

PCI64 Programmer's Documentation

Last significant revision: 6 May 2001

This documentation is also available as a [pdf file](#).

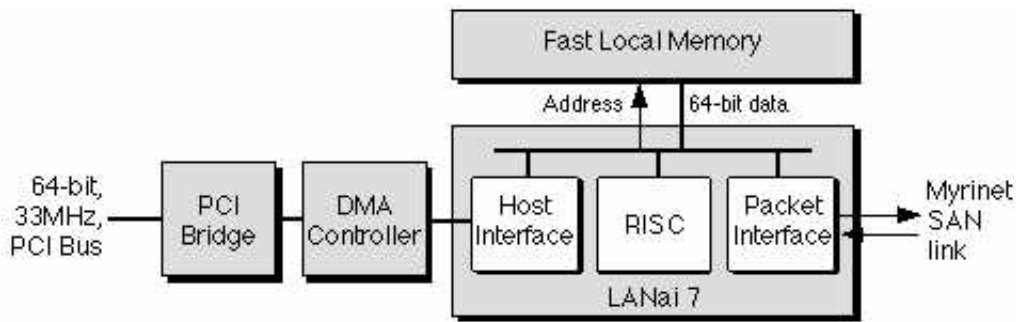
1. Introduction & Scope

This documentation is provided principally for people who write their own Myrinet Control Programs (MCPs) and/or device drivers. Customers who use Myricom software or third-party software will generally not find this information to be useful.

This documentation applies to the following Myricom products (listed chronologically):

PCI64

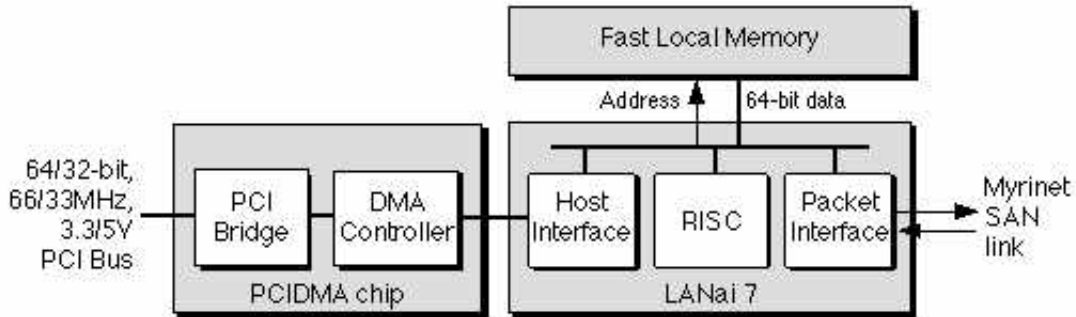
Product codes	Description	LANai	Dates produced
M2M-PCI64-2	64-bit, 33MHz, Myrinet-SAN/PCI interface (PCI short card)	7.0 or 7.1	4/99 - 8/99
M2L-PCI64-2	64-bit, 33MHz, Myrinet-LAN/PCI interface (PCI short card)	7.0 or 7.1	4/99 - 8/99



In the block diagram above, the LAN (M2L) version of this interface includes, in addition, LAN-SAN conversion between the SAN port of the LANai chip and the Myrinet-LAN port on the PCI faceplate.

PCI64A

Product codes	Description	LANai	Dates produced
M2M-PCI64A-2 M2M-PCI64A-4	64-bit, 66MHz, Myrinet-SAN/PCI interface (PCI short card)	7.2	9/99 - 6/00
M2L-PCI64A-2 M2L-PCI64A-4	64-bit, 66MHz, Myrinet-LAN/PCI interface (PCI short card)	7.2	9/99 - 6/00
M2M-PMC64A-2 M2M-PMC64A-4	64-bit, 66MHz, Myrinet-SAN/PCI interface (PCI Mezzanine Card)	7.2	11/99 - 6/00



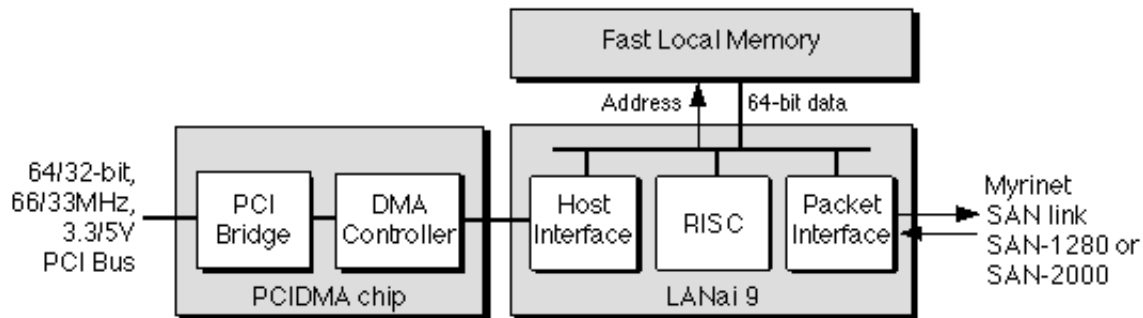
In the block diagram above, the LAN (M2L) version of this interface includes, in addition, LAN-SAN conversion between the SAN port of the LANai chip and the Myrinet-LAN port on the PCI faceplate.

PCI64B

Product codes	Description	LANai	Dates produced
M3M-PCI64B-2 M3M-PCI64B-4 M3M-PCI64B-8	64-bit, 66MHz, Myrinet-SAN/PCI interface (PCI short card)	9.0 , 9.1, or 9.2	5/00 -
M2L-PCI64B-2 M2L-PCI64B-4	64-bit, 66MHz, Myrinet-LAN/PCI interface (PCI short card)	9.0 , 9.1, or 9.2	6/00 -
M3M-PMC64B-2 M3M-PMC64B-4	64-bit, 66MHz, Myrinet-SAN/PCI interface (PCI Mezzanine Card)	9.0 , 9.1, or 9.2	7/00 -
M3S-PCI64B-2 M3S-PCI64B-4	64-bit, 66MHz, Myrinet-2000-Serial-Link/PCI interface (PCI short card)	9.0 , 9.1, or 9.2	8/00-
M3F-PCI64B-2 M3F-PCI64B-4 M3F-PCI64B-8	64-bit, 66MHz, Myrinet-2000-Fiber/PCI interface (PCI short card)	9.0 , 9.1, or 9.2	8/00-

PCI64C

Product codes	Description	LANai	Dates produced
M3M-PCI64C-2	64-bit, 66MHz, Myrinet-SAN/PCI interface (PCI short card)	9.2	2/01-
M3M-PMC64C-2	64-bit, 66MHz, Myrinet-SAN/PCI interface (PCI Mezzanine Card)	9.2	2/01-
M3S-PCI64C-2	64-bit, 66MHz, Myrinet-2000-Serial-Link/PCI interface (PCI short card)	9.2	2/01-
M3F-PCI64C-2	64-bit, 66MHz, Myrinet-2000-Fiber/PCI interface (PCI short card)	9.2	2/01-



In the block diagram above, the LAN (M2L) version of this interfaces includes, in addition, LAN-SAN conversion between the SAN port of the LANai chip and the Myrinet-LAN port on the PCI faceplate, and the Myrinet-2000 serial (M3S) and fiber (M3F) versions include, in addition, Serial-SAN conversion.

Myricom will introduce other products in this PCI64 series, but is committed to keeping the basic architecture and programming model constant for an extended period.

The PCI64 and PCI64A products are program-compatible except for:

- The PCI64A interfaces require an extra step in initialization to determine the frequency of the PCI bus, and to set the clock multiplier in the LANai 7 chip accordingly.
- The LANai 7.0 and 7.1 chips in the PCI64 interfaces have a minor erratum in the "sig" bits. The LANai 7.2 or 7.3 in the PCI64A interfaces have no known errata.

The PCI64B products differ from the PCI64A in:

- The 133MHz LANai-9 RISC has a different instruction set from the 66MHz LANai-7 RISC; however, the LANai 9 RISC is source-code compatible with the LANai 7 RISC.
- The control of the LANai 9 SAN port is internal to the LANai 9 chip, whereas certain of these functions were performed in the LANai 7 SAN port by an external microcontroller. There are, accordingly, minor differences in the control of the SAN-2000 port on the LANai 9 versus the SAN-1280 port on the LANai 7 (see section 4).

The PCI64C products differ from the PCI64B only in the maximum frequency of the LANai RISC and memory being 200MHz instead of 133 MHz.

1.1 Block Diagram and Organization

As illustrated in block diagrams above, the interface consists of:

- PCI Bridge.
- DMA Controller.
- Myricom LANai 7 or LANai 9 chip, which includes a RISC processor.
- The LANai's local memory: 2 , 4, or 8 MegaBytes of synchronous SRAM.

The LANai chip, which is the core of the interface, contains a RISC processor, DMA logic (packet interface) to/from the network, "E bus" interface logic to/from the host, timers, and local configuration registers. These LANai chips are described in detail in separate documents that can be found at:

<http://www.myri.com/vlsi/LANai7.pdf>

<http://www.myri.com/vlsi/LANai9.pdf>

1.2 Relevant Standards and Specifications

These 64-bit Myrinet/PCI interfaces are compliant with the latest PCI specifications (version 2.2), except that

- the latency timer is not implemented.

The SAN or LAN port of these interfaces is compliant at the Physical and Data Link levels with the ANSI/VITA 26-1998 Standard. The last VITA Standards Organization draft of this ANSI Standard can be found linked from:

<http://www.myri.com/open-specs/>

1.3 Software

All of these PCI64 interfaces are supported by the current GM release and software-development tools found at:

<http://www.myri.com/scs/>

Here are a copies of the current [lanai7_def.h](#) and [lanai9_def.h](#) files for those who may not wish to download the entire GM source distribution.

1.4 Organization of this Documentation

This documentation is organized according to the functions performed in the

- PCI Bridge
- DMA controller
- Myrinet port status and control

2. PCI Bridge

2.1 Configuration Space & Memory Layout

The PCI64 interfaces are standard, PCI-2.2-compliant implementations except for the omission of the latency timer. The card supports a PCI configuration space and a single function/base_addr region on the card. All the memory and registers on the card are memory mapped. There are no I/O port mappings on the card.

Configuration Space

Vendor ID 0x14C1
 Device ID 0x8043
 Revision ID 0x3

All other required configuration registers are implemented. The remainder of the card is memory mapped at the address found in `Base_Address_Register_0/Base_Address_Register_1`. The card occupies 16MB of PCI space. The card implements a 64-bit Base Address Register, but will also function properly when programmed with a 32-bit address, per the PCI specifications. A mechanical switch on the interface board allows switching to a 32-bit Base Address Register to accommodate operating systems and firmware that either cannot properly load a 64-bit address into a 64-bit Base Address Register, or cannot correctly write a 32 bit address into a 64-bit Base Address Register. Click [here](#) for details.

Memory Layout - Byte offset from beginning of the 16MB space:

0x00000000 - LANai SRAM space: 8 Megabytes maximum allowed
 0x00800000 - Configuration register: Remap of config space (64 bytes)
 0x00800040 - Control register: (4 bytes)
 0x00800100 - Bank of pointers for DMA and Doorbell: (64 bytes)
 0x00804000 - LANai special registers: (16K bytes)
 0x00808000 - Doorbell Region: (8 Megabytes - 32K bytes, write-only)
 0x00A00000 - EEPROM board information (4 Megabytes, read-only)
 0x01000000 - End of mapped space

The overlap between the EEPROM board information and the doorbell region is intentional, and is possible because the EEPROM is read-only, while the Doorbell Region is write-only.

2.2 LANai SRAM (Offset 0x00000000)

This is the memory of the LANai processor. It should be accessed as big-endian memory since the LANai is a big-endian device.

2.3 Control register (Offset 0x00800040)

This register is the main control register for the board. It is a little-endian register.

Bit	Write Operation	Read Meaning
31	Allow LANai to run	LANai is running
30	Hold LANai in reset	LANai is stopped
29	Allow E-bus to run	E-bus is running
28	Hold E-bus in reset	E-bus is disabled
27	Allow Board to run	Board is running
26	Hold Board in reset	Board is disabled
25	Allow Interrupts	Interrupts enabled
24	Disable Interrupts	Interrupts disabled

Bit	Write Operation	Read Meaning
23	DMA trigger 3	-
22	DMA trigger 2	-
21	DMA trigger 1	-
20	DMA trigger 0	-
19	-	-
18	-	-
17	-	-
16	-	-

Bit	Write Operation	Read Meaning
15	-	-
14	-	correct RST/REQ64 sequence detected
13	do not use	-
12	do not use	-
11	do not use	-
10	do not use	-
09	do not use	-
08	-	-

Bit	Write Operation	Read Meaning
07	force 64-bit mode	forcing 64-bit mode
06	force 32-bit mode	forcing 32-bit mode
05	-	-
04	enable doorbell limit checking	-
03	enable multi-queue doorbells	-
02	queue size select 2	-
01	queue size select 1	-
00	queue size select 0	-

Note: A Write Operation with bits 07 and 06 both 1 stops forcing either mode.

The 8 high-order bits control the gross operation of the board. The operations are enabled when a "1" is written to the particular field. You do not need to (and should not) repeat commands to "preserve" the state. Only a "1" bit will activate a command. For instance, if the board is out of reset (bit 27 == 1), then to remove ebus reset, you only need to write (reg = (1<<29)).

The DMA trigger bits are discussed in the DMA Controller section (3).

The "do not use" bits are used for production diagnostics, and should not be written with a "1" value.

The Queue control bits are discussed the "doorbell" FIFO queue section (3.3).

Bit 14 indicates whether the board detected a proper power-up sequence for a 64-bit slot. If not, the card will operate

in 32-bit mode. Bit 7 and bit 6 can be used to force the card to the correct mode if the power-up sequence is not correct.

Bits 04-00 are a normal register that writes destructively with all binary values. Care must be taken to preserve their values when manipulating other bits in the Control register. Use of a byte-write operation when manipulating bits in other bytes of the control register is appropriate for avoiding accidental overwrites of the values stored in bits 04-00.

2.4 Initialization

To bring the board out of reset, you should implement the following psuedo-code.

```

1) Turn OFF pci_config_dma_master bit
2) Turn ON board_reset
   control_reg = (1<<26);
3) Turn OFF board_reset, ON LANai reset , ON EBus reset
   control_reg = ((1<<27) | (1<<30) | (1<<28))
4) Write a temporary clockval to the LANai clockval register
   lanai_special->clockval = 0x80;
5) Delay 10millisec
6) Read the clockval from the EEPROM.
7) Write a 1x clock_value into the LANai.
8) Delay 10millisec
9) Turn OFF EBus reset
   control_reg = (1<<29);
10) Delay 10millisec
11) Measure the speed of the LANai processor.
    For LANai9, use CPUC and RTC.
    For LANai7, load a rate measuring program.
12) Calculate a new clock_value multiplier using
    the LANai rate and the max_lanai_rate from the EEPROM.
13) Turn ON EBus reset
   control_reg = (1<<28);
14) Write the new clock_value into the LANai
   lanai_special->clockval = clockval_read_from_eeprom;
15) Delay 10millisec
16) Turn OFF EBus reset
   control_reg = (1<<29);
17) Load code into the LANai SRAM
18) Turn ON pci_config_dma_master_bit
19) Turn OFF lanai_reset
   control_reg = (1<<31);

```

The PCI64A, PCI64B, and PCI64C cards are capable of running in either a 33MHz or a 66MHz PCI slot (or at any frequency in between). The LANai chip derives its internal clocks, including the processor clock, as a multiple of the PCI bus. The clockval multipliers are

(eeprom_clockval & 0xFFFFFFFF8F) 0x80	1.0 X PCI
(eeprom_clockval & 0xFFFFFFFF8F) 0x90	1.5 X PCI
(eeprom_clockval & 0xFFFFFFFF8F) 0xA0	2.0 X PCI
(eeprom_clockval & 0xFFFFFFFF8F) 0xB0	2.5 X PCI
(eeprom_clockval & 0xFFFFFFFF8F) 0xC0	3.0 X PCI

You should read out the `max_lanai_speed`, which is the maximum *processor* clock rate, from the eeprom to determine the maximum multiplier. For the LANai 7, the processor operates at the frequency selected. For the LANai 9, however, the processor operates at 2x the frequency selected. Thus, for example, if the `max_lanai_speed` for a LANai-9 board is 134MHz, and the PCI clock is 66MHz, the appropriate clockval multiplier is 1.

C code for this initialization can be found in the GM software available at <http://www.myri.com/scs/>. In the GM software, look at gm/drivers/gm_instance.c: gm_update_clockval_rev3()

2.5 DMA and FIFO Bank of Pointers (Offset 0x00800100)

This region of the board provides storage for pointers for the DMA control blocks and the doorbell FIFO region.

Offset	Function
0x00	DMA_CHANNEL_0 chain base
0x04	DMA_CHANNEL_1 chain base
0x08	DMA_CHANNEL_2 chain base
0x0C	DMA_CHANNEL_3 chain base
0x10	FIFO base pointer 0
0x14	FIFO base pointer 1
0x18	FIFO base pointer 2
0x1C	FIFO base pointer 3

The exact requirements and usage of these pointers is found in the DMA Controller section (3) and the Doorbell FIFO section (3.3). These are PCI registers and are little-endian.

2.6 LANai Special Registers (Offset 0x00804000)

This memory area provides host access to the LANai special registers. Details on these registers may be found in the LANai 7 documentation.

2.7 Doorbell FIFO region (Offset 0x00808000) (write-only)

When properly enabled and initialized, any writes to this region will store the address and the data of a write operation into a queue in the LANai SRAM. Further details may be found in the Doorbell FIFO section (3.3).

2.8 EEPROM (Offset 0x00A00000) (read-only)

Each Myrinet board contains an EEPROM that is programmed with information about the board. This information is accessed from offset 0 of the EEPROM section, and is listed below.

```

/* NOTE: the EEPROMs are programmed big-endian */
/* so the cpu version will look like 0x07 0x02 etc. */
struct MYRINET_EEPROM { /* byte offset - meaning */
uint32 LANai_clockval; /* 0 - 32-bit value to be used to initialize the LANai chip */
uint16 LANai_cpu_version; /* 4 - 0x0701 would be LANai 7.1 */
uint8 LANai_board_id[6]; /* 6 - 48-bit MAC Address */
uint32 LANai_sram_size; /* 12 - LANai SRAM size in bytes */
uint8 fpga_version[32]; /* 16 - Used to encode information about FPGA, if any */
uint8 more_version[16]; /* 48 - Used to encode information about FPGA, if any */
uint16 delay_line_value; /* 64 - Obsolete - unused/
uint16 board_type; /* 66 */
uint16 bus_type; /* 68 */
uint16 product_code; /* 70 */
uint32 serial_number; /* 72 - Myricom serial number */
uint8 board_label[32]; /* 76 - Expected board label */
uint16 max_lanai_speed; /*108 - Maximum LANai processor speed (MHz) */
uint16 future_use[7]; /*110 */
uint32 unused_4_bytes; /*124 */
};

```

Here are *typical* values read from a PCI64B board:

```

lanai_clockval    = 0x052082a0
lanai_cpu_version = 0x0900
lanai_board_id   = 00:60:dd:7f:bf:d6
lanai_sram_size  = 0x00200000 (2048K bytes)
fpga_version     = "Thu Dec  9 16:13:40 1999"
more_version     = ""
delay_line_value = 0x0000
board_type       = 0x0003 (GM_MYRINET_BOARD_TYPE_L5)
bus_type        = 0x0002 (GM_MYRINET_BUS_PCI)
product_code    = 0x003e
serial_number    = 52689
max_lanai_speed = 134

```

You should, of course, use the actual values read, not these typical values.

3. DMA Controller

PCI64 interfaces contain a versatile DMA controller. The controller uses chains of control blocks stored in the LANai memory to initiate DMA-mastering operations. To support zero-copy APIs efficiently, the DMA operations can be performed with arbitrary byte counts and alignments.

3.1 DMA Control Block

The structure of a DMA control block in the LANai memory, as seen from the perspective of the big-endian LANai processor, is given below.

```

struct DMA_BLOCK {
    volatile uint32 next;    /* next block in chain pointer */
    uint16 spare;          /* unused */
    uint16 csum;           /* ones complement cksum of this block */
    uint32 len;            /* byte count */
    uint32 lar;            /* lanai address */
    uint32 eah;           /* high PCI address - unused for 32-bit PCI */
    uint32 eal;           /* low 32-bit PCI address */
};

```

The low-order bits of the "next" pointer are used as flags to determine the DMA direction, whether this is the last block, and whether to give the LANai a WAKE_INT when this block has completed. Note that this means that all DMA control blocks must reside on an 8-byte boundary.

Bit	Name	Function
0	D	Direction (1=host->LANai, 0=LANai->host)
1	T	Terminal (1=terminal, 0=not-terminal)
2	I	Interrupt (1=set WAKE_INT_BIT, 0=no set)

The terminal bit indicates that this is the end of the chain. If the terminal bit of a DMA control block is already set when it is examined by the DMA engine, the DMA operation specified by that block is NOT executed.

The DMA controller holds 4 pointer registers at locations in PCI address space:

Byte Offset	Function
0x00800100	DMA channel 0 chain base address
0x00800104	DMA channel 1 chain base address
0x00800108	DMA channel 2 chain base address

```
0x0080010C DMA channel 3 chain base address
```

These registers are not true 32-bits registers. They only have enough bits to span the 8 Megabyte potential LANai-7 address space on the interface. The high-order bits and the lowest 3 bits in the registers are not stored. These are little-endian PCI registers.

3.2 DMA Operation

To use a particular channel of the DMA controller, the channel's chain base address register must be initialized to a LANai memory location that is free, and that will be used to hold a DMA control block. This address is an offset from the base of the LANai memory, *i.e.*, location 0 would be written as "0." (Note: If a pointer register is written and then read, some top and bottom address bits will not necessarily have the same value read as what was written.) The control registers must be written from the host since the LANai has no ability access the PCI control registers on the card.

Once the chain base address has been written from the host, and the control block has been properly initialized, the DMA can be triggered from the host or from the LANai.

There are 4 trigger bits in the control register, one corresponding to each channel. These bits are writeable from the host. Writing a "1" into any of these bits will signal the controller to pick up the 1st of this channel's control blocks from the LANai memory.

DMA channel	Control register bit
0	20 (0x00100000)
1	21 (0x00200000)
2	22 (0x00400000)
3	23 (0x00800000)

DMA channels 0 and 2 can be controlled from the LANai. After writing to the control block, the LANai processor should set

```
PULSE = 0x2; /* for DMA channel 0 */
```

or

```
PULSE = 0x4; /* for DMA channel 2 */
```

A chain is terminated by either:

- a block with a terminal bit set, or
- a block with the next pointer set to 0 (all 0s word).

The DMA engine will execute each block until it sees a terminal block. It will not perform the DMA described in the terminal block. Instead, it will go to sleep on that channel until the next pointer of that terminal block is altered to make it valid, and the channel is reactivated by the LANai or the host. The content of that block may be altered at any time by the host or the LANai before the next pointer is made valid. Multiple reactivations when a channel is already active is harmless.

When the DMA for a block is completed, the DMA engine will set the T bit of that block. It will generate the LANai's "wake" signal if the wake_bit in the control block was set. It will always compute the IP checksum, and store it in csum[0]. This value will not be complemented, but is the 16-bit one's-complement sum of all the data transferred. The only exception to the processing of a non-terminal block is when length = 0, in which case nothing is done and DMA advances to next block.

NOTE: When writing LANai code to wait for the completion of a DMA operation, you should watch for the "T" bit to change in the descriptor rather than waiting for the WAKE_BIT alone. Once the "T" bit is set you are guaranteed that the DMA has completed.

The controller visits the channels in priority order with 0 being the highest and 3 being the lowest. The controller will complete an entire DMA block (not an entire chain) before moving to the highest priority channel. For example, if the controller is working on a chain on channel 1 and channel 0 gets a chain enqueued, the controller will finish the current block on channel 1 before moving back to channel 0.

When the DMA controller advances to the next block, the DMA controller's pointer register for the channel is updated.

3.3 The Doorbell Region (FIFO Queue Region)

An 8 MByte region of the PCI space is dedicated to doorbell operation, a mechanism that allows the host to write anywhere in the region and to have the address and data of the operation be stored into a FIFO queue in the LANai memory.

The structure of a FIFO entry is:

```
struct FIFO_ENTRY {
    uint32 addr;
    uint32 data;
};
```

Addresses written to the addr field always have 0 for the highest 8 bits and lowest 2 bits. It is therefore possible to use an appropriate constant to indicate an empty FIFO entry.

Operation of the FIFO requires that a base address of the FIFO in LANai memory be written into the controller at the following offsets in PCI address space. The addresses written at these offsets are addresses in the LANai address space. The base address of the region must be on a power of 2 boundary that is the same as the size of the region. These are *little-endian* PCI registers.

Offset	Function
0x00800110	FIFO base pointer 0
0x00800114	FIFO base pointer 1
0x00800118	FIFO base pointer 2
0x0080011C	FIFO base pointer 3

Five bits in the control register encode the size and other aspects of the FIFO operation.

Bit	Function
04	enable doorbell limit checking
03	enable multi-queue doorbells
02	queue size select 2
01	queue size select 1
00	queue size select 0

Queue size select

2	1	0	

1	1	1	512KB = 64K entries
1	1	0	256KB = 32K entries
1	0	1	128KB = 16K entries
1	0	0	64KB = 8K entries
0	1	1	32KB = 4K entries
0	1	0	16KB = 2K entries
0	0	1	8KB = 1K entries

0 0 0 4KB = 512 entries

Bit 3 when set to "1" enables 4 separate FIFO queues to be used, and the user must have written all four of the FIFO base pointer registers. When 4 FIFOs are used, the size refers to each queue, not to the size of all four together. When 4 queues are used, the 8 MB doorbell space is divided into 4 regions. The first 0-2 MB go to FIFO_0, 2-4 MB to FIFO_1, 4-6 MB to FIFO_2 and 6-8 MB to FIFO_3.

Bit 4 enables limit checking when set to "1". The last 4 KBytes of the LANai memory is reserved to a set of limit counters.

```
struct FIFO_COUNTER {
    uint8 limit;
    uint8 count;
};
```

There are 2048 of these counter/limit pairs stored in the last 4KB of the LANai memory. If the limit checking bit is set in the control register, then these counters will be used/checked before the controller will write a FIFO_ENTRY to the FIFO in the LANai.

When a particular 4KB page of the 8 MB space is written, the controller will fetch the count/limit pair and compare them. If the count and limit do NOT match, the controller will

- a) write the address and data of the operation to the FIFO
- b) increment the count and write it back to the LANai
- c) set the WAKE_INT bit in the ISR of the LANai

If the count and limit match, then the FIFO will not be written, the count will not change, and the wake_int bit is not modified.

The idea behind the limit counters is to prevent a particular process from filling the FIFO. Each 4KB page in the mapped space has a `limit` and a `count`, so each process can be prevented from writing more than `limit` entries before the LANai has a chance to process them. During normal operation, the LANai should use Byte writes to update the limit entry. The doorbell mechanism uses Byte writes to update the counter entry.

Note that the 8 MB space is really 8MB *minus* 32K, inasmuch as the first 32KB pages of this space are used for other storage. The "8MB" space really goes from 0x00808000 to 0x01000000. The EEPROM space that overlaps the doorbell space is read-only, while the doorbell space is write-only.

4. Myrinet Port Status and Control

4.1 PCI64B and PCI64C interfaces (LANai 9)

For PCI64B and PCI64C (LANai 9) interfaces, the MYRINET register is used to set all of the link control information. See the <http://www.myri.com/vlsi/LANai9.pdf> file for details on the bits in the MYRINET register. Typically, the MYRINET register will be set to

```
MYRINET = TIMEOUT0 | TIMEOUT1 | WINDOW0 | WINDOW1 |
          RX_CRC8_ENABLE_BIT | TX_CRC8_ENABLE_BIT | CRC32_ENABLE_BIT;
```

The SAN port of the LANai 9 operates at either as a SAN-1280 port at 1,280+1,280 Mb/s, or as a SAN-2000 port at 2,000+2,000 Mb/s. The speed is switchable in the M3M-PCI64B, M3M-PMC64B, and M3M-PMC64C interfaces, is fixed at 1,280+1,280 Mb/s in the M2L-PCI64B and M2L-PCI64C interfaces, and is fixed at 2,000+2,000 Mb/s in the M3S-PCI64B, M3S-PCI64C, M3F-PCI64B, and M3F-PCI64C interfaces. The conversion between the LANai 9's SAN link and

- the LAN link of the M2L-PCI64B or M2L-PCI64C,
- the Serial link of the M3S-PCI64B or M3S-PCI64C,
- the Fiber link of the M3F-PCI64B or M3F-PCI64C,

is transparent to BEAT and to illegal control symbols.

M3S and M3F Interfaces: The programmer should be aware of the following **initialization behavior** of the early SerDes-SAN chip (before version 1.2) used in PCI64B and PCI64C interfaces that have Serial (M3S) and Fiber (M3F) ports.

The SerDes-SAN chip is a converter between a SAN link and the data format and protocols of a Serializer-Deserializer (SerDes) chip. On the LANai-9-based Serial and Fiber interfaces, the SerDes-SAN chip connects on the SAN side to the LANai, and on the SerDes side to a SerDes chip whose 2.5 Gb/s serial signals connect either to the HSSDC Serial connector or to the optical component.

During initialization (upon reset, or after a cable has been re-connected), the early SerDes-SAN chips drop the first packet that goes from its SAN input to the SerDes output. Therefore, each M3S or M3F link using early SerDes-SAN chips drops the first message that goes through it (in each direction).

When a network mapper is used, the probe packets will initialize the entire network. Similarly, retries in a reliable communication layer (such as the one provided by GM) will initialize any links that are in an uninitialized state.

This following **error-rate information** is meant to be an additional reason why authors of Myrinet Control Programs should implement a reliable-communication layer.

The M3S and M3F links are a full duplex, 2.5+2.5 GBaud Physical layer. The signals on these serial links are 8b/10b encoded, the same encoding as Fibre Channel, etc, licensed from IBM. Thus, the data rate after 8b/10b encoding is 2+2 Gbits/s (250+250 MBytes/s). Typical errors on the serial links are single-bit errors. However, because of the 8b/10b encoding and the 8b/10b disparity bit, a bit error will generally produce multiple bit errors in 2 or more successive bytes in the packet data. Thus, a bit-error rate (BER) is not a useful measure for the user.

Myricom guarantees an error rate of less than one packet-data error per hour per link on average (corresponding to a BER of $\sim 10^{-13}$) on M3S and M3F links. This error rate is sufficiently low that the performance cost of a reliable-communication layer is quite small, but is sufficiently high that any practical MCP will include this feature. (Typical observed error rates are at least two orders of magnitude smaller than what Myricom guarantees.)

4.2 PCI64A interfaces (LANai 7)

The SAN port of the LANai 7 operates at 1,280+1,280 Mb/s (SAN-1280), and includes the implementation of the BEAT (Myrinet heartbeat) control symbol and the detection of illegal control symbols. The SAN-LAN conversion in the M2L-PCI64 and M2L-PCI64A LAN interfaces is transparent to BEAT and to illegal control symbols, and provides link status information to the LANai status register via a scan path.

A microcontroller on the interface performs a variety of functions for the Myrinet port. The microcontroller provides status information and is controlled as follows through LANai registers.

The TIMEOUT register has only 4 implemented bits. The other bits are ignored on writes and will return zero when

read. Two bits are used to configure the high-availability (HA) features of the network interface, and the other two bits are used to select the timeout interval:

bit 3 - if 0, interface does not require BEAT before sending or receiving data
if 1, interface requires BEAT to start sending or receiving data

bit 2 - if 0, interface ignores illegal symbols
if 1, interface will shut link down if illegal symbols are received

bits 1,0 - set timeout interval as follows

value	timeout
0	1/16 second
1	1/4 second
2	1 second
3	4 seconds

The status of the Myrinet port is indicated by four bits in the ISR. These bits are provided by the microcontroller. They are:

nres_int - set when the LANai has blocked receive for longer than a timeout interval
link2_int - set when the link times out for any other reason: Lanai sending a single packet for too long, transmit blocked for too long, or received packet is too long.
link1_int - set when link state changes from "UP" to "DOWN"
link0_int - set when link state changes from "DOWN" to "UP"

Nres_int and link2_int are set ~1 millisecond before the microcontroller activates the timeout mechanism, giving the LANai some warning that data is about to be lost.

"UP" means that the link is in a valid state. If the BEAT and/or illegal symbol shutdown features are enabled, "UP" also means that BEATs are being received, and/or illegal symbols are not being received, respectively.

When the link state changes from "UP" to "DOWN", the LANai may receive up to three invalid packets. Also, when the link is "DOWN", anything sent by the LANai will be discarded.

[end]