-Free Counter	0x0CB	22
P1_SEND_FREE_LIMIT[3:0]	+	+	P1 Send-FIFO-Free Limit	0x0D3	22
P1_SEND_COUNT[7:0]	+	D*	P1 Send Packet Counter	0x0DB	22
P1_PAUSE_COUNT[7:0]	+	D*	P1 Pause Counter	0x0E3	23
P1_SEND_CONTROL[1:0]	+	+	P1 Send Control	0x0EB	23
Receive Channel-Independent Special Registers					
Register	Read	Write	Description	Offset	Page
P1_MEMORY_DROP[15:0]	Z*		P1 No-Cycles Packet-Drop Counter	0x0C0	15
Regular Receive Channel Special Registers					
Register	Read	Write	Description	Offset	Page
P1_RB[31:0]	+	+	P1 Receive Buffer	0x088	15
P1_RPL_CONFIG[31:0]	+	+	P1 Receive-Channel Configuration	0x080	16
P1_RPL_INDEX[15:0]	+		P1 Received-Packet-List Offset	0x090	15
P1_RECV_COUNT[15:0]	+	D*	P1 Received-Packet-List Counter	0x092	15
P1_RPL_SNAPSHOT[31:0]	+		P1 Received-Packet-List Snapshot	0x090	15
P1_BUFFER_DROP[15:0]	Z*		P1 No-Buffers Packet-Drop Counter	0x0B2	15
Alternate Receive Channel Special Registers					
Register	Read	Write	Description	Offset	Page
P1_A_RB[31:0]	+	+	P1_A Receive Buffer	0x0A0	15
P1_A_RPL_CONFIG[31:0]	+	+	P1_A Receive-Channel Configuration	0x098	16
P1_A_RPL_INDEX[15:0]	+		P1_A Received-Packet-List Offset	0x0A8	15
P1_A_RECV_COUNT[15:0]	+	D*	P1_A Received-Packet-List Counter	0x0AA	15
P1_A_RPL_SNAPSHOT[31:0]	+		P1_A Received-Packet-List Snapshot	0x0A8	15
P1_A_BUFFER_DROP[15:0]	Z*		P1_A No-Buffers Packet-Drop Counter	0x0BA	15

Z* — these registers are reset to zero when read; because of read side-effects, they cannot be accessed through the PCI

D* — these registers are decremented when written

B. SUMMARY OF COMMUNICATION-PORT REGISTERS

The communication ports operate in clock domains that are independent from the Lanai-core clock. As a result, the control/status registers associated with the port circuitry cannot be accessed using the standard, memory-mapped, special-register-access mechanism (page 7).

Instead, the port registers are accessed through the two port-access special registers described on page 33, using the Offset specified below as the port-register address.

SPECIAL REGISTERS, Port-Register Access					
Register	Read	Write	Description	Offset	Page
PORT_ADDR[15:0]	+	+	Port-Register Address	0x070	33
PORT_DATA[15:0]	+	+	Port-Register Data	0x078	33

PORT REGISTERS, Port 0

Shared Registers					
Port Register	Read	Write	Description	Offset	Page
P0_MODE[3:0]	+	+	P0 Mode	0x10	34
P0_TEST[4:0]	+	+	P0 Test	0x20	35
P0_STATE[5:0]	+	+	P0 State	0x11	35
P0_RECV_CONFIG[13:0]	+	+	P0 Receive Configuration	0x09	36
P0_SEND_CONFIG[9:0]	+	+	P0 Send Configuration	0x23	39
P0_SHORT_DROP[15:0]	Z*		P0 Too-Short Packet-Drop Counter	0x0A	41
P0_8B10B_ERROR[15:0]	Z*		P0 8b/10b Error Counter	0x12	41
Myrinet-Specific Registers					
Port Register	Read	Write	Description	Offset	Page
P0_MYRI_SEND[2:0]	+	+	P0 Myrinet Send Control Symbols	0x22	41
Gigabit-Ethernet-Specific Registers					
Port Register	Read	Write	Description	Offset	Page
P0_GEX_CONTROL[15:0]	+	+	P0 GEX Control	0x00	43
P0_GEX_STATUS[15:0]	+		P0 GEX Status	0x01	43
P0_GEX_AN_ADVERTISEMENT[15:0]	+	+	P0 GEX AN Advertisement	0x02	43
P0_GEX_AN_LP_ABILITY_BP[15:0]	+		P0 GEX AN Link Partner Ability Base Page	0x03	43
P0_GEX_AN_EXPANSION[15:0]	+		P0 GEX AN Expansion	0x04	43
P0_GEX_AN_NP_TRANSMIT[15:0]	+	+	P0 GEX AN Next Page Transmit	0x05	43
P0_GEX_AN_LP_ABILITY_NP[15:0]	+		P0 GEX AN Link Partner Ability Next Page	0x06	43
P0_GEX_EXTENDED_STATUS[15:0]	+		P0 GEX Extended Status	0x07	43
P0_GE_CONFIG[2:0]	+	+	P0 GE Configuration	0x21	42
P0_GE_FALSE_CARRIER[15:0]	Z*		P0 GE False-Carrier Counter	0x08	42
InfiniBand-Specific Registers					
Port Register	Read	Write	Description	Offset	Page
P0_IB_CONFIG[3:0]	+	+	P0 IB Configuration	0x17	44
P0_IB_LDC[7:0]	Z*		P0 IB Link Downed Counter	0x14	44
P0_IB_LERC[7:0]	Z*		P0 IB Link Error Recovery Counter	0x13	44
P0_IB_SEC[15:0]	Z*		P0 IB Symbol Error Counter	0x12	44
P0_IB_LEC[15:0]	Z*		P0 IB Local Error Counter	0x15	44
P0_IB_REC[15:0]	Z*		P0 IB Remote Error Counter	0x16	44

Z* — these registers are reset to zero when read

PORT REGISTERS, Port 1

Shared Registers					
Port Register	Read	Write	Description	Offset	Page
P1_MODE[3:0]	+	+	P1 Mode	0x50	34
P1_TEST[4:0]	+	+	P1 Test	0x60	35
P1_STATE[5:0]	+	+	P1 State	0x51	35
P1_RECV_CONFIG[13:0] **	+	+	P1 Receive Configuration	0x49	36
P1_SEND_CONFIG[9:0] **	+	+	P1 Send Configuration	0x63	39
P1_SHORT_DROP[15:0]	Z*		P1 Too-Short Packet-Drop Counter	0x4A	41
P1_8B10B_ERROR[15:0]	Z*		P1 8b/10b Error Counter	0x52	41
Myrinet-Specific Registers					
Port Register	Read	Write	Description	Offset	Page
P1_MYRISEND[2:0] **	+	+	P1 Myrinet Send Control Symbols	0x62	41
Gigabit-Ethernet-Specific Registers					
Port Register	Read	Write	Description	Offset	Page
P1_GEX_CONTROL[15:0]	+	+	P1 GEX Control	0x40	43
P1_GEX_STATUS[15:0]	+		P1 GEX Status	0x41	43
P1_GEX_AN_ADVERTISEMENT[15:0]	+	+	P1 GEX AN Advertisement	0x42	43
P1_GEX_AN_LP_ABILITY_BP[15:0]	+		P1 GEX AN Link Partner Ability Base Page	0x43	43
P1_GEX_AN_EXPANSION[15:0]	+		P1 GEX AN Expansion	0x44	43
P1_GEX_AN_NP_TRANSMIT[15:0]	+	+	P1 GEX AN Next Page Transmit	0x45	43
P1_GEX_AN_LP_ABILITY_NP[15:0]	+		P1 GEX AN Link Partner Ability Next Page	0x46	43
P1_GEX_EXTENDED_STATUS[15:0]	+		P1 GEX Extended Status	0x47	43
P1_GE_CONFIG[2:0]	+	+	P1 GE Configuration	0x61	42
P1_GE_FALSE_CARRIER[15:0]	Z*		P1 GE False-Carrier Counter	0x48	42
InfiniBand-Specific Registers					
Port Register	Read	Write	Description	Offset	Page
P1_IB_CONFIG[3:0]	+	+	P1 IB Configuration	0x57	44
P1_IB_LDC[7:0]	Z*		P1 IB Link Downed Counter	0x54	44
P1_IB_LERC[7:0]	Z*		P1 IB Link Error Recovery Counter	0x53	44
P1_IB_SEC[15:0]	Z*		P1 IB Symbol Error Counter	0x52	44
P1_IB_LEC[15:0]	Z*		P1 IB Local Error Counter	0x55	44
P1_IB_REC[15:0]	Z*		P1 IB Remote Error Counter	0x56	44

Z* — these registers are reset to zero when read

** — only these registers are in Lanai XM

C. MESSAGE-PASSING EXAMPLE

This program is a simple but complete example, illustrating various aspects of chip initialization, event dispatch, and accessing the network interfaces of the Lanai XM chip.

```

-----

#include      "lanaiX_def.h"

#define RECV_BUFFER_SIZE      (64*1024)
#define RECV_BUFFERS          4
#define MAX_MESSAGE_SIZE      (4*1024)

#define ANY_VALUE              0

// some forward declarations

typedef void    (*HANDLER)();

extern void    initialize_PLLs();
extern void    initialize_arbitration();
extern void    initialize_channels();
extern void    initialize_dispatch();

extern void    no_backup_buffer0();
extern void    no_backup_buffer1();
extern void    packet_received0();
extern void    packet_received1();
extern void    free_to_send0();
extern void    free_to_send1();
extern void    packet_sent0();
extern void    packet_sent1();
extern void    dispatch_error();
extern void    dispatch_null();

extern int     state;
extern HANDLER dispatch_table[];

int     send_count0 = 0;           // PORT 0
int     recv_count0 = 0;           // messages sent
int     error_count0 = 0;         // messages received
RPD*    p0_rpl;                   // bad messages received
// points to RPL of the current receive buffer

int     send_count1 = 0;           // PORT 1
int     recv_count1 = 0;           // messages sent
int     error_count1 = 0;         // messages received
RPD*    p1_rpl;                   // bad messages received
// points to RPL of the current receive buffer

```

```

//
// we will be using BUFFER_SIZE_64K and BLOCK_SIZE_64; this means that there
// can be a maximum of 64K/64=1024 messages in a single receive buffer, and
// the size of the RPL is 1024*sizeof(RPD)=8K per receive buffer
//
// in general, the size of the RPL is BUFFER_SIZE/BLOCK_SIZE*sizeof(RPD)
//
char   recv_buffer0 [RECV_BUFFERS] [RECV_BUFFER_SIZE+RECV_BUFFER_SIZE/8]
      __attribute__((aligned(8)));
int    recv_buffer0_free = RECV_BUFFERS;      // number of free buffers
int    recv_buffer0_next = 0;                 // index of the next buffer
int    recv_buffer0_curr = 0;                 // index of the current buffer

char   recv_buffer1 [RECV_BUFFERS] [RECV_BUFFER_SIZE+RECV_BUFFER_SIZE/8]
      __attribute__((aligned(8)));
int    recv_buffer1_free = RECV_BUFFERS;      // number of free buffers
int    recv_buffer1_next = 0;                 // index of the next buffer
int    recv_buffer1_curr = 0;                 // index of the current buffer

// in this example, we are using hard-wired routes; the ports are sending
// to each other, and the switch connectivity is such that both routes are
// {-8, +8}; all messages are sent out of the same buffer

#define ROUTE_LENGTH  2

int    route0_length = ROUTE_LENGTH;
char   route0[ROUTE_LENGTH] = {0xb8, 0x88};

int    route1_length = ROUTE_LENGTH;
char   route1[ROUTE_LENGTH] = {0xb8, 0x88};

extern char   send_buffer[];

main()
{
    initialize_PLLs();
    initialize_arbitration();
    initialize_dispatch();
    initialize_channels();

    while(1)
    {
        DISPATCH_STATE = state;           // compute the next handler to
        WAIT_2_CYCLES;                     // run based on ISR and state
        (*dispatch_table[DISPATCH_INDEX/4])(); // call that handler
    }
}

```

```
void
initialize_PLLs ()
{
    int f, cpuc, rtc;          // measure the CPU clock frequency

    rtc = RTC;
    while (rtc == RTC)
        ;                    // wait for RTC transition #1
    CPUC = 0;

    rtc = rtc + 2048;
    while (rtc != RTC)
        ;                    // wait for RTC transition #2049
    cpuc = CPUC;

    f = cpuc/1024;           // CPU clocks per microsecond, i.e., MHz

    PLL = PLL_XM_M;         // Myricom-supplied configuration, for
    SYNC = SYNC_TABLE_XM_M[f]; // XM chip with Port 0 in Myrinet mode

    ITO = 20000;           // writing ITO clears TIMEO_INT, but there
    WAIT_1_CYCLE;         // may be a 1 clock cycle delay
    while (!(ISR & TIMEO_INT))
        ;                 // wait 10 ms for the PLLs to stabilize
}
```

```
void
initialize_arbitration()           // the recommended setup for Lanai XM
{
    ARBITER_SLOT = 0x00;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x01;    ARBITER_CODE = CODE_PO_S;
    ARBITER_SLOT = 0x02;    ARBITER_CODE = CODE_P1_R;
    ARBITER_SLOT = 0x03;    ARBITER_CODE = CODE_P1_S;
    ARBITER_SLOT = 0x04;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x05;    ARBITER_CODE = CODE_COPY;
    ARBITER_SLOT = 0x06;    ARBITER_CODE = CODE_PO_S;
    ARBITER_SLOT = 0x07;    ARBITER_CODE = CODE_P1_R;
    ARBITER_SLOT = 0x08;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x09;    ARBITER_CODE = CODE_P1_S;
    ARBITER_SLOT = 0x0A;    ARBITER_CODE = CODE_COPY;
    ARBITER_SLOT = 0x0B;    ARBITER_CODE = CODE_PO_S;
    ARBITER_SLOT = 0x0C;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x0D;    ARBITER_CODE = CODE_P1_R;
    ARBITER_SLOT = 0x0E;    ARBITER_CODE = CODE_P1_S;
    ARBITER_SLOT = 0x0F;    ARBITER_CODE = CODE_COPY;
    ARBITER_SLOT = 0x10;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x11;    ARBITER_CODE = CODE_PO_S;
    ARBITER_SLOT = 0x12;    ARBITER_CODE = CODE_P1_R;
    ARBITER_SLOT = 0x13;    ARBITER_CODE = CODE_P1_S;
    ARBITER_SLOT = 0x14;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x15;    ARBITER_CODE = CODE_COPY;
    ARBITER_SLOT = 0x16;    ARBITER_CODE = CODE_PO_S;
    ARBITER_SLOT = 0x17;    ARBITER_CODE = CODE_P1_R;
    ARBITER_SLOT = 0x18;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x19;    ARBITER_CODE = CODE_P1_S;
    ARBITER_SLOT = 0x1A;    ARBITER_CODE = CODE_COPY;
    ARBITER_SLOT = 0x1B;    ARBITER_CODE = CODE_PO_S;
    ARBITER_SLOT = 0x1C;    ARBITER_CODE = CODE_PO_R;
    ARBITER_SLOT = 0x1D;    ARBITER_CODE = CODE_P1_R;
    ARBITER_SLOT = 0x1E;    ARBITER_CODE = CODE_P1_S;
    ARBITER_SLOT = 0x1F;    ARBITER_CODE = CODE_JTAG;
}
```

```

//
// The following setup can be used to ensure fair dispatch for two events:
//
// ----- dispatch table -----
// high priority:      event0 & event1 & !state => event0_handler()
// mid  priority:      event1                  => event1_handler()
// low  priority:      event0                  => event0_handler()
// -----
//
// In event0_handler() and event1_handler(), the state bit is toggled to give
// priority to the other handler.
//

int     state = 0;          // the 4 least-significant bits of "state"
                          // are used for fairness, one bit per event pair

void
initialize_dispatch ()
{
// not using any AISR bits
    AISR_ON  = 0;
    AISR_OFF = 0;

// we don't want to run out of the receive buffers,
// so the NO_BACKUP bits have the highest priority, 0-2

    DISPATCH_ISR_ON  = P0_NO_BACKUP | P1_NO_BACKUP;
    DISPATCH_ISR_OFF = 0;
    DISPATCH_STATE_ON = 0;
    DISPATCH_STATE_OFF = 0x1;    // use state bit 0 for fairness
    DISPATCH_CONFIG = 0;
    WAIT_1_CYCLE;

    DISPATCH_ISR_ON  = P1_NO_BACKUP;
    DISPATCH_ISR_OFF = 0;
    DISPATCH_STATE_ON = 0;
    DISPATCH_STATE_OFF = 0;
    DISPATCH_CONFIG = 1;
    WAIT_1_CYCLE;

    DISPATCH_ISR_ON  = P0_NO_BACKUP;
    DISPATCH_ISR_OFF = 0;
    DISPATCH_STATE_ON = 0;
    DISPATCH_STATE_OFF = 0;
    DISPATCH_CONFIG = 2;
    WAIT_1_CYCLE;

// handling of the received packets has priority over
// sending, so the PACKET_RCVD bits have the next highest priority, 3-5

    DISPATCH_ISR_ON  = P0_PACKET_RCVD | P1_PACKET_RCVD;
    DISPATCH_ISR_OFF = 0;
    DISPATCH_STATE_ON = 0;
    DISPATCH_STATE_OFF = 0x2;    // use state bit 1 for fairness
    DISPATCH_CONFIG = 3;
    WAIT_1_CYCLE;

```

```
DISPATCH_ISR_ON = P1_PACKET_RCVD;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0;
DISPATCH_CONFIG = 4;
WAIT_1_CYCLE;
```

```
DISPATCH_ISR_ON = P0_PACKET_RCVD;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0;
DISPATCH_CONFIG = 5;
WAIT_1_CYCLE;
```

```
// in the simple example here, we always send from the same send_buffer,
// so this step isn't really necessary; in general, however, we would
// have to recycle the send buffers
```

```
DISPATCH_ISR_ON = P0_PACKET_SENT | P1_PACKET_SENT;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0x4; // use state bit 2 for fairness
DISPATCH_CONFIG = 6;
WAIT_1_CYCLE;
```

```
DISPATCH_ISR_ON = P1_PACKET_SENT;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0;
DISPATCH_CONFIG = 7;
WAIT_1_CYCLE;
```

```
DISPATCH_ISR_ON = P0_PACKET_SENT;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0;
DISPATCH_CONFIG = 8;
WAIT_1_CYCLE;
```

```
// the lowest priority, 9-11, is reserved for sending
```

```
DISPATCH_ISR_ON = P0_SEND_READY | P1_SEND_READY;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0x8; // use state bit 3 for fairness
DISPATCH_CONFIG = 9;
WAIT_1_CYCLE;
```

```
DISPATCH_ISR_ON = P1_SEND_READY;
DISPATCH_ISR_OFF = 0;
DISPATCH_STATE_ON = 0;
DISPATCH_STATE_OFF = 0;
DISPATCH_CONFIG = 10;
WAIT_1_CYCLE;
```



```
void
initialize_channels ()
{
//
// configure PORT0
//
    PO_RPL_CONFIG = ( ENABLE_BUFFER_DROP // a must for the X port
                    | BUFFER_SIZE_64K // 64K for buffer
                    | BLOCK_SIZE_64 // 8K for RPL
                    | MAX_PACKET_8K // (4096+5) rounded up
                    | STOP_SIZE_32K // worst case receive-buffer utilization
                    | PORT_ENABLE // this bit turns the port on
                    );
    PO_SEND_FREE_LIMIT = 2; // we send messages with 2 send
                          // descriptors (4 words), the
                          // route and the payload, so we
                          // set PO_SEND_FREE_LIMIT to 2;
                          // we are free to send when
                          // PO_SEND_FREE_COUNT >=
                          // (PO_SEND_FREE_LIMIT + 2) = 4

    port_reg_write ( PO_SEND_CONFIG,
                    ( ENABLE_CRC8 // enable CRC8 on send
                    | ENABLE_CRC32 // enable CRC32 on send
                    )
                    );
    port_reg_write ( PO_RECV_CONFIG,
                    ( ENABLE_CRC8 // enable CRC8 on receive
                    | ENABLE_CRC32 // enable CRC32 on receive
                    )
                    );
    port_reg_write (PO_MYRI_SEND, M_AUTO_BEAT); // send BEAT every 10us

    port_reg_write (PO_MODE, MODE_M); // put X-port in Myrinet mode
    while (!(ISR & PO_LINK_READY))
        ; // wait for the link to come up

    p0_rpl = (RPD*)&(recv_buffer0[0][RECV_BUFFER_SIZE]); // prime the RPL pointer
}
```

```

//
// configure PORT1
//
P1_RPL_CONFIG = ( BUFFER_SIZE_64K      // 64K for buffer
                  | BLOCK_SIZE_64     // 8K for RPL
                  | MAX_PACKET_8K     // (4096+5) rounded up
                  | PORT_ENABLE       // this bit turns the port on
                  );
P1_SEND_FREE_LIMIT = 2;                // we send messages with 2 send
                                        // descriptors (4 words), the
                                        // route and the payload, so we
                                        // set P1_SEND_FREE_LIMIT to 2;
                                        // we are free to send when
                                        // P1_SEND_FREE_COUNT >=
                                        // (P1_SEND_FREE_LIMIT + 2) = 4

port_reg_write ( P1_SEND_CONFIG,
                 ( ENABLE_CRC8        // enable CRC8 on send
                 | ENABLE_CRC32      // enable CRC32 on send
                 )
               );
port_reg_write ( P1_RECV_CONFIG,
                 ( ENABLE_CRC8        // enable CRC8 on receive
                 | ENABLE_CRC32      // enable CRC32 on receive
                 | SAN_BEAT_ENABLE   // turn on the HA features
                 | SAN_ILGL_ENABLE   // turn on the HA features
                 | SAN_WINDOW        // required for SAN links
                 )
               );
port_reg_write (P1_MYRI_SEND, M_AUTO_BEAT); // send BEAT every 10us

// this initialization procedure ignores the first two, bogus messages that
// the SAN interface injects when it brings the port up; they will be
// detected as incorrect and included in the "error_count1"

while (!(ISR & P1_LINK_READY))
    ;                                // wait for the link to come up

p1_rpl = (RPD*)&(recv_buffer1[0][RECV_BUFFER_SIZE]); // prime the RPL pointer
}

```

```
void
no_backup_buffer0 ()
{
    if (recv_buffer0_free > 0)
    {
        P0_RB = &(recv_buffer0[recv_buffer0_next][0]);
        recv_buffer0_next = (recv_buffer0_next + 1) % RECV_BUFFERS;
        recv_buffer0_free--;
    }
    else
        // there are no free buffers, they
        // must be full of received packets
    {
        if (ISR & P0_PACKET_RCVD)
            packet_received0();
    }

    state |= 0x1;
    // set state bit to ensure fairness
}

void
no_backup_buffer1 ()
{
    if (recv_buffer1_free > 0)
    {
        P1_RB = &(recv_buffer1[recv_buffer1_next][0]);
        recv_buffer1_next = (recv_buffer1_next + 1) % RECV_BUFFERS;
        recv_buffer1_free--;
    }
    else
        // there are no free buffers, they
        // must be full of received packets
    {
        if (ISR & P1_PACKET_RCVD)
            packet_received1();
    }

    state &= ~(0x1);
    // clear state bit to ensure fairness
}
```

```

void
packet_received0 ()
{
    RPD* packet = p0_rpl++;

    // the packet should end with at least 5 zero bytes (4 bytes CRC32, 1 byte CRC8)
    // it should not be marked INVALID, LINK, or MARKED_BAD
    // the length should be between 6 and (MAX_MESSAGE_SIZE+5)

    if ( (((packet->length & DESC_ZEROES) >> 25) < 5)
        || (packet->length & DESC_INVALID)
        || (packet->length & DESC_LINK)
        || (packet->length & DESC_MARKED_BAD)
        || ((packet->length & DESC_LENGTH) < 6)
        || ((packet->length & DESC_LENGTH) > (MAX_MESSAGE_SIZE+5))
    )
    {
        error_count0++;
    }

    if (packet->length & DESC_LAST) // this is the last packet
    { // in this receive buffer
        recv_buffer0_free++;
        recv_buffer0_curr = (recv_buffer0_curr + 1) % RECV_BUFFERS;
        p0_rpl = (RPD*)&(recv_buffer0[recv_buffer0_curr][RECV_BUFFER_SIZE]);
    }

    recv_count0++;
    PO_RECV_COUNT = ANY_VALUE; // decrement PO_RECV_COUNT

    state |= 0x2; // set state bit to ensure fairness
}

```

```

void
packet_received1 ()
{
    RPD* packet = p1_rpl++;

    // the packet should end with at least 5 zero bytes (4 bytes CRC32, 1 byte CRC8)
    // it should not be marked INVALID, LINK, or MARKED_BAD
    // the length should be between 6 and (MAX_MESSAGE_SIZE+5)

    if ( (((packet->length & DESC_ZEROES) >> 25) < 5)
        || (packet->length & DESC_INVALID)
        || (packet->length & DESC_LINK)
        || (packet->length & DESC_MARKED_BAD)
        || ((packet->length & DESC_LENGTH) < 6)
        || ((packet->length & DESC_LENGTH) > (MAX_MESSAGE_SIZE+5))
    )
    {
        error_count1++;
    }

    if (packet->length & DESC_LAST) // this is the last packet

```

```
{
    // in this receive buffer
    recv_buffer1_free++;
    recv_buffer1_curr = (recv_buffer1_curr + 1) % RECV_BUFFERS;
    p1_rpl = (RPD*)&(recv_buffer1[recv_buffer1_curr][RECV_BUFFER_SIZE]);
}

recv_count1++;
P1_RECV_COUNT = ANY_VALUE;           // decrement P1_RECV_COUNT

state &= (~0x2);                     // clear state bit to ensure fairness
}
```

```
void
packet_sent0 ()
{
    P0_SEND_COUNT = ANY_VALUE;       // decrement P0_SEND_COUNT

    state |= 0x8;                   // set state bit to ensure fairness
}
```

```
void
packet_sent1 ()
{
    P1_SEND_COUNT = ANY_VALUE;       // decrement P1_SEND_COUNT

    state &= ~(0x8);               // clear state bit to ensure fairness
}
```

```
void
free_to_send0 ()
{
    int length;

    CRC32_WORD = CPUC;                // CRC32 can be used as a
    length = (CRC32 % MAX_MESSAGE_SIZE) + 1; // pseudo-random number generator
                                        // length is from 1 to MAX

    PO_SEND_POINTER = (void*)route0; // in this example, always
    PO_SEND_LENGTH = route0_length | DESC_ROUTE; // use the same route

    PO_SEND_POINTER = send_buffer; // in this example, always
    PO_SEND_LENGTH = length | DESC_LAST; // use the same buffer

    send_count0++;

    state |= 0x4; // set state bit to ensure fairness
}

```

```
void
free_to_send1 ()
{
    int length;

    CRC32_WORD = CPUC;
    length = (CRC32 % MAX_MESSAGE_SIZE) + 1;

    P1_SEND_POINTER = (void*)route1;
    P1_SEND_LENGTH = route1_length | DESC_ROUTE;

    P1_SEND_POINTER = send_buffer;
    P1_SEND_LENGTH = length | DESC_LAST;

    send_count1++;

    state &= ~(0x4); // clear state bit to ensure fairness
}

```

```
void
dispatch_error ()
{
    while(1); // this should never happen
}

```

```
void
dispatch_null ()
{
    // this is invoked when there is nothing to do
}

```

D. PCI-DMA EXAMPLE

This program illustrates the enqueueing/dequeueing mechanism of the PCI DMA interface.

```

-----

#include      "lanaiX_def.h"

DRD* drd_free;          // linked list of free DMA request descriptors (DRDs)

DRD* drd_last[4];      // the four DMA-channel queues
DRD* drd_first[4];

DRD* drd_stage_last[4]; // the four pre-staging linked lists, in case we
DRD* drd_stage_first[4]; // want to append more than one DRD at a time

void
dma_enqueue (int channel,          // one of four DMA channels
             int initiate,        // initiate-DMA/pre-stage-only flag
             int read,           // DMA direction
             int end,            // DMA_END flag
             int flush,         // DMA_FLUSH flag
             void* lar,         // DMA Lanai address
             unsigned int len,   // DMA length
             unsigned int eal,   // DMA PCI address
             unsigned int eah,   // DMA PCI64 address
             unsigned short offset) // DMA checksum offset
{
    DRD* drd;

    drd = drd_free;          // get a new DRD from the free list,
    drd_free = drd->next;    // assumes there is always one available

    drd->next = 0;           // fill in the DRD
    drd->len = len
        | ((read) ? DMA_READ : 0)
        | ((end) ? DMA_END : 0)
        | ((flush) ? DMA_FLUSH : 0)
        ;
    drd->eal = eal;
    drd->eah = eah;
    drd->lar = lar;
    drd->offset = offset;

                                // append to the pre-staging linked list
    if (drd_stage_first[channel]) // the list is non-empty
        drd_stage_last[channel]->next = (DRD*) ( ((unsigned)drd)
                                                    | DMA_VALID
                                                    );
    else                          // the list is empty
        drd_stage_first[channel] = drd;
    drd_stage_last[channel] = drd;
}

```

```

if (initiate)                // append to the DMA queue
{
    drd = drd_stage_first[channel];

    if (drd_first[channel])    // the queue is non-empty
        drd_last[channel]->next = (DRD*) ( ((unsigned)drd)
            | DMA_VALID
            | DMA_APPEND
            );

    else                        // the queue is empty
        drd_first[channel] = drd;
    drd_last[channel] = drd_stage_last[channel];

    DMA_POINTER(channel) = drd;    // notify the DMA engine

    drd_stage_first[channel] = 0;    // clear pre-staging area
}
}

//
// The DMA dequeue function, typically invoked in response to an ISR DMA_DONE bit
//
void
dma_dequeue (int channel)
{
    int end;
    DRD* drd;

    DMA_COUNT(channel) = ANY_VALUE;    // decrement DMA_COUNT

    do
    {
        drd = drd_first[channel];    // start from the head of the DMA queue
        end = drd->len & DMA_END;    // check DMA_END flag

        drd_first[channel] = (DRD*) ( ((unsigned)(drd->next))
            & DMA_NEXT
            );    // remove one DRD
        drd->next = drd_free;    // recycle it
        drd_free = drd;

    } while (!end);    // keep going until we see DMA_END
}

```