PM3350

ELAN 8X10

8 PORT ETHERNET SWITCH

DATA SHEET

ISSUE 3: APRIL 1998

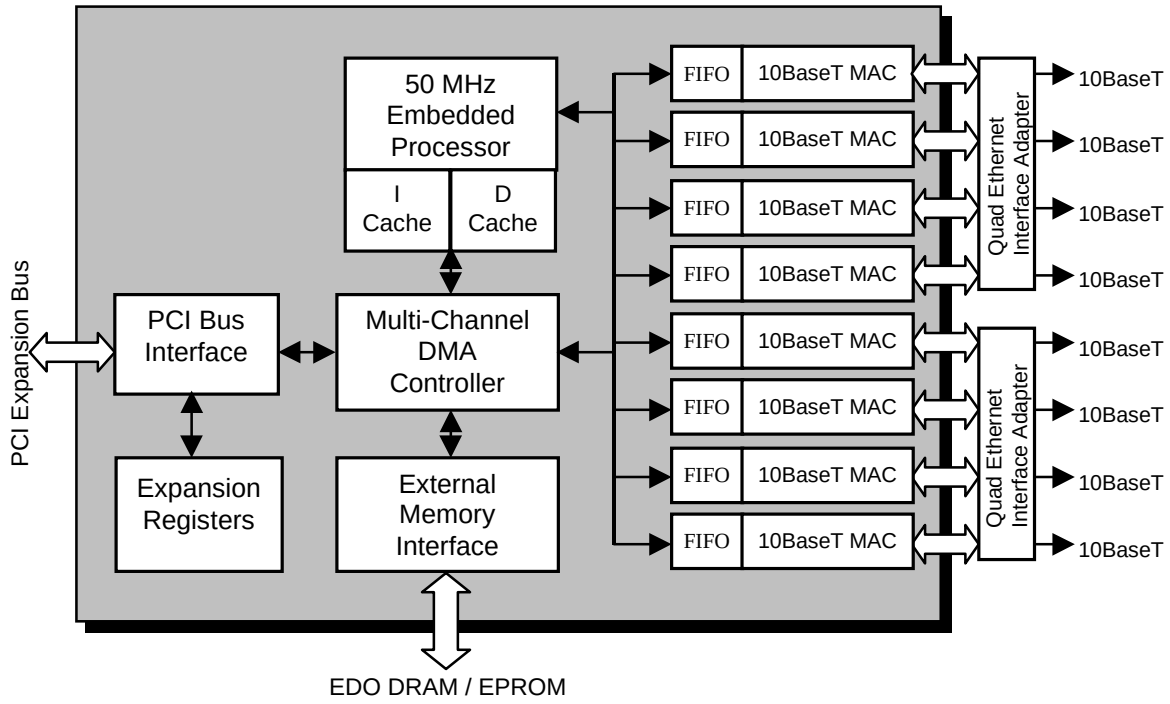
CONTENTS

FEATURES

- Single-chip, 8-port 10BaseT Ethernet switch device for low-cost unmanaged and managed networks.
- On-chip 50 MHz RISC CPU processor core, multi-channel DMA controller, MAC-layer interface logic, FIFOs, PCI-based expansion port and a flexible memory controller.
- CPU supports background applications running on local OS (e.g., SNMP or RMON), and real-time data oriented applications (e.g., packet forwarding and filtering decisions).
- Concurrently switches packets between 8 independent half-duplex ports at the full Ethernet rate of 10 Mbit/s.
- Fully compatible with the PM3351 1-port 10/100 Mbit/s switch device; may be used to create a compact and inexpensive mixed 10/100 Mbit/s switch.
- Store-and-forward operation with full error checking and filtering.
- Filtering and switching at wire rates (up to 14,880 packets per second per port), supporting a mix of Ethernet and IEEE 802.3 protocols.
- Performs all address learning, address table management and aging functions for up to 32,768 MAC addresses (limited by external memory) with an address learning rate of up to 10,000 addresses per second.
- Maximum broadcast/multicast rate of 14,880 packets per second per port with configurable broadcast storm rate limiting.
- Low-latency operation in both unicast and broadcast modes.
- Implements the Link Partition function to isolate malfunctioning segments or hosts.
- IEEE 802.1d compliant spanning-tree transparent bridging supported on-chip, with configurable aging time and packet lifetime control.
- On-chip user-enabled backpressure flow control with configurable per-port buffer thresholds and limits.
- Expandable to 64 ports without loss in throughput using multiple PM3350 devices via an on-chip 1 Gbit/s expansion port.

- Expansion port supports a peak system bandwidth of 1 Gbit/s, and is compatible with industry-standard PCI bus (version 2.1).
- Interfaces directly to industry-standard 10BaseT Ethernet Medium Access Unit devices (LXT944 or similar) with no glue logic.
- Configuration, management, MIB statistics and diagnostics available in-band or out-of-band.
- Maintains and collects per-port and per-host statistics at wire rates, allowing a network switch comprised of PM3351 and PM3350 chips to implement RMON statistics (EtherStats and HostStats) using supplied on-chip firmware.
- Fully static CMOS operation at 50 MHz clock rates.
- 3.3 Volt core, 5 Volt compatible I/O
- 256 pin SBGA package.

BLOCK DIAGRAM



DESCRIPTION

The PM3350 is a low-cost, highly integrated, stand-alone, single-chip switching device for Ethernet/IEEE 802.3 switching and bridging applications. The device supports all processing required for switching Ethernet/IEEE 802.3 packets between eight independent half-duplex 10 Mbit/s ports. In addition, a switch built around the ELAN 8x10 can be expanded simply by connecting up to 7 additional devices to the on-chip 1 Gbit/s expansion port. Switch configuration and management can be performed either remotely (in-band), via the on-chip SNMP MIB, agent and integrated TCP/UDP/IP stack, or from a local CPU interfaced to the expansion port. The ELAN 8x10 also collects per-port and per-host RMON statistics at wire rates on all ports. The ELAN 8x10 chip contains all the required elements of a high-performance Ethernet switch: MAC-layer interfaces, buffer FIFOs, a high-speed DMA engine for fast packet transfers, a local memory interface for up to 16 MB of external buffer memory, a compatible PCI bus master and slave unit for modular expansion, and a switch processing unit that implements the switching and bridging functions. The only additional components required to create a complete 8-port switch are Ethernet Medium Access Unit (MAU) devices, line transformers, a bank of external memory and a system clock.

The ELAN 8x10 device is implemented in high-density CMOS technology for low cost and high performance. It is available in a 256-pin SBGA, and is ideally suited for compact, low-cost desktop, workgroup and departmental Ethernet switching applications.

DEVICE DATA

Introduction

The PM3350 ELAN 8x10 offers a complete system-level solution, integrating all required elements (except packet-buffer/address-table memory and transceiver logic) in a single high-density VLSI chip. It is a true single-chip managed switch; all the required functions, including address learning/aging, management, RMON-level statistics collection, spanning tree support and self-configuration, are performed by the ELAN 8x10 without need for external CPUs or logic. In addition, the functions required for expandability are also integrated into the device.

The ELAN 8x10 is built around a RISC CPU based Switch Processor core, coupled with a multi-channel DMA controller, MAC-layer interface logic, FIFOs, a PCI-based expansion port and a flexible memory controller.

Switch Processor

An on-chip Switch Processor is primarily responsible for performing the Ethernet / IEEE 802.3 packet switching functions, and can switch packets arriving simultaneously from the eight 10BaseT ports and the expansion port at full wire rates using address tables that it creates and maintains in external local memory. Store-and-forward switching is performed, allowing the Switch Processor to detect CRC, length and alignment errors and reject bad packets. The Switch Processor also supports IEEE 802.3 group/functional address handling. Address aging, topology change updates, and statistics collection are performed by the Switch Processor as well.

The Switch Processor unit allows the device to support high-level capabilities. In addition, it implements the full IEEE 802.1d spanning-tree transparent bridging protocol, which allows the ELAN 8x10 to act as a eight-port expandable learning bridge, performing learning, filtering and redirection at full speed. Finally, the Switch Processor can host an SNMP agent for powerful remote switch configuration and diagnostic capabilities, allowing systems built around the ELAN 8x10 to be managed in-band using standard management platforms. When additional switch devices are connected to the ELAN 8x10 expansion port, the Switch Processors in all ELAN 8x10s intercommunicate to transparently support a distributed SNMP MIB.

In host-based applications, the host CPU may bypass the on-chip SNMP agent to communicate directly with the Switch Processor for configuration, control and monitoring purposes.

Multichannel DMA Processor

The on-chip DMA Coprocessor contains eleven independent and concurrently operating channels, one for each of the eight 10BaseT ports and three dedicated to the

expansion port. The DMA Coprocessor operates under the control of the Switch Processor unit to transfer packets and data at high speed between the 10BaseT ports, the local memory and the expansion port. It also computes 32-bit IEEE Frame Check Sequence (FCS) CRC remainders over the transferred data, allowing the Switch Processor to filter packets with errors and generate CRCs for transmitted packets as required.

Ethernet/IEEE 802.3 MAC Interfaces

Eight independent Ethernet/IEEE 802.3 MAC-layer interfaces are built into the ELAN 8x10 chip. These interfaces connect directly to external 10BaseT Medium Access Unit (MAU) devices via the industry-standard 7-wire serial interface, and perform most of the MAC-layer processing tasks required for CSMA/CD networks. In addition, each MAC interface contains a 32-byte FIFO buffer that enhances throughput and minimizes latency issues.

The signaling protocol used on the 7-wire serial interfaces supported by the ELAN 8x10 chip follows that required by the Advanced Micro Devices, Inc (AMD) Am7990 LANCE device.

Expansion Port

A 32-bit parity-checked expansion port is provided to allow systems using the ELAN 8x10 to be expanded transparently from 8 to 64 ports. The expansion port supports a peak throughput of 1 Gbit/s, and requires only a single external PAL or similar device (to serve as a bus arbiter). Packets received on an ELAN 8x10 MAC port that are destined for an external ELAN 8x10 are transferred over the expansion bus prior to transmission on the designated destination port. Broadcast and multicast packets are handled using a two-level replication scheme, in which the broadcast/multicast packet is first transferred to all of the external ELAN 8x10s, after which it is transmitted out the required destination ports without any further use of the expansion bus. In addition, ELAN 8x10s interconnected via the expansion port exchange information to maintain the distributed MIB.

Local Memory Controller

An external local memory is used for holding configuration information, MAC address tables and statistics tables, node management data, packet buffers, and host communication queues (if a local host CPU is present). The ELAN 8x10 integrates a memory controller that is capable of addressing and directly driving up to 16 Mbytes of external memory, divided into four banks of 4 Mbytes each, with decoded selects for each bank. Independent, software programmable access times may be set for each bank, allowing a glueless interface to a mix of EDO DRAM, EEPROM, EPROM and ROM in the same system. An external memory timing generator may also be used if desired. The memory controller accepts simultaneous requests from the Switch

Processor, the DMA processor, and the expansion port, and efficiently partitions the 100 Mbytes/s peak memory port bandwidth among them.

The ELAN 8x10 is capable of auto-configuring after power-up via an 8-bit EPROM or EEPROM connected to the memory port. Parameters (such as the MAC address, IP address, configuration options, etc.) may be placed in this EPROM or EEPROM, and will be loaded automatically by the ELAN 8x10.

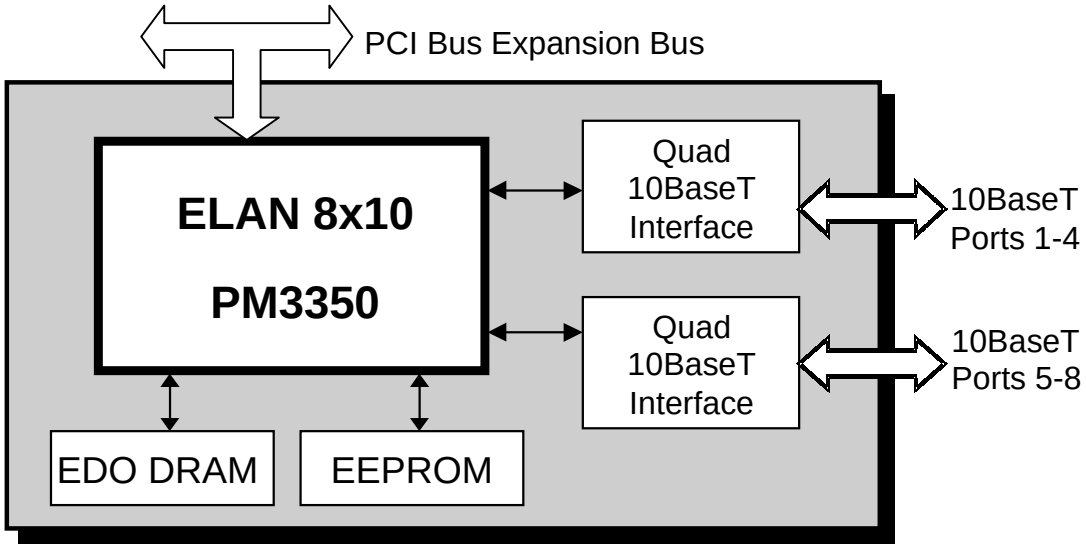
Clocking and Test

The ELAN 8x10 is implemented in fully-static CMOS technology, and is intended to operate at a device clock frequency of 50 MHz (40 MHz for the expansion port bus). The Switch Processor performs a comprehensive power on self test (POST), and can report failure conditions and device status, if necessary, via an 8-bit LED interface register connected to the local memory port.

TYPICAL SYSTEM APPLICATIONS

Simple 8-Port Switch

The ELAN 8x10 chip can act as a stand-alone managed or unmanaged switching device in low-cost, compact switch applications. Such a switch can be created from ELAN 8x10 chips (one for every 8 ports), a bank of two 256k x 16-bit 60 nsec EDO chips per ELAN 8x10 chip, a 32k x 8-bit EPROM or EEPROM for initialization and configuration information, two LXT944 or similar quad Ethernet Interface Adapters, and suitable passive components (filters, transformers, crystal oscillators, etc.) A block diagram of a typical 8-port 10BaseT stackable switch is given below. The PCI expansion bus allows multiple ELAN 8x10 switch assemblies to be easily stacked up to a maximum of 64 10Mbit/s ports.



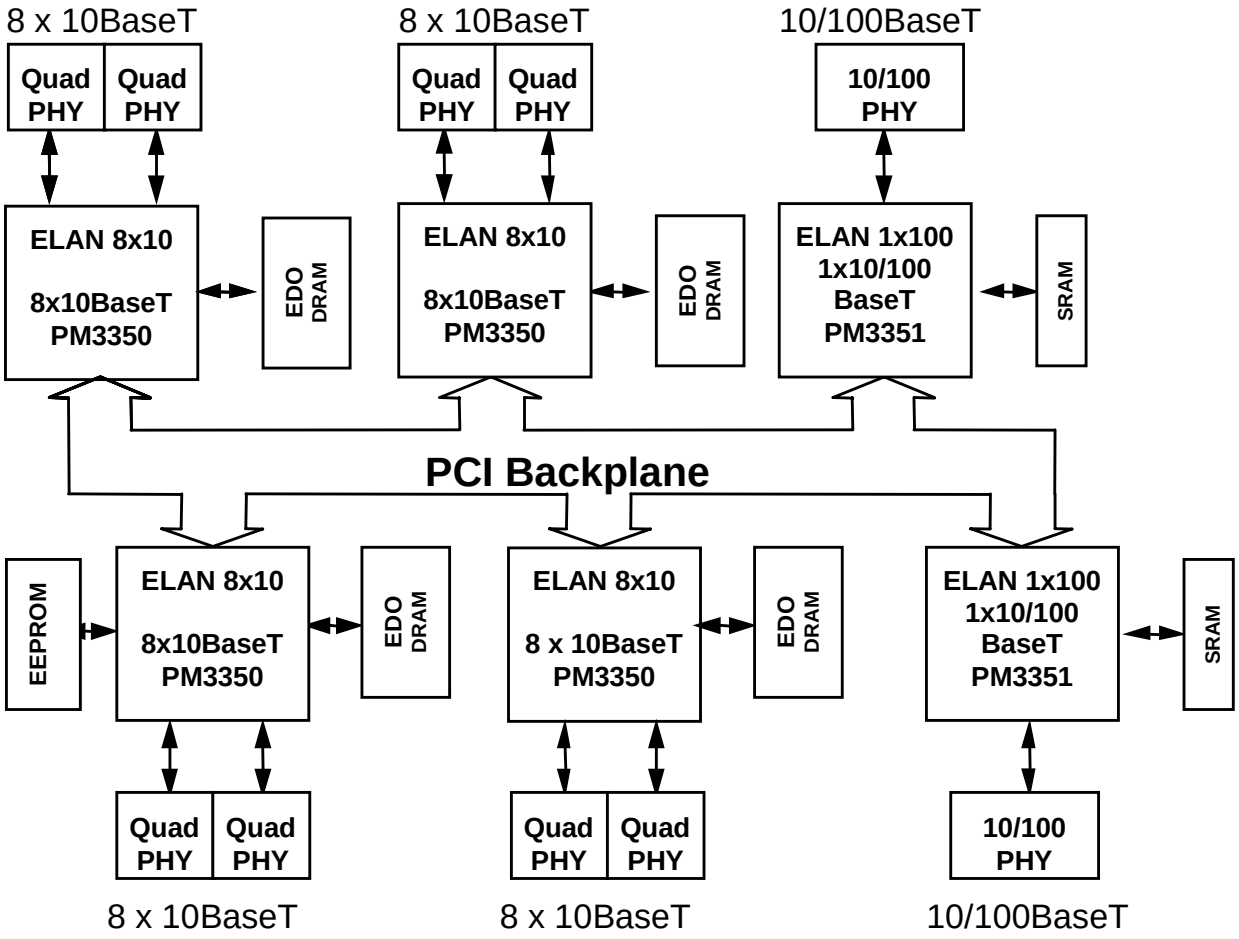
Managed 8-Port Ethernet Switch

The above system will operate as a fully managed system simply by replacing the 32k x 8-bit EEPROM with a 256k x 8-bit EPROM or EEPROM containing the SNMP agent, support firmware, spanning tree, and TCP/IP stack.

Low-cost 10/100 Mbit/s Switch

The ELAN 8x10 chip can be part of a 10/100 switch using the ELAN 1x100 10/100 Mbit/s switch chip. Together they can form a workgroup or desktop switch supporting a high-speed server or backbone port in low-cost, compact Ethernet switching hub applications. Such a switch can be created from one or two PM3351 devices (one for

every 100 Mbit/s port required), 1 to 7 PM3350 chips (one for every eight 10 Mbit/s ports required), a bank of memory per device (60 nsec EDO DRAM for each PM3350, 15 ns SRAM for each PM3351) for holding packet buffers and switching tables, a single 32k x 8-bit EEPROM device for configuration information, two LXT944 10BaseT interface adapters per PM3350 chip, one LXT970 100 Mbit/s PHY device per PM3351, and suitable passive components (filters, transformers, crystal oscillators, etc.) A block diagram of a typical 32-port 10BaseT stackable switching hub with two 100 Mbit/s server/backbone ports is given in the following diagram. The PCI expansion bus is used seamlessly to stack the ELAN 8x10s and ELAN 1x100s.



Expandable 10/100 Mbit/s Ethernet Switch

PRIMARY FEATURES AND BENEFITS

Wire-speed Packet switching

Each ELAN 8x10 chip offers 8 separate Ethernet ports all capable of accepting and processing packets simultaneously at wire rates (ranging from 14,880 packets/sec per port with a packet length of 64 bytes to 813 packets/sec per port with 1518 byte packets), immediately multiplying the aggregate bandwidth of the target network by a factor of eight. In addition, the low cost and compact size permitted by the single-chip ELAN 8x10 solution makes micro-segmentation highly feasible, allowing even more improvement in bandwidth.

Combined Input- and Output-buffered switch

The ELAN 8x10 implements a hybrid input-buffered/output-queued switching algorithm which minimizes packet loss, allows packet buffers to be allocated on a demand basis, and permits limits to be established to prevent any one port from consuming all system buffer memory. Buffer limits are configured on a per-port basis. Packet storage is allocated within the external memory by the ELAN 8x10 from a central pool using an on-demand method, employing linked lists of small, fixed-length buffers to hold variable sized packets in order to maximize memory utilization.

Modular design

Multiple ELAN 8x10 chips interconnect seamlessly, allowing transparent expansion of switching hubs from 8 to 64 ports without significant redesign. The 1 Gbit/s expansion port bandwidth ensures that network capacity grows linearly as more ELAN 8x10 chips are added (up to the maximum number of supported ports).

Advanced switching features

The ELAN 8x10 directly implements backpressure flow control (as a user-selectable option), with configurable thresholds and limits, to minimize packet loss in heavily loaded networks. Backpressure is performed by jamming, or colliding with, incoming packets on an individual port when the port has run out of buffer space; all other ports continue to run normally, with no head-of-line blocking effects. The ELAN 8x10 also implements per-packet lifetime control to ensure that transmit queues are flushed properly in the event of bottlenecks at the output ports. Address aging is handled on-chip, as is purging of the address table in the event of a network reconfiguration. Broadcast storm rate limiting is implemented (with configurable rate limits) to reduce the effects of high broadcast rates on the traffic flowing through the switch.

Spanning tree bridging capabilities

The ELAN 8x10 optionally implements the 802.1d spanning tree protocol, allowing it to interoperate with IEEE-standard transparent bridges. The spanning tree protocol is supported by the on-chip Switch Processor unit, and does not require an external CPU for implementation.

Management and monitoring support

The ELAN 8x10 maintains full RMON port and host statistics for all ports and all learned MAC addresses at wire rates. These statistics may be retrieved either in-band (via SNMP agent) or out-of-band (via the expansion bus).

The ELAN 8x10 implements an optional on-chip SNMP agent on top of an integral UDP/IP protocol stack, supporting SNMP and the RFC1493 bridge MIB. When multiple ELAN 8x10 chips are present in a system, they may be configured to intercommunicate and create a distributed MIB in a transparent manner. Alternatively, the system vendor may elect to disable the SNMP agent and access the ELAN 8x10 statistics and configuration variables directly from the expansion port.

Status codes may be displayed on a set of LEDs (Light Emitting Diodes) during self-test and operation at the system implementer's discretion. These status codes are output to a register mapped into the ELAN 8x10 memory data bus at a specific address location. Device failure during self-test, or specific operating conditions, may be displayed using front-panel LEDs connected to the register.

Autoconfiguration via Local PROM/EEPROM

The ELAN 8x10 will automatically self-configure upon power-up using user-defined parameters supplied in an external EPROM or EEPROM. The EPROM/ EEPROM may be connected to the memory bus and mapped to any address range; the ELAN 8x10 will automatically locate the EPROM or EEPROM and load the configuration parameters from it. The ELAN 8x10 also contains hardware that enables it to write to standard 3.3-volt EEPROM devices, thus permitting configuration information to be changed dynamically.

PIN DIAGRAM

The ELAN 8x10 is packaged in a 256-pin cavity-down SBGA, with 188 signal pins and associated VDD (supply) and VSS (ground) pins.

256-Pin SBGA Pin Diagram

	20	19	18	17	16	15	14	13	12	11
A	VSS	VSS	VSS	MDATA_31	MDATA_27	MDATA_24	MDATA_20	VSS	MDATA_14	MDATA_10
B	VSS	VDD	VDD	MRAS_	MDATA_28	MDATA_25	MDATA_21	MDATA_17	MDATA_15	MDATA_11
C	VSS	VDD	VDD	MWEN_1_	MDATA_30	MDATA_26	MDATA_23	MDATA_19	MDATA_16	MDATA_12
D	MRDY_	MRD_	MWEN_2_	VDD	MWEN_0_	MDATA_29	VDD	MDATA_22	MDATA_18	MDATA_13
E	MCS_3_	MCS_2_	MCS_0_	MWEN_3_						
F	MADDR_2	MADDR_1	MADDR_0	MCS_1_						
G	MADDR_6	MADDR_5	MADDR_3	VDD						
H	MADDR_10	MADDR_8	MADDR_7	MADDR_4						
J	VSS	MADDR_12	MADDR_11	MADDR_9						
K	VSS	MADDR_13	MADDR_14	VDD						
L	MADDR_15	MADDR_16	MADDR_17	MADDR_18						
M	MADDR_19	COL_7	CD_7	RXD_7						
N	VSS	RCLK_7	TEN_7	COL_6						
P	TXD_7	TCLK_7	CD_6	VDD						
R	RCLK_6	RXD_6	TEN_6	COL_5						
T	TXD_6	TCLK_6	CD_5	TEN_5						
U	RCLK_5	RXD_5	TXD_5	VDD	COL_4	RXD_4	VDD	RXD_3	CD_2	VDD
V	VSS	VDD	VDD	TCLK_5	LBK	TCLK_4	RCLK_3	TCLK_3	RXD_2	TCLK_2
W	VSS	VDD	VDD	CD_4	TEN_4	COL_3	TEN_3	COL_2	TEN_2	TXD_2
Y	VSS	VSS	VSS	RCLK_4	TXD_4	CD_3	TXD_3	RCLK_2	VSS	VSS
	20	19	18	17	16	15	14	13	12	11

256 BGA
BOTTOM VIEW

10	9	8	7	6	5	4	3	2	1	
VSS	VSS	MDATA_5	MDATA_1	DBG_2	PC_3	PC_7	VSS	VSS	VSS	A
MDATA_8	MDATA_7	MDATA_3	MDATA_0	DBG_1	PC_4	PC_8	VDD	VDD	VSS	B
MDATA_9	MDATA_6	MDATA_2	MINTR_	DBG_0	PC_6	PC_10	VDD	VDD	VSS	C
VDD	MDATA_4	MGWE_	VDD	PC_5	PC_9	VDD	N/C	N/C	ERST_	D
						N/C	TST_	RST_	GNT_	E
						INT_	REQ_	AD_31	AD_30	F
						VDD	AD_29	AD_27	AD_26	G
						AD_28	AD_25	CBE_3_	VSS	H
						AD_24	VDD5	IDEL_	AD_23	J
						AD_22	AD_21	AD_20	AD_19	K
						VDD	AD_18	AD_17	VSS	L
						IRDY_	CBE_2_	AD_16	VSS	M
						SERR_	DEVSEL_	TRDY_	FRAME_	N
						VDD	PAR	PERR_	STOP_	P
						AD_11	AD_14	AD_15	CBE_1_	R
						AD_7	AD_10	AD_12	AD_13	T
COL_1	TXD_1	RCLK_0	VDD	AD_2	AD_6	VDD	PCI_CLK	AD_8	AD_9	U
MCLK	RXD_1	TCLK_1	RXD_0	TCLK_0	AD_3	CBE_0_	VDD	VDD	VSS	V
SYSCLK	RCLK_1	TEN_1	CD_0	TXD_0	AD_1	AD_5	VDD	VDD	VSS	W
CLK20	CD_1	VSS	COL_0	TEN_0	AD_0	AD_4	VSS	VSS	VSS	Y
10	9	8	7	6	5	4	3	2	1	

256 BGA
BOTTOM VIEW

PIN DESCRIPTION

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
D1	ERST_	R3	AD_14	W12	TEN_2
E3	TST_	T1	AD_13	V12	RXD_2
F4	INT_	T2	AD_12	Y13	RCLK_2
E2	RST_	R4	AD_11	U12	CD_2
E1	GNT_	T3	AD_10	W13	COL_2
F3	REQ_	U1	AD_9	V13	TCLK_3
F2	AD_31	U2	AD_8	Y14	TXD_3
F1	AD_30	T4	AD_7	W14	TEN_3
G3	AD_29	U3	PCI_CLK	U13	RXD_3
H4	AD_28	V4	CBE_0_	V14	RCLK_3
G2	AD_27	U5	AD_6	Y15	CD_3
G1	AD_26	W4	AD_5	W15	COL_3
H3	AD_25	Y4	AD_4	V15	TCLK_4
J4	AD_24	V5	AD_3	Y16	TXD_4
H2	CBE_3_	U6	AD_2	W16	TEN_4
J3	VDD5	W5	AD_1	U15	RXD_4
J2	IDSEL_	Y5	AD_0	V16	LBK
J1	AD_23	V6	TCLK_0	Y17	RCLK_4
K4	AD_22	W6	TXD_0	W17	CD_4
K3	AD_21	Y6	TEN_0	U16	COL_4
K2	AD_20	V7	RXD_0	V17	TCLK_5
K1	AD_19	U8	RCLK_0	U18	TXD_5
L3	AD_18	W7	CD_0	T17	TEN_5
L2	AD_17	Y7	COL_0	U19	RXD_5
M2	AD_16	V8	TCLK_1	U20	RCLK_5
M3	CBE_2_	U9	TXD_1	T18	CD_5
N1	FRAME_	W8	TEN_1	R17	COL_5
M4	IRDY_	V9	RXD_1	T19	TCLK_6
N2	TRDY_	W9	RCLK_1	T20	TXD_6
N3	DEVSEL_	Y9	CD_1	R18	TEN_6
P1	STOP_	U10	COL_1	R19	RXD_6
P2	PERR_	V10	MCLK	R20	RCLK_6
N4	SERR_	W10	SYSCLK	P18	CD_6
P3	PAR	Y10	CLK20	N17	COL_6
R1	CBE_1_	V11	TCLK_2	P19	TCLK_7
R2	AD_15	W11	TXD_2	P20	TXD_7

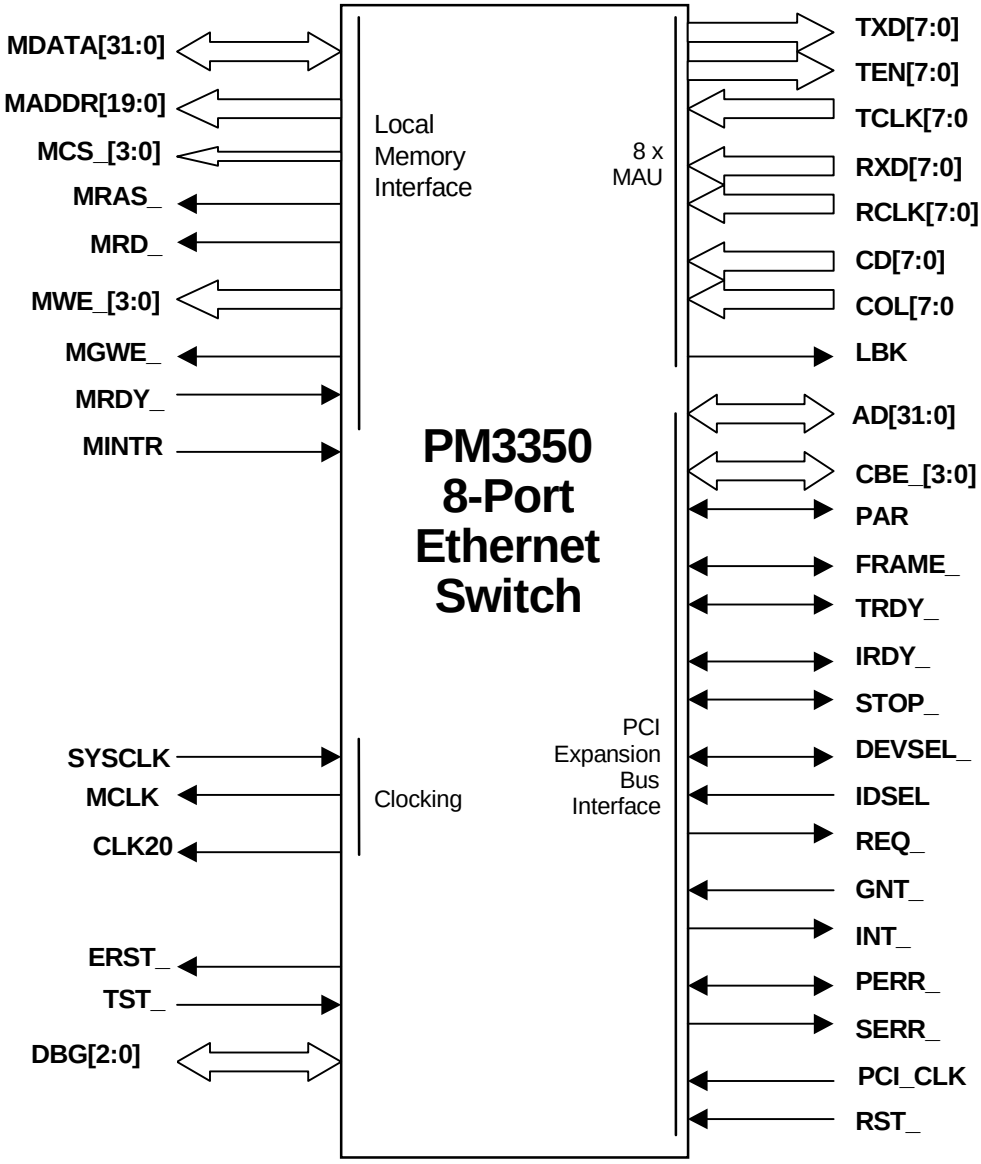
PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
N18	TEN_7	F17	MCS_1_	D11	MDATA_13
M17	RXD_7	E18	MCS_0_	C11	MDATA_12
N19	RCLK_7	D20	MRDY_	B11	MDATA_11
M18	CD_7	D19	MRD_	A11	MDATA_10
M19	COL_7	E17	MWEN_3_	C10	MDATA_9
M20	MADDR_19	D18	MWEN_2_	B10	MDATA_8
L17	MADDR_18	C17	MWEN_1_	B9	MDATA_7
L18	MADDR_17	D16	MWEN_0_	C9	MDATA_6
L19	MADDR_16	B17	MRAS_	A8	MDATA_5
L20	MADDR_15	A17	MDATA_31	D9	MDATA_4
K18	MADDR_14	C16	MDATA_30	B8	MDATA_3
K19	MADDR_13	D15	MDATA_29	C8	MDATA_2
J19	MADDR_12	B16	MDATA_28	A7	MDATA_1
J18	MADDR_11	A16	MDATA_27	B7	MDATA_0
H20	MADDR_10	C15	MDATA_26	D8	MGWE_
J17	MADDR_9	B15	MDATA_25	C7	MINTR_
H19	MADDR_8	A15	MDATA_24	A6	DBG_2
H18	MADDR_7	C14	MDATA_23	B6	DBG_1
G20	MADDR_6	D13	MDATA_22	C6	DBG_0
G19	MADDR_5	B14	MDATA_21	A5	PC_3
H17	MADDR_4	A14	MDATA_20	B5	PC_4
G18	MADDR_3	C13	MDATA_19	D6	PC_5
F20	MADDR_2	D12	MDATA_18	C5	PC_6
F19	MADDR_1	B13	MDATA_17	A4	PC_7
F18	MADDR_0	C12	MDATA_16	B4	PC_8
E20	MCS_3_	B12	MDATA_15	D5	PC_9
E19	MCS_2_	A12	MDATA_14	C4	PC_10
D3	N/C	E4	N/C	D2	N/C

Pin	Signal	Pin	Signal
B2	VDD	A1	VSS
B3		A2	
B18		A3	
B19		A9	
C2		A10	
C3		A13	
C18		A18	
C19		A19	
D4		A20	
D7		B1	
D10		B20	
D14		C1	
D17		C20	
G4		H1	
G17		J20	
K17		K20	
L4		L1	
P4		M1	
P17		N20	
U4		V1	
U7		V20	
U11		W1	
U14		W20	
U17		Y1	
V2		Y2	
V3		Y3	
V18		Y8	
V19		Y11	
W2		Y12	
W3		Y18	
W18		Y19	
W19		Y20	

All VDD and VSS lines **must** be connected to supply and ground respectively. Standard decoupling practices should be followed for proper device operation.

Functional Grouping

The following diagram shows the functional grouping of the ELAN 8x10 signal pins.



PCI Expansion Bus Interface

Signal Name	Size	Type	Description
AD[31:0]	32	I/O	Multiplexed PCI address/data bus, used by an external bus master (e.g., a PCI host) or the ELAN 8x10 to transfer addresses or data.
CBE_[3:0]	4	I/O	Command/Byte-Enable lines. These lines supply a command (during PCI address phases) or byte enables (during data phases) for each bus transaction.
PAR	1	I/O	Address/data/command parity, supplies the even parity computed over the AD[31:0] and CBE_[3:0] lines during valid data phases; it is sampled (when the ELAN 8x10 is acting as a target) or driven (when the ELAN 8x10 acts as an initiator) one clock edge after the respective data phase.
FRAME_	1	I/O	Bus transaction delimiter (framing signal); a HIGH-to-LOW transition on this signal indicates that a new transaction is beginning (with an address phase); a LOW-to-HIGH transition indicates that the next valid data phase will end the currently ongoing transaction.
IRDY_	1	I/O	Transaction Initiator (master) ready, used by the transaction initiator or bus master to indicate that it is ready for a data transfer. A valid data phase ends with data transfer when both IRDY_ and TRDY_ are sampled asserted on the same clock edge.
TRDY_	1	I/O	Transaction Target ready, used by the transaction target or bus slave to indicate that it is ready for a data transfer. A valid data phase ends with data transfer when both IRDY_ and TRDY_ are sampled asserted on the same clock edge.
STOP_	1	I/O	Transaction termination request, driven by the current target or slave to abort, disconnect or retry the current transfer.
DEVSEL_	1	I/O	Device select acknowledge: driven by a target to indicate to the initiator that the address placed on the AD[31:0] lines (together with the command on the CBE_[3:0] lines) has been decoded and accepted as a valid reference to the target's address space. Once asserted, it is held asserted until FRAME_ is de-asserted; otherwise, it indicates (in conjunction with STOP_ and TRDY_) a target-abort.
IDSEL	1	I	Device identification (slot) select. Assertion of IDSEL signals the ELAN 8x10 that it is being selected for a configuration space access.
REQ_	1	O	Bus request (to bus arbiter), asserted by the ELAN 8x10 to request control of the PCI bus.
GNT_	1	I	Bus grant (from bus arbiter); this indicates to the ELAN 8x10 that it has been granted control of the PCI bus, and may begin driving the address/data and control lines after the current transaction has ended (indicated by FRAME_, IRDY_ and TRDY_ all de-asserted simultaneously).
INT_	1	OD	Open Drain Interrupt request. This pin signals an interrupt request to an external PCI host system. The INT_ pin should be tied to the INTA* line on the PCI bus.
PERR_	1	I/O	Bus parity error signal, asserted by the ELAN 8x10 as a bus slave, or sampled by the ELAN 8x10 as a bus master, to indicate a parity error on the AD[31:0] and CBE_[3:0] lines.
SERR_	1	OD	Open Drain System error, used by the ELAN 8x10 to indicate a system error, or a parity error on the AD[31:0] and CBE_[3:0] lines during an address phase.
PCI_CLK	1	I	PCI bus clock; supplies the PCI bus clock signal to the ELAN 8x10.

RST_	1	I	PCI bus reset (system reset). Performs a hardware reset of the ELAN 8x10 and associated peripherals when asserted. The RST_ input uses a Schmitt trigger to accommodate slow rise and fall times, allowing a simple RC network to be used to provide power-on reset capability.

MAU Interface Pins

Signal Name	Size	Type	Description
TXD[7:0]	8	O	Transmit data outputs (to 8 MAUs). The TXD lines are used to carry outgoing data bytes to the Medium Access Units (MAUs). Each bit of the bus is connected to the serial transmit data input of a separate MAU device. The TXD[7:0] lines are synchronous to the TCLK[7:0] lines (TXD[0] to TCLK[0], and so on). Data are driven on to the TXD[7:0] lines after the rising edge of the corresponding TCLK[7:0] input, and may be sampled on the next rising edge of the latter.
TEN[7:0]	8	O	Transmit enables. The data carried on the TXD[7:0] lines is only valid when the TEN[7:0] lines are active. This also indicates to the MAU devices that the ELAN 8x10 is acquiring the medium. Each of the TEN[7:0] lines should be connected to a separate MAU device. The TEN[7:0] outputs are asserted or de-asserted on the rising edge of the corresponding TCLK[7:0] input, and may be sampled on the next rising edge of the latter.
TCLK[7:0]	8	I	Transmit clocks; these inputs provide the synchronization references for the TXD[7:0] and TEN[7:0] lines. TCLK[7:0] should be driven with a 10 MHz transmit data clock reference by the external MAU devices (each MAU device should drive a separate line of the TCLK[7:0] bus). Each of the TCLK[7:0] lines may be driven completely asynchronously to all the others. The rising edges of the TCLK[7:0] signals are used as timing references. The TCLK[7:0] inputs are Schmitt-triggered for improved resistance to slow rise and fall times.
RXD[7:0]	8	I	Receive data inputs (from 8 MAUs). The RXD lines transfer incoming serial received data from the external MAU devices to the ELAN 8x10. Each bit of the bus is connected to the serial receive data output of a separate MAU device. The RXD[7:0] lines are sampled synchronously by the ELAN 8x10 at the rising edges of the clocks supplied on the RCLK[7:0] inputs (RXD[0] to RCLK[0], and so on).
RCLK[7:0]	8	I	Receive clocks; should be driven by the external MAU devices with the receive clock references recovered from the incoming serial receive data. The RCLK[7:0] inputs need not be driven with a continuous clock reference; however, they must be running whenever the CD[7:0] inputs are asserted, and must continue running for at least five clock cycles after the CD[7:0] inputs transition LOW in order to permit the internal MAC logic to function properly. The RCLK[7:0] inputs are Schmitt-triggered for improved resistance to slow rise and fall times.
CD[7:0]	8	I	Receive carrier detect signals. These carrier detect inputs should be driven with the carrier detect (i.e., data being received from the medium) signals generated by the eight external MAU devices, and are sampled synchronously to the rising edges of the clocks input on the RCLK[7:0] lines and the TCLK[7:0] lines to implement the CSMA/CD algorithm. The data presented on RXD[7:0] are only sampled when the corresponding CD[7:0] lines are asserted HIGH. (If not all of the eight ELAN 8x10 MAC ports are connected to external MAU devices, the CD inputs for to the missing MAU devices should be tied LOW.)

COL[7:0]	8	I	Receive collision detect signals. These inputs pass the collision detect and SQE test signals generated by the external MAU devices to the ELAN 8x10. They are sampled synchronously to the TCLK[7:0] clock references (COL[0] corresponds to TCLK[0], and so on), and should be asserted by the MAU devices to indicate collisions on the medium as well as to signal a successful SQE test after transmit. If not all of the 8 collision signal inputs are connected to external MAU devices, the unused inputs should be tied LOW.
LBK	1	O	MAU loopback mode select (all 8 ports). This pin can be left as a no-connect or wired to the loopback mode input of an attached PHY device. The loopback mode feature on the PM3350 is not operational. Hence, this pin has no implied functionality.

Local Memory Interface

Signal Name	Size	Type	Description
MDATA[31:0]	32	I/O	Memory data bus. MDATA[31:0] carries the data driven to the external local memory by the ELAN 8x10 during local memory writes, and the data sent back to the ELAN 8x10 by the memory devices during local memory reads. In addition, configuration information is latched from the MDATA[31:0] lines during ELAN 8x10 reset and loaded into an internal configuration register; either pullup-pulldown resistors or tri-state buffers (enabled by the RST* input) drive configuration data on to the MDATA[31:0] lines during reset. All MDATA[31:0] pins have internal pullups.
MADDR[19:0]	20	O	Memory address bus; supplies a word-aligned address to the external memory devices (i.e., address bits 22 through 2 of the 24-bit byte address generated by the internal ELAN 8x10 logic), and thus selects a single 32-bit word to be read or written. Up to 4 MB of memory may be directly addressed in each bank using these address lines. In addition, the lower 11 bits of MADDR[19:0] (i.e., MADDR[10:0]) carry a multiplexed row/column address when DRAM accesses are being made, with the row address being presented when MRAS_ is high and the column address being presented when it is low. Multiplexing for 8-, 9-, 10- and 11-bit column addresses is supported. Up to 4 MB of memory may be addressed in each bank when multiplexing is enabled for that bank (i.e., by configuring the bank for DRAM accesses).
MCS_[3:0]	4	O	Memory bank chip selects. The MCS_[3:0] outputs select one of four memory banks; each bank is 4 megabytes in size. They are decoded directly from the most significant 2 bits (bits 23 and 22) of the 24-bit physical byte address generated by the internal ELAN 8x10 logic, and are synchronous to MEMCLK. When driving DRAM memory devices other than 2-CAS DRAMs, the MCS_[3:0] signals function as the Column Address Strobe (CAS) signals to the memories.
MRAS_	1	O	DRAM Row Address Strobe output; supplies the Row Address Strobe (RAS) signal to one or more external DRAM banks. It is asserted to latch the row address supplied on the MADDR lines into the DRAM array, and allow the column address to be output one cycle later. It may be tied directly to the RAS* inputs of standard asynchronous DRAM devices.

MRD_	1	O	Memory read enable. This output signals the external memory banks that a read is being performed and data should be output on the MDATA[31:0] lines from the specified address. The MRD_ output may be tied to the OE* inputs of standard memory devices.
MWE_[3:0]	4	O	Memory write enables, used by the ELAN 8x10 to enable the data presented on individual byte lanes of MDATA[31:0] to be individually written to memory. MWE_[0] corresponds to MDATA[7:0], MWE_[1] corresponds to MDATA[15:8], and so on. When using 2-CAS asynchronous DRAM devices, the MWE_[3:0] outputs should be connected to the memory CAS* inputs; otherwise, the MWE_[3:0] outputs should be connected to the appropriate byte write enables.
MGWE_	1	O	Global memory write enable. This signal is used to signal that a write access is occurring, and should be connected to the WE* inputs of dual CAS asynchronous DRAM devices.
MRDY_	1	I	Memory ready input. If an external memory timing generator is used, it can be connected to the MRDY_ input to force the ELAN 8x10 to insert wait states into memory accesses. If the MRDY_ line is de-asserted, the ELAN 8x10 will hold the MADDR[15:0], MCS_[3:0], MRD_ and MWR_[3:0] lines at their present values (as well as MDATA[31:0] for memory writes). The MRDY* input is only sampled by the ELAN 8x10 when performing an SRAM-type access; it is ignored for all other memory types. If an external memory timing generator is not used, the MRDY_ line should be tied LOW.
MINTR_	1	I	Local interrupt input. The MINTR_ may be used to provide an interrupt input to the ELAN 8x10 in special applications. If the MINTR_ input is not used, it should be tied HIGH.

Clock Inputs and Outputs

Signal Name	Size	Type	Description
SYSCLK	1	I	50 MHz master device clock input, This should be driven by a 50 MHz symmetrical clock source with a duty cycle between 40% and 60%. It is re-timed and driven out on the MEMCLK line, and is also used in the internal device logic.
MCLK	1	O	50 MHz clock output derived from SYSCLK; supplies the re-timed 50 MHz clock (input on the SYSCLK pin) to external devices.
CLK20	1	O	20 MHz clock output (for use by MAU devices). The 50 MHz input clock is internally divided by 2.5 and output as an asymmetrical 20 MHz clock reference on the CLK20 output; this clock may be used as an input to external MAU devices.

Miscellaneous Inputs and Outputs

Signal Name	Size	Type	Description
VDD5	1	I	The 5 volt bias pin must be connected to 5.0 volts for the input and bi-directional pins to be 5 volt tolerant. This pin may be tied to VDD provided the maximum static signal level is below VDD + 0.3V. During power-up, the voltage on the VDD5 pin must be kept equal to or greater then the voltage on all input pins to avoid damage to the device. In addition, the voltage on the VDD5 pin is to be kept greater than or equal to the voltage on the VDD pins.
TST_	1	I	Test select signal used for production testing. It must be tied high for correct operation.
ERST_	1	OD	Open-drain external reset output. The ERST_ pin is driven low by the ELAN 8x10 to reset other components in the system. This output is asserted for a pre-set duration (10 milliseconds) upon the detection of a falling edge on the RST_ input, or when the ELAN 8x10 senses a condition requiring a system hardware reset. It is an open-drain output, and should be pulled up using a 2.2k resistor. The ERST_ output may be tied directly to the RST_ input to implement a debounce function in pushbutton reset applications. (Note that the ERST_ output should be left unconnected in host-based applications where the ELAN 8x10 must not be allowed to reset the host CPU.)
DBG[2:0]	3	I/O	PMC hardware debug pins. These pins are used for hardware debug and should not be used. These pins should be implemented as no-connects.
PC[10:3]	8	O	PMC hardware debug pins. Program counter selected outputs. These pins are used for hardware debug and should not be used. These pins should be implemented as no-connects.

Notes on Pin Description:

1. All inputs and bi-directionals present minimal capacitive loading and operate at TTL logic levels.
2. All digital outputs and bi-directionals have 2 mA D.C. drive capability.
3. Pins MDATA[31:0], MINTR_,RST_, GNT_, TST_ and DBG[2:0] have internal pull-up resistors.

DC CHARACTERISTICS

Absolute Maximum Ratings

Maximum rating are the worst case limits that the device can withstand without sustaining permanent damage. They are not indicative of normal mode operation conditions.

Parameter	Symbol	Value	Units
Supply Voltage	Vdd	-0.3 to +5.0.	Vdc
VDD5 pin voltage with respect to Vss	Vbias	Minimum: VDD – 0.3V Maximum: 5.5V	Vdc
Input Voltage	Vin	VDD5 +0.3	Vdc
Input Current	Iin	+/-10	mAdc
Static Discharge Voltage		±1000	V
Latch-Up Current		±100	mA
Lead Temperature		+220	°C
Storage Temperature	Tst	-45 to +125	°C
Junction Temperature	Tj	+125	°C

Recommended Operating Conditions

Parameter	Symbol	Value			Units
		Min	Nom	Max	
Supply Voltage	Vdd	+3.13	+3.30	+3.47	Vdc
VDD5 pin voltage	Vbias	+4.75	+5.0	+5.25	Vdc
Operating Ambient Temp.	Ta	0		+70	°C

NOTE: the PM3350 has been characterized over the industrial temperature range (Ta = -40°C to +85°C). All DC and AC parametrics met the limits presented in the following tables. In addition the package thermal characteristics of the 256-pin SBGA and power consumption of the device are such that the device can be operated without any forced air (i.e. still air) over the full industrial temperature range.

D.C. Characteristics

DC characteristics are specified over recommended operating conditions (T_A = 0°C to +70°C, V_{DD} = 3.3 V ±5%).

Parameter	Description	TTL I/Os		PCI I/Os		Units
		Min	Max	Min	Max	
Vih	Input High Voltage	2.0	VDD + 0.5	2.0	VDD + 0.5	Vdc
Vil	Input Low Voltage	-0.5	0.8	-0.5	0.8	Vdc
Voh	Output High Voltage (V_{DD} = min, I_{OH} = 2 mA, Note 2)	2.4	Vdd	2.4	Vdd	Vdc
Vol	Output Low Voltage (V_{DD} = min, I_{OL} = -2 mA, Note 2)	0	0.4	0	0.4	Vdc
Iil	Input Low Leakage Current, Note 3	-10	10	-10	10	µA
Iih	Input High Leakage Current, Note 3	-10	10	-10	10	µA
Iilpu	Input Low Current (Pull ups, V_{IL} = GND, Notes 1)	+100	+20			µA
Iihpu	Input High Current (Pull ups, V_{IH} = V_{DD}, Notes 1)	-10	+10			µA
Iddop	Supply Current, VDD = 3.47V, Outputs Unloaded, SYSCLK = 50MHz		450			mA

Notes on D.C. Characteristics:

1. Input pin or bi-directional pin with internal pull-up resistor.
2. Output pin or bi-directional pins. Voh not measured on open drain ouputs
3. Negative currents flow into the device (sinking), positive currents flow out of the device (sourcing).

AC CHARACTERISTICS

AC characteristics are specified over recommended operating conditions (T_A = 0°C to +70°C, V_{DD} = 3.3 V ±5%).

The PM3350 only supports 5V signalling environment

Notes on Input Timing:

1. When a set-up time is specified between an input and a clock, the set-up time is the time in nanoseconds from the 1.4 Volt point of the input to the 1.4 Volt point of the clock.

2. When a hold time is specified between an input and a clock, the hold time is the time in nanoseconds from the 1.4 Volt point of the clock to the 1.4 Volt point of the input.

Notes on Output Timing:

1. Output propagation delay time is the time in nanoseconds from the 1.4 Volt point of the reference signal to the 1.4 Volt point of the output.

2. Maximum and minimum output propagation delays are specified with a 50 pF load on the outputs, unless otherwise noted.

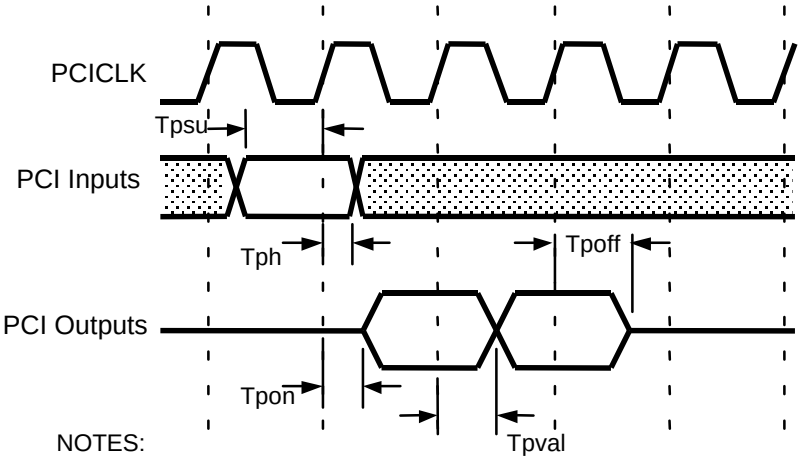
3. Output tri-state delay is the time in nanoseconds from the 1.4 Volt point of the reference signal to ±300mV of the termination voltage on the output. The test load is 50Ω to 1.4V in parallel with 10 pF to GND.

Notes on Typical Values:

AC parameters that are shown as **Typical** in the following tables are tested for functionality under typical conditions. No guarantees are implied for max or min performance.

PCI Bus Interface

Parameter	Description	MIN	TYP	MAX	Units
Tpsu	Setup time of all PCI inputs from PCICLK rising	7			nsec
Tph	Hold time of all PCI inputs from PCICLK rising	0			nsec
Tpon	Min Float to active delay of all outputs from PCICLK rising		2		nsec
Tpoff	Max Active to float delay of all outputs from PCICLK rising		28		nsec
Tpval	Signal valid of all outputs from PCICLK rising	2		13	nsec
Trstoff	RST* active to output float delay		40		nsec

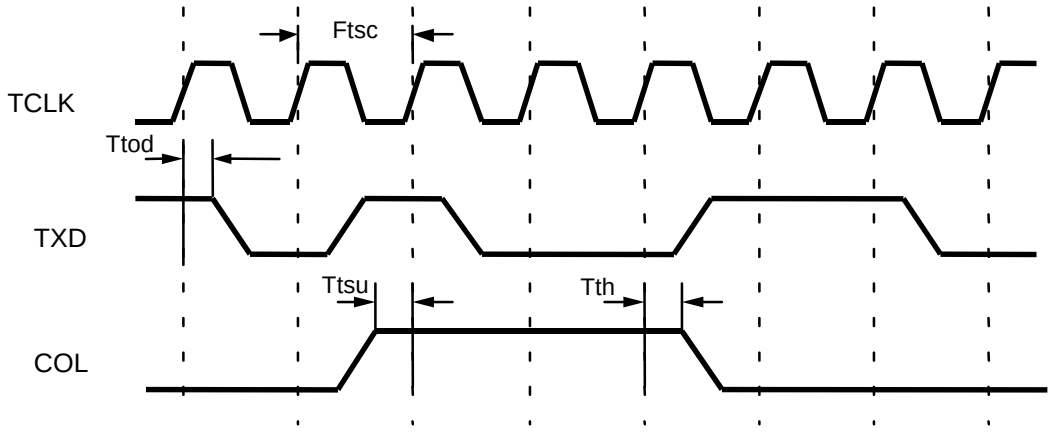


NOTES:

(1) PCI inputs are considered to be those signals that are driven by an external PCI device, while PCI outputs are signals that are driven by the PM3350. Note that many of the PCI signals are treated as outputs in some cycles and inputs in others.

MAC Interface

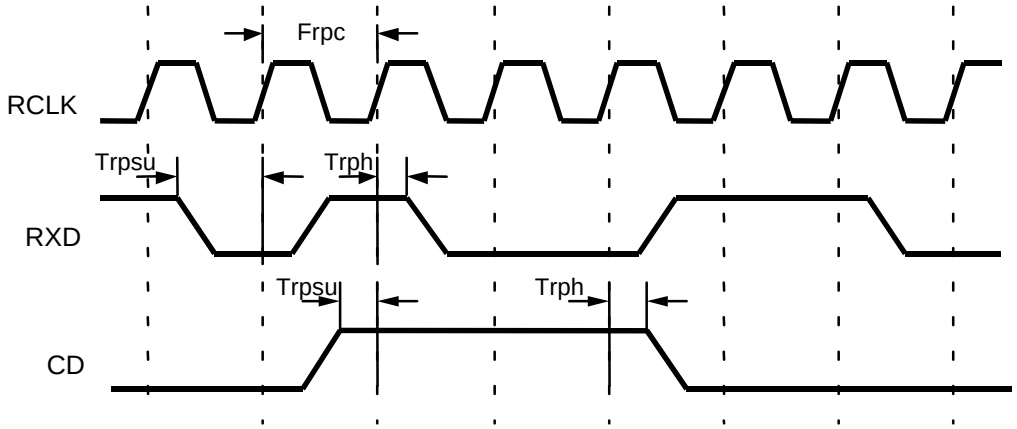
Parameter	Description	MIN	MAX	Units
Ftsc	TCLK frequency	0	10	MHz
	TCLK duty cycle	40	60	Percent
Ttod	TXD delay from TCLK	3	20	nsec
Ttsu	COL setup to TCLK	20		nsec
Tth	COL hold to TCLK	20		nsec



MAC Interface Transmit Signals

NOTES:
(1) Timing waveform is identical for all MAC ports

Parameter	Description	MIN	MAX	Units
Frpc	RCLK frequency	0	10	MHz
	RCLK duty cycle	40	60	Percent
Trpsu	RXD,CD setup to RCLK	20		nsec
Trph	RXD,CD hold to RCLK	20		nsec



MAC Interface Receive Signals

NOTES:
(1) Timing waveform is identical for all MAC ports

Memory Interface

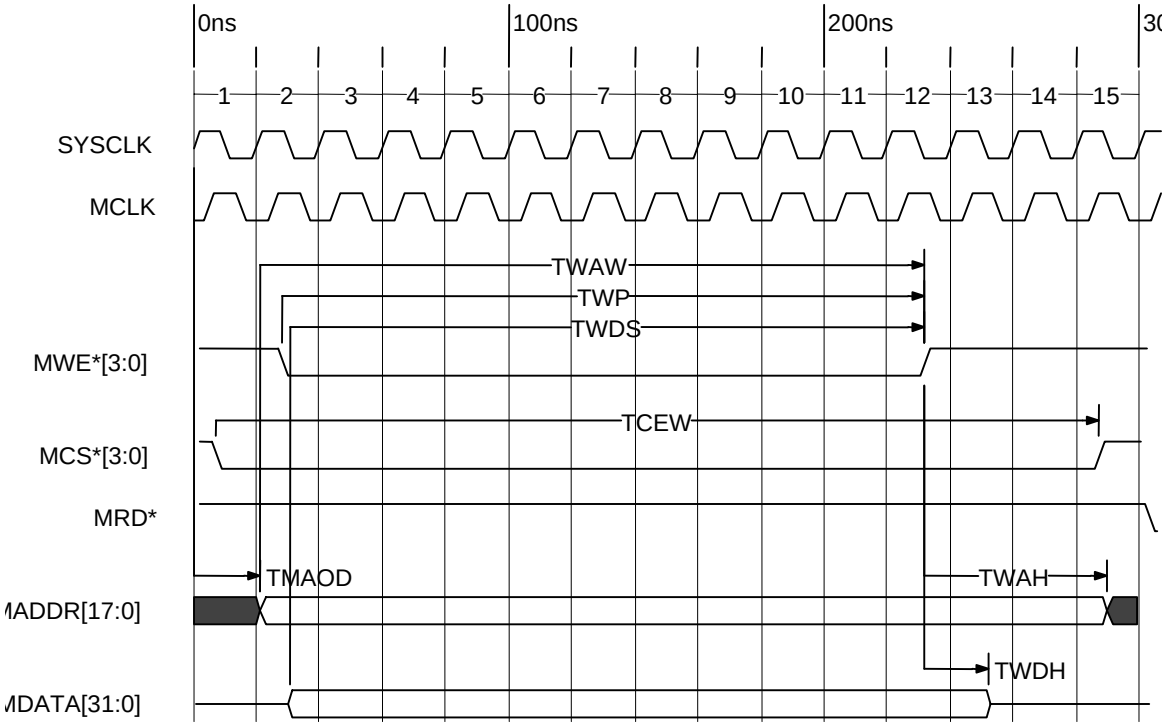
60 ns EDO DRAM AC Timing

Parameter	Description	MIN	TYP	MAX	Units
TCKP	MCLK period		20		nsec
TARD	Row address stable to MRAS* fall delay	15			nsec
TRAW	Row address width	35			nsec
TACD	Column address stable to CAS* fall delay	15			nsec
TCAW	Column address width	35			nsec
TCP	CAS* period	35			nsec
TCPL	CAS* low time	15			nsec
TWDS	Write data setup to CAS* fall	15			nsec
TWDH	Write data hold to CAS* fall	15			nsec
TRDS	Read data setup to CAS* fall	20			nsec
TRDH	Read data hold from CAS* fall	0			nsec
TREC	Read data HiZ to write data drive		15		nsec

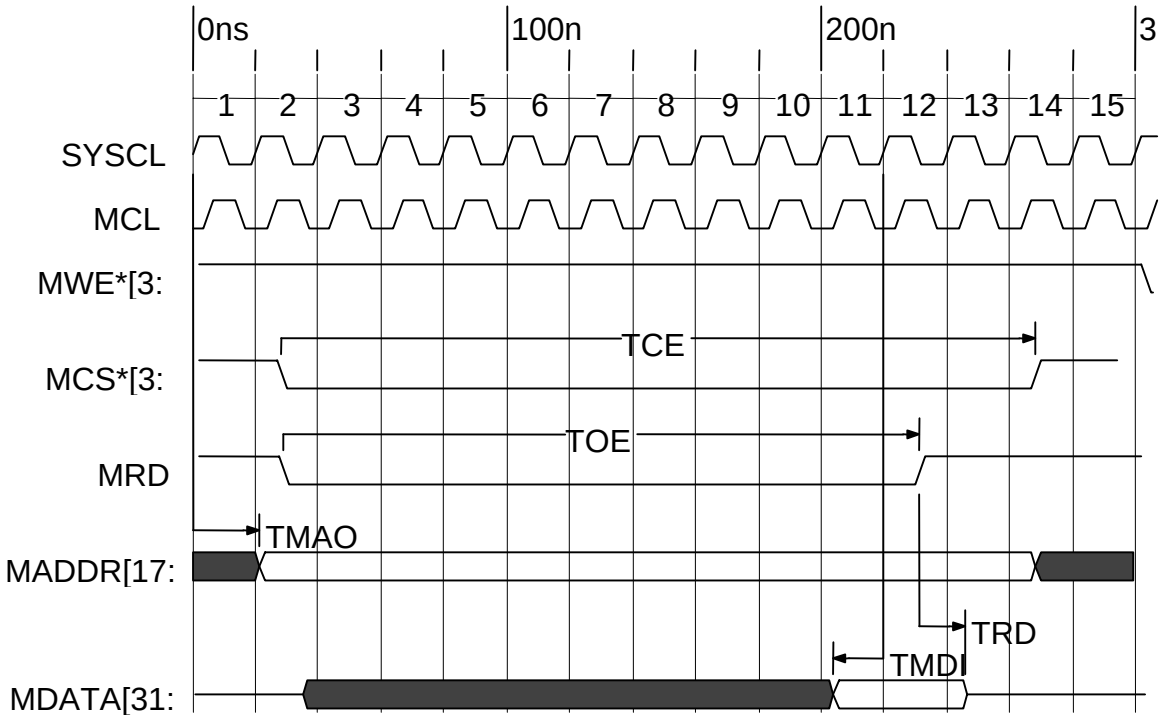
150 ns EEPROM/EPROM AC Timing

Parameter	Description	MIN	TYP	MAX	Units
TCKP	MCLK period		20		ns
TMAOD	Max MADDR[17:0] delay from SYSCLK (read, first cycle)		21		ns
TCEW	Min MCS*[3:0] pulse width		190		ns
TOEW	Min MRD* pulse width		190		ns
TMDIS	Min MDATA[31:0] setup to SYSCLK (read, 11th cycle)		10		ns
TRDH	Min MDATA[31:0] hold from MRD* rise (read, 12th cycle) (by design, data is latched internally in the cycle before the rise of MRD*)		0		ns
TWAW	Min MADDR[17:0] setup to MWE*[3:0] rise		190		ns
TWAH	Min MADDR[17:0] hold from MWE*[3:0] rise		40		ns
TWP	Min MWE*[3:0] pulse width (write, second cycle)		190		ns
TWDS	Min MDATA[31:0] setup to MWE*[3:0] rise		180		ns
TWDH	Min MDATA[31:0] hold to MWE*[3:0] rise		15		ns

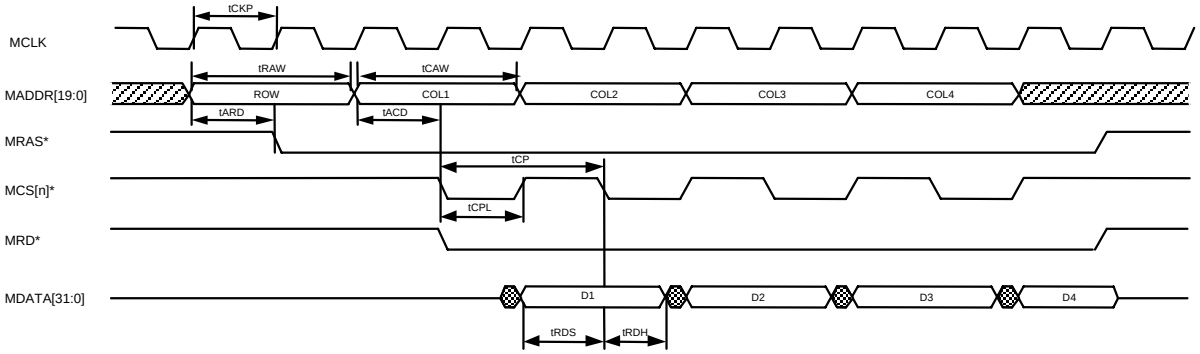
EEPROM/EPROM WRITE CYCLE



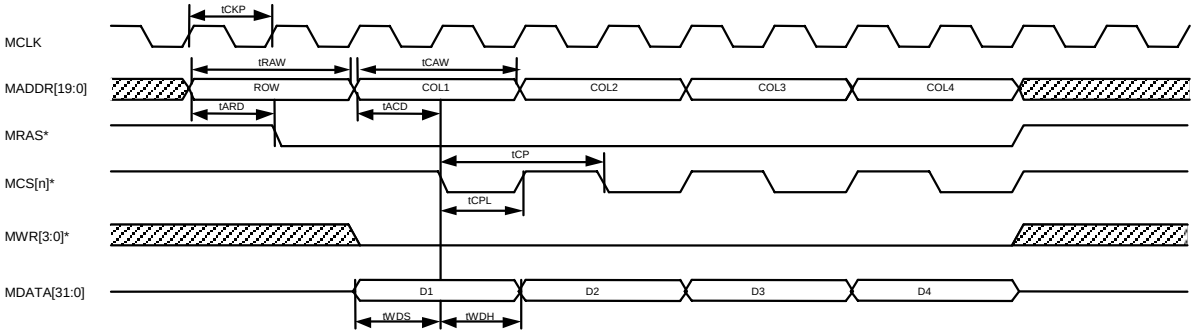
EEPROM/EPROM READ CYCLE



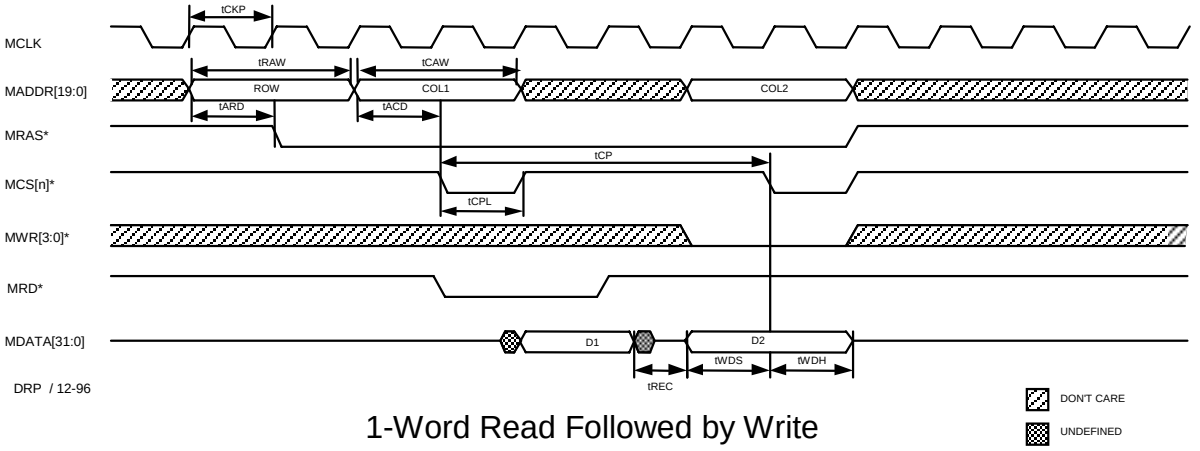
60 ns Single CAS EDO DRAM



4-Word Burst Read



4-Word Burst Write



1-Word Read Followed by Write

DRP / 12-96

Notes:

(1) Pin definitions shown above are for single CAS DRAM. Dual CAS DRAM is supported by selecting the MDCAS bit in the memory configuration register. In this mode, the CAS signals are driven by MWE*[3:0] and the RAS signals are driven by MCS*[3:0].

(2) Asserting the MSLO bit in the memory configuration register extends the CAS low time to support 80 nS DRAM

Clocking

Parameter	Description	MIN	TYP	MAX	Units
Fck	SYSCLK frequency[1]	1	50	50.5	MHz
Tch	SYSCLK High Pulse Width	8			nsec
Tcl	SYCLK Low Pulse Width	8			nsec
Fpck	PCICLK frequency[2]	0	40	40.5	MHz
Tpch	PCICLK High Pulse Width	11			nsec
Tpcl	PCICLK Low Pulse Width	11			nsec
Trst	RST* active time after PCICLK and SYSCLK stable	10			usec

Notes:

1) The minimum clock frequency for SYSCLK is required to reliably refresh local DRAM memory. The nominal frequency of 50 MHz must be provided for full throughput.

2) The minimum clock frequency for PCICLK reflects completely static operation. The nominal frequency of 40 MHz must be provided for full throughput.

Miscellaneous

Parameter	Description	MIN	MAX	Units
Tmisu	MINTR* setup to posedge SYSCLK	10		nsec
Tmih	MINTR* hold to posedge SYSCLK	0		nsec

FUNCTIONAL DESCRIPTION

Overview

This section will provide an overview of the PM3350 ELAN 8x10 system and internal components, the device initialization and configuration process, the structures built and used by the device to perform its functions, and a brief description of how packets are switched by the device.

System Components

In order to describe the functions and operation of the ELAN 8x10 device, it is first necessary to discuss the operating environment that is intended to be built around it. A typical simple switching system utilizing the ELAN 8x10 device consists of:

1. The ELAN 8x10 device itself.
2. A set of DRAM devices that provide the operating memory for the ELAN 8x10, mapped to the lowest bank of the ELAN 8x10 address space (i.e., addresses starting at 0x000000 hex). The amount of DRAM provided is a system-dependent parameter; the minimum amount required is 1 MB of RAM.
3. An EPROM containing the bootstrap image for the ELAN 8x10, i.e., the operating firmware, operating parameters, and system initialization code; this EPROM is mapped into the second lowest bank of the ELAN 8x10 memory address space, i.e., the 4 MB address range starting at address 0x400000 hex. The size of the EPROM is highly dependent on the operating configuration of the ELAN 8x10, with the minimum requirement being 32 kB.
4. External transceiver devices that implement the PHY layer functions required for 10BaseT Ethernet.
5. An LED register, connected to a bank of eight LEDs, that may be used to report status information for diagnostic purposes if required. The LED register is mapped into the third bank of ELAN 8x10 memory address space, starting at 0x800000 hex.
6. A system voltage monitor or other means of asserting a system reset for a specified time after the power supply voltage has stabilized, plus, if necessary, a pushbutton switch for forcing a system reset after power-up.
7. A set of pull-up and pull-down resistors that are connected to the ELAN 8x10 data bus in order to drive device configuration information on to the data bus during system reset.

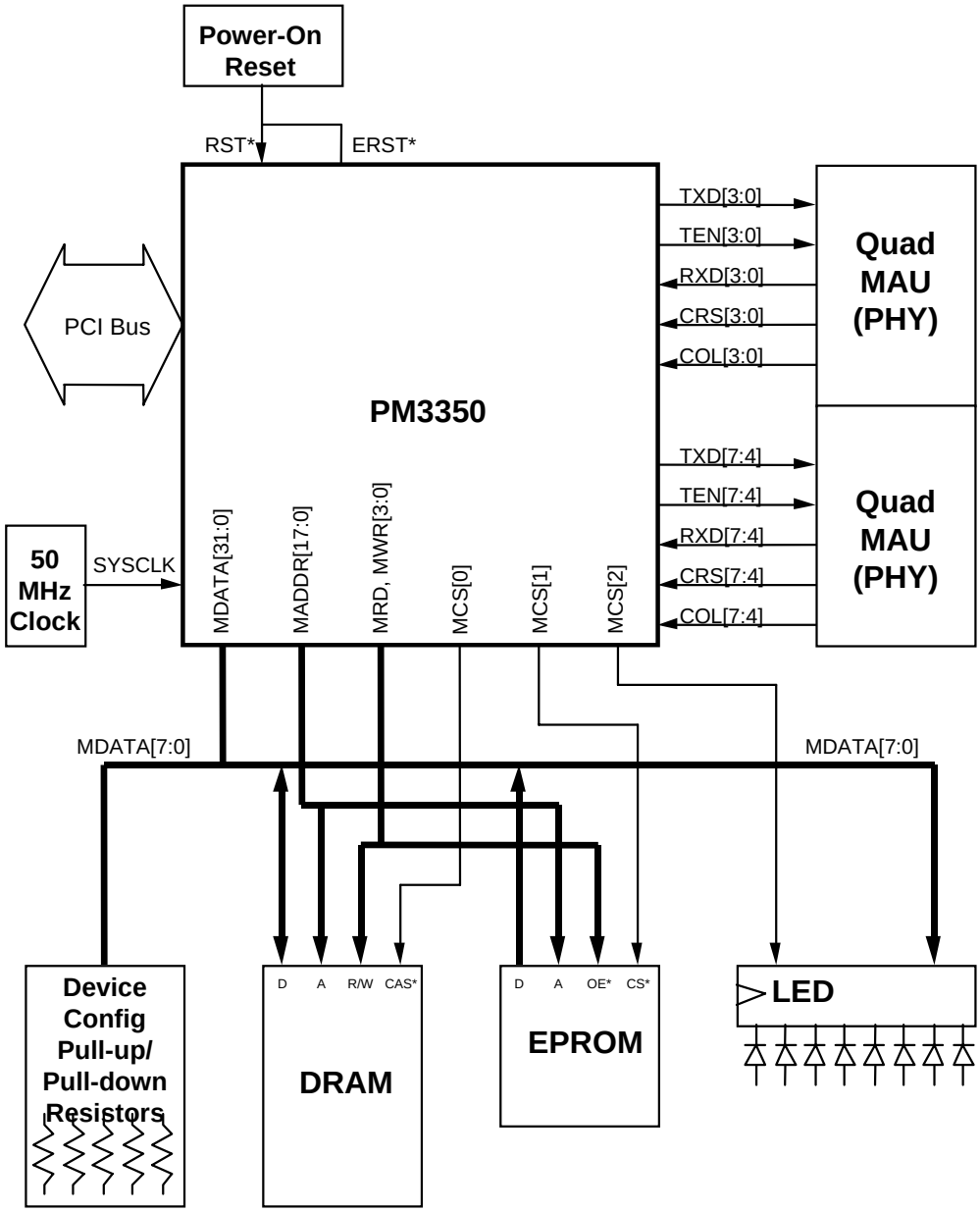
Note that the above system description assumes, for simplicity, a single ELAN 8x10 device in the system. Multiple devices may be interconnected via the PCI expansion

port of the ELAN 8x10 to build larger switches; however, the fundamental system operation does not change as additional ELAN devices are used.

RAM requirements for other system configurations may be computed using parameters supplied later in this document.

System Block Diagram

The following diagram represents a high-level view of the simple switch system described above:



In the diagram above, the quad-PHY devices, the EPROM, the DRAM and the 50 MHz clock oscillator are all created in a straightforward manner from off-the-shelf parts. The

device configuration resistors are merely resistor pull-ups and pull-downs that drive the memory data bus lines to specific values during reset, as described later; the resistor values used are not critical, and may range from 4.7k to 10k.

It is assumed that the minimum recommended configuration of 1 megabyte of DRAM and 32 Kbytes of EPROM are used. A typical implementation would use two 256k x 16-bit dual-CAS 60 ns Extended Data Out (EDO) DRAMs, together with a 256 kbit (32k x 8-bit) 150 ns EPROM. Larger memories may also be used if more buffer space or MAC addresses are to be supported; if this is done, the configuration parameters in the EPROM must be changed to reflect the increased memory size. The DRAM and EPROM device types and speeds are defined by the setting of the pull-up / pull-down resistors on the memory data bus at reset time.

The power-on reset generator can be created either from discrete components, or from a low-cost CPU monitor; the ERST_ output from the ELAN 8x10 chip is strapped to the reset signal to implement the watchdog capability of the ELAN device. Note that the ERST_ output may, as an alternative, be used to signal some external processor that the ELAN 8x10 device has encountered a fatal error condition requiring a software or hardware reset; in this case, the ERST_ output should be pulled up using a 4.7k resistor.

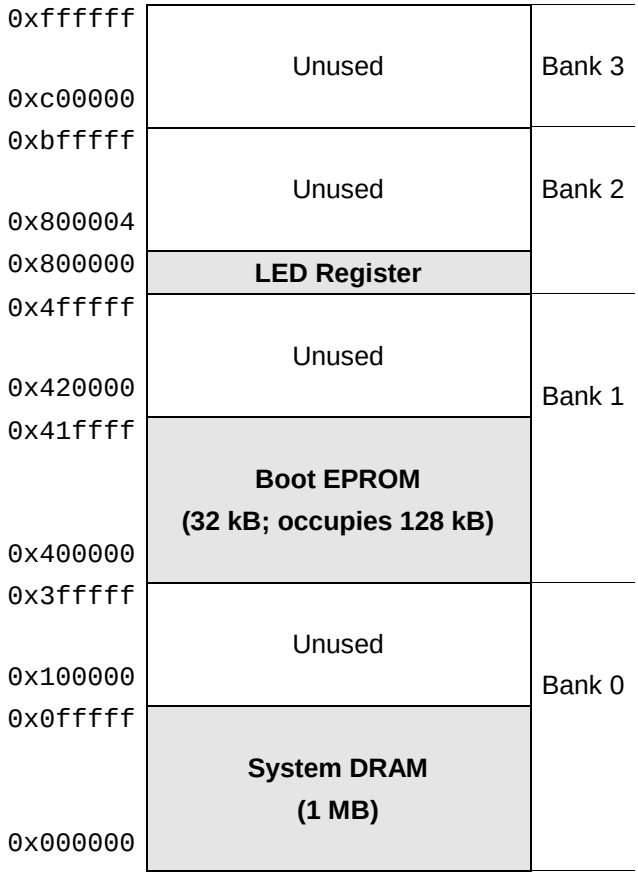
The LED register is implemented using a simple 8-bit TTL register with a clock enable that is tied to the indicated chip select output from the ELAN 8x10. Eight LEDs may be connected to the outputs of this register to present the diagnostic status codes output by the ELAN 8x10 firmware during self-test, system boot and operation. If a simple TTL register is used, the LED register is effectively write-only; writes to this register will modify the state of the LEDs, but reads from this register return invalid values. A read-back register can be used if this is a significant issue.

The first three chip select lines from the ELAN 8x10 (i.e., MCS_[0], MCS_[1] and MCS_[2]) are tied to the DRAM, the EPROM and the LED register, respectively; the remaining chip select is unused. This maps the DRAM into the first 4 MB bank of address space, the EPROM into the second bank, and the LED register into the third bank. The address map in the following subsection gives the 24-bit address ranges assigned to each resource.

This system has only been presented to serve as a basis for the following discussion on the device and system operation, and is not intended to serve as a complete example or reference design. More details on actual system construction with the ELAN 8x10 may be found in the relevant application notes and reference design documents.

System Memory Map

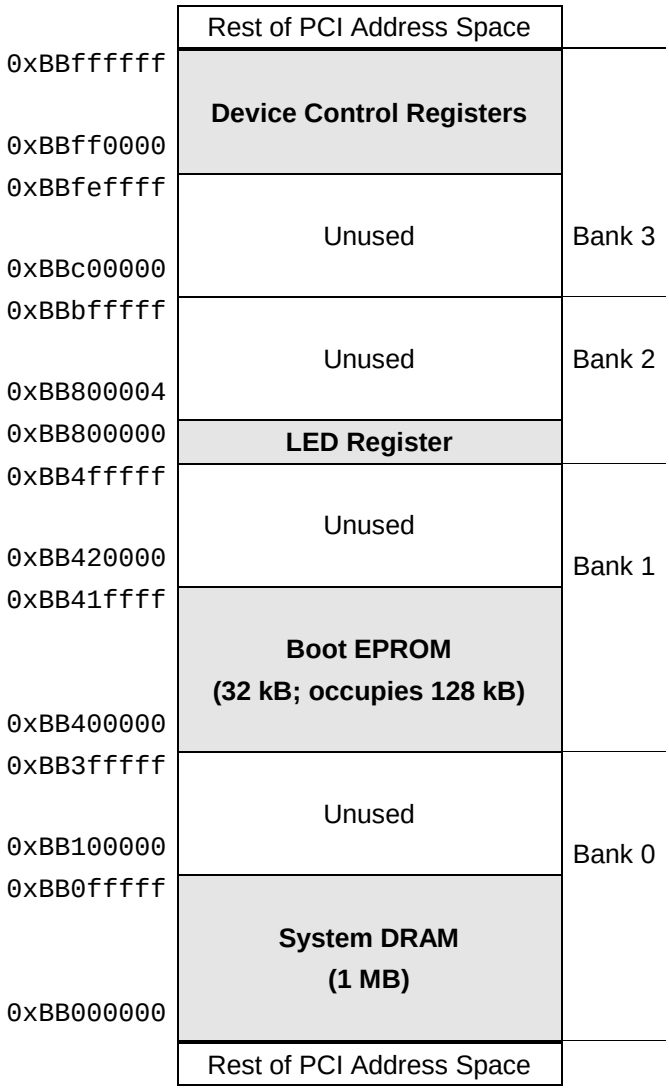
The ELAN 8x10 device uses a single linear (flat) byte-addressable 16 MB address space for accessing memory and memory-mapped devices. The external memory map for the system described above is quite straightforward. From the point of view of the local memory bus, it is as follows (note that the memory addresses shown increase upwards):



Note that the 32 kB boot EPROM actually occupies 128 kB of address space. This is because the EPROM is only 8 bits wide (i.e., a 32k x 8-bit configuration), and so is connected to the least significant byte lane of the memory data bus. Hence each byte of the EPROM takes up a full 32 bits worth of address space, leading to the 128 kB requirement.

In a similar manner to the 8-bit wide EPROM, the LED register is mapped to the least significant 8 bits of the data bus, but takes up a full 32 bits of address space. No other device is shown as being mapped to the ELAN 8x10 address space in this simple and minimal system; however, other devices (such as RS232-C serial ports) may be interfaced as well, provided that firmware is developed to support them.

When viewed from the PCI bus, the ELAN 8x10 device appears to take up a 16 MB block of contiguous addresses in the total 4 GB PCI address space. A virtually identical memory map is presented to the PCI bus when accessing the ELAN 8x10 device as a PCI slave (target), with the exception that a set of device control and communication registers are implemented in the uppermost 64 kB of the 16 MB address space used by the ELAN 8x10 device (note that the memory addresses shown increase upwards):



The start address of the block occupied by the ELAN 8x10 is defined by the setting of the memory base address register within the PCI configuration register space of the ELAN 8x10 device, and is represented by the *BB* component of the addresses given in the table above. The memory base address register may be set to any arbitrary value via a PCI configuration write after system reset, provided that the base address is on a

16 MB boundary and that the ELAN 8x10 operating firmware parameters are set consistently with the selected base address.

Device Internal Blocks

The ELAN 8x10 consists of the following major components: a Switch Processor, eight MAC interfaces, a DMA Controller, a memory controller and a PCI expansion port. In addition, it also implements a watchdog reset circuit and some auxiliary functions.

Switch Processor

The Switch Processor is a 50 MHz proprietary RISC processor that executes the firmware required for carrying out all the packet switching and device control functions of the ELAN 8x10. It is specifically designed to support LAN protocols at high speeds in a closed embedded system environment. The Switch Processor contains various hardware features that permit it to carry out all of its functions at maximum efficiency, and is tightly coupled to the rest of the ELAN 8x10 device logic

The Switch Processor interfaces to the rest of the ELAN 8x10 device via several dedicated hardware ports:

1. It uses a special control register access bus to read or write any of up to 96 16-bit control registers that are implemented by the internal hardware units; these registers are used to set configuration parameters in various ELAN 8x10 units, read the unit status, set various operating parameters (such as address pointers), and perform device self-test.
2. It implements a set of thirteen level-sensitive hardware interrupts that are connected to various blocks within the ELAN 8x10; these interrupts are the primary task dispatching entity for the base switching code. Assertion of an interrupt line causes the corresponding interrupt service routine (ISR) to begin executing, and execution normally proceeds until the ISR has finished servicing the unit that required attention.
3. A set of 32 general-purpose outputs and 15 general-purpose inputs are provided. These are connected to various low-level control and status signals presented by various ELAN 8x10 internal logic blocks. The general-purpose inputs and outputs considerably speed up the testing of the state of the logic blocks and also the control of their functions, as multiple tests on inputs or multiple modifications of outputs can be performed in a single instruction.
4. A set of 16 coprocessor condition tests are implemented by the Switch Processor. These inputs are used to signal high-level device conditions generated by various ELAN 8x10 functional units to the Switch Processor firmware.

The internal and external registers implemented by the Switch Processor and the associated ELAN 8x10 functional units, as well as the view of the debug registers from the PCI bus interface, are presented in subsequent sections.

The Switch Processor expects to locate its operating firmware as part of a *boot image* present in the external memory space. The format of the boot image is described later.

Ethernet MAC Interfaces

Eight identical interfaces to external Ethernet/IEEE 802.3 MAU devices are provided on-chip. The ELAN 8x10 performs only the 802.3 MAC-layer functions: serial/ parallel conversion, packet generation and extraction, jamming and backoff after collision, deference, interframe gap enforcement, and buffering. These are performed with a combination of dedicated hardware in the MAC interfaces and microcode running on the Switch Processor. The external transceivers (MAU devices) are expected to implement all the standard 10BaseT MAU functions: line driving/receiving, clock generation and recovery, Manchester encoding and decoding, and carrier and collision detect.

Each MAC interface unit consists of the following functional blocks:

- Interfaces to the receive and transmit ports of the external 10 Mb/s MAU devices. These interfaces transfer serial, independently clocked receive and transmit data between the ELAN 8x10 and the MAU, and also allow the MAU to report carrier detect and collision status.

- A serializer/deserializer that converts between the 8-bit parallel data format used by the ELAN 8x10 MAC logic and the 1-bit serial format used by the MAU.

- A 32-byte bi-directional FIFO buffer that holds parallel transmit data prior to parallel-to-serial conversion, and receive data words after serial-to-parallel conversion. The FIFO buffer also decouples the external MAU transmit and receive clocks from the internal ELAN 8x10 device clock, and converts between a byte-wide interface to the serializer/deserializer and a word-wide interface to the DMA Controller.

- Control/status registers and logic that allows the Switch Processor to control the MAC port and the MAU devices, and also to monitor status. The control/status registers contain the timers required for the CSMA/CD algorithm.

During transmission, 32-bit parallel data to be transmitted are placed in the bi-directional FIFO buffer of each MAC port by the DMA Controller. After any required deference or interframe gap time, these are read out as 8-bit bytes, serialized and sent to the external MAU for transmission on to the twisted-pair or coaxial link. The IEEE 802.3 standard 2-part interframe gap timing algorithm is implemented.

If a collision is detected (as reported by the external MAU) the MAC port forces transmission of a jam sequence (of configurable length) and then halts transmission for the specified backoff time, flushing its FIFO at the same time and signaling a collision. An internal backoff timer is then loaded with the appropriate value; when the backoff period is timed out, the MAC logic will re-attempt transmission. The computation of the backoff period used for each collision on each MAC channel is implemented by the Switch Processor, using a random number generator modified according to the truncated binary exponential backoff algorithm specified by the IEEE 802.3 standard. The Switch processor always pre-loads the backoff period that will be used for the **next** collision into a backoff reload register within the MAC channel, thereby avoiding collision backoff time variation due to firmware latencies.

Data received from the twisted-pair or coaxial cable by the MAU are converted to 8-bit parallel bytes by the deserializer and written to the FIFO, and subsequently read out by the DMA Controller as a stream of 32-bit words. The MAC logic performs the preamble detection and stripping necessary to frame to the incoming data. As few as 14 consecutive bits of valid preamble (i.e., a 1010... bit pattern) may be presented to the MAC port, along with an SFD, to guarantee proper frame synchronization; this permits the phase locked loop settling time requirements in the physical layer transceivers to be considerably relaxed. If an invalid pattern is detected in the preamble prior to receiving a valid SFD, then the MAC logic rejects the entire frame, waits until the carrier sense from the physical layer transceiver goes inactive, and begins looking for the next valid frame.

After the carrier sense input from the external transceiver is de-asserted (indicating that frame reception has ended), the ELAN 8x10 MAC logic blinds the receiver for a programmable interval. During this time, transients on the carrier sense input will be ignored (unless the MAC port was transmitting just previously) as per the IEEE 802.3 standard. The blind timer also defines the minimum tolerable interframe gap that can be accepted by the MAC port during back-to-back frame receives.

The MAC logic also implements an internal configurable jabber counter to time out excessively long transmissions or reception; thus the ELAN 8x10 does not require the external MAU to provide jabber protection.

Each MAC channel implements an optional flow-control mechanism to backpressure the channel in the case of local congestion (e.g., an out-of-buffers condition). Backpressure is accomplished by continuously transmitting an extended jam sequence (all-zeros) pattern, with gaps of one standard interframe spacing inserted periodically to prevent the backpressure pattern from being interpreted as a jabber. Collisions encountered during backpressure will not cause the MAC channel logic to back off in the normal manner; instead, the MAC channel will send the standard jam pattern, time out a normal interframe gap, and then resume the backpressure pattern. If, during backpressure, the MAC channel detects that one or more frames are ready to be transmitted over the channel by the ELAN 8x10 (e.g., a frame was received from some

other channel that is destined for an end-station on the flow-controlled channel), the MAC channel will cease transmitting the backpressure pattern, wait for a normal interframe gap, and then transmit the desired frame(s). After all the pending frames have been transmitted, the MAC logic will resume backpressuring the channel.

The backpressure mechanism described above is an optional feature that may be enabled on a per-port basis by means of configuration parameters in the boot image. It should be noted that the backpressure method will result in the complete shutdown of the MAC channel (i.e., even local segment traffic will not be allowed to proceed), and so it is recommended that backpressure be enabled only on ports that are connected to a small number of end-stations.

The ELAN 8x10 MAC channels also provide a frame type filtering feature that extracts the 16-bit EtherType fields of incoming frames (occupying the portion of the Ethernet frame header immediately after the destination and source MAC addresses), compares it to a programmable 16-bit value, and signals the Switch Processor firmware if a match is found. This facility may be used, for example, to quickly identify MAC Control frames (defined in IEEE 802.3x) or Address Resolution Protocol frames belonging to the TCP/IP protocol. The Switch Processor firmware may then handle such marked frames in a special manner.

Each MAC channel maintains a pair of control and status registers that allow the Switch Processor to monitor and manage the state of the MAC channels. The MAC interfaces are configurable (as a group) by a set of configuration registers that are loaded at initialization time. The following parameters are configurable:

Parameter	Min	Max	Units
Interframe spacing, Part 1	0	255	bit times
Interframe spacing, Part 2	0	255	bit times
Transmitted preamble length (not including SFD byte)	0	255	bits
Jam length (after collision)	1	255	bit times
Maximum frame size	1	4095	bytes
Minimum frame size	1	255	bytes
Late collision threshold	0	1024	bit times
Receiver blind time	0	255	bit times
EtherType comparison value	0	65535	N/A

It should be noted that the Part 1 interframe spacing may be set to zero, in which case the interframe behavior of the ELAN 8x10 MAC defaults to that specified by the Ethernet V2.0 standard. If the Part 1 interframe spacing is non-zero, then the minimum interframe gap that can be tolerated by the MAC port during back-to-back frame receive will be the **greater** of the Part 1 interframe spacing and the receiver blind time.

Note:

None of the MAC control, status or configuration registers are visible from the PCI bus interface; they may be accessed only by the Switch Processor. More details on individual registers within the MAC channels are given later.

Multichannel DMA Controller

The DMA Controller is responsible for performing all data block transfers within the ELAN 8x10 and for computing the 32-bit Ethernet CRC check performed on packets received from or transmitted to the MAU channels. A special feature of the DMA Controller is its ability to automatically allocate buffer storage from a central free pool (organized as a linked list pointed to by a dedicated device register) when receiving data from a MAC channel or the expansion port. In addition, the DMA Controller implements an address hash table look-up capability that automatically resolves the source and destination MAC addresses of incoming Ethernet frames using an address table built in external RAM.

The DMA Controller supports a total of eleven separate channels, all of which may be running concurrently. Each channel is assigned a dedicated set of locations within an internal register file to hold transfer parameters and status. All but two of the DMA channels are capable of handling linked-lists of fixed-length *packet buffers*, which are chained together in varying numbers to hold Ethernet packets of different lengths. The eleven channels are dedicated to various functions as follows:

- Eight channels are used to perform data transfers between the 8 MAC channel FIFOs and the local memory, copying data from the local memory to the MAC FIFOs on transmit or from the FIFOs to the local memory on receive. These channels are capable of generating (on receive) and following (on transmit) linked-lists of packet buffers in external memory. A 32-bit CRC is computed on both transmit and receive data transfers; the CRC result may be optionally appended to the transmit data, and is accessible to the Switch Processor firmware in the form of a CRC check error bit after a packet has been received. Packet buffers are automatically allocated from a central free pool on receive. The source and destination MAC addresses of frames received from the MAC channels will also be extracted and used to look up the address entries in an external hash table.

- One channel is used to perform block transfers from the external PCI expansion bus to the local memory. This channel can follow a chain of remote source packet buffers, reading data over the PCI bus and creating a corresponding chain of packet buffers in the local memory, automatically allocating the required local packet buffers from the free pool.

- One channel is used to perform block transfers from the local memory to external memory space accessible via the PCI expansion bus. This channel is not intended to handle packet data; it is generally used for special control and inter-device communication functions.

- The last channel facilitates the packet exchange handshake that takes place between multiple ELAN 8x10 devices located on the same PCI bus. This channel can be set up to perform up to seven writes in a single series of PCI transactions to special request and acknowledge counters in up to seven external ELAN 8x10 devices. The request and acknowledge counters are used to indicate the presence of a packet being forwarded and to acknowledge receipt of the packet, respectively.

Transfers are initiated under firmware control by the Switch Processor. In operation, the Switch Processor loads the appropriate locations within the DMA register file with the desired transfer parameters, and then enables the channel using the DMA control register. The DMA Controller logic then reads out the transfer parameters into internal temporary registers, performs the transfer, and updates the register file. The DMA automatically multiplexes channel transfers by switching to a new channel after a burst of up to 4 data words (i.e., 16 bytes) has been read or written, thus minimizing latency.

Memory Controller

An external memory is required to hold packet buffers for received and transmitted Ethernet packets, as well as data structures needed to perform switching and support system management. The external memory may also contain extension firmware for the on-chip Switch Processor. The ELAN 8x10 therefore contains an integral memory controller unit to support a variety of standard memories without glue logic.

The memory controller addresses a total of 16,777,216 bytes (16 megabytes) of EDO DRAM, ROM, EPROM and EEPROM (either separately or in combination). The address space is divided into four banks, each of which has an independently programmable access time. This allows a mix of fast and slow devices to be used in the system. Memory device types or speeds cannot be mixed within a bank (i.e., a bank cannot consist of part EDO DRAM and part EPROM, for example). Memory devices of different types must be assigned to different banks, and selected with different chip selects.

Programming of EEPROM devices is done under control of Switch Processor microcode.

To support memories other than those listed above, or special applications, a memory ready input pin (MRDY_) is provided. This pin is sampled by the ELAN 8x10 when accessing memories in asynchronous SRAM mode; it may be driven inactive prior to the end of any memory cycle by an external memory timing generator to suitably

lengthen the access cycle. The MRDY_ pin should be tied LOW (permanently asserted) if it is not used.

The ELAN 8x10 generates four separate write enables to enable individual bytes within each addressed 32-bit word to be written to independently of the others. This allows the ELAN 8x10 to perform byte (8-bit), halfword (16-bit), tribyte (24-bit) and fullword (32-bit) memory accesses without using read-modify-write operations.

An interrupt pin (MINTR_) is provided for special applications. A LOW level on this pin causes an interrupt to be generated to the internal Switch Processor, the PCI bus (via the INT_ output), or both. The use of this interrupt input is application dependent and beyond the scope of this datasheet. The MINTR_ pin should be tied HIGH (de-asserted) if it is not used.

The memory controller is capable of sustaining a throughput of 84 Mbytes/s (672 Mbit/s) to and from standard EDO DRAM devices, with peak throughputs of 100 Mbytes/s during burst accesses.

PCI Expansion Port

The ELAN 8x10 includes a PCI v2.1 compatible bus master and slave interface, which serves as an expansion port allowing multiple ELAN 8x10s to be seamlessly interconnected in the same system. The expansion port supports a maximum PCI bus clock of 40 MHz (resulting in a >1 Gbit/s peak transfer rate and a sustained throughput of over 400 Mbit/s), and contains several FIFOs to increase burst throughput and perform clock synchronization. This port may be used for expanding a switch built around 8-port ELAN 8x10 chips (to a maximum of 64 ports In this application, the on-chip DMA Controller uses the PCI master interface to notify other ELAN 8x10 devices of the presence of packets to be transferred, and to copy packets or data structures under control of the Switch Processor between external ELAN 8x10 devices to the local memory space. Note that data are never transferred directly between the MAC channels and the PCI bus.

The PCI bus master interface serves to allow the DMA controller as well as the CPU to initiate transactions on the PCI bus. The bus master is compatible with the PCI v2.1 specifications for standard transaction initiator devices, and can perform configuration space as well as memory space read and write transfers. (Note that the ELAN 8x10 does not support I/O space transactions.) The PCI bus master unit contains a 64-byte write FIFO to buffer data being written by the ELAN 8x10 device to an external target on the PCI bus, as well as a 128-byte read FIFO to hold data that has been read from an external target. These FIFOs permit the bus master to operate using long burst transactions for increasing the PCI bus bandwidth utilization. The bus master interface is also compatible with the PCI v2.1 latency timer requirements, and supports back-to-back transfers.

The PCI slave interface logic within the expansion port block responds to transfer requests from external bus masters, performing the necessary accesses and transferring the requested data. The slave interface logic acts as a responder during frame transfers between devices belonging to the ELAN chipset, supplying Ethernet frame data to requesters as necessary. An external processor can also use the PCI slave interface to gain access to internal device registers and data structures in the local memory space, or to download firmware or data to the ELAN 8x10 Switch Processor. The slave interface unit contains a 32-byte write FIFO and a 128-byte read FIFO to improve burst behavior during PCI reads and writes: the write FIFO buffers data being written to the ELAN 8x10 by external devices, while the read FIFO holds data read from local memory or internal registers in response to PCI memory read commands from external requesters. Note that the slave interface only responds to configuration space and memory space reads and writes; I/O space reads and writes are not supported. The slave interface conforms to the access latency, access ordering and disconnect rules of the PCI v2.1 standard. A set of special registers are provided in the PCI configuration space that may be used to alter the access latency rules imposed by the slave logic in order to improve PCI bus utilization if required.

The expansion port also contains hardware to speed up intercommunication between ELAN 8x10 devices via the PCI bus. This hardware takes the form of seven request counters and seven acknowledge counters. Each request/ acknowledge counter pair is dedicated to supporting communications with a specific external ELAN 8x10 device. An external ELAN 8x10 device will increment a request counter to signal that a packet or message data is available to be read, and will increment an acknowledge counter to acknowledge that it (the external ELAN 8x10) has read a message or packet from the local ELAN 8x10. Standard PCI reads and writes can be used to increment the request and acknowledge counters. More details on these counters are supplied below.

The expansion port is compatible with the "PCI Local Bus Specification", release 2.1. To support the use of the ELAN 8x10 in standalone switch system applications, a simple external bus arbiter (easily implemented in a single programmable logic device) must be provided to arbitrate between PCI bus accesses of multiple ELAN 8x10 devices.

The PCI bus interface within the expansion port runs synchronously to the PCI bus clock, which must be supplied on the PCI_CLK input pin. The duty cycle of the PCI_CLK pin is as specified in the PCI Local Bus Specification whereas the operating frequency used by the ELAN expansion bus interface is 40 MHz, as opposed to the PCI Local Bus Specification value of 33 MHz. The ELAN 8x10 PCI bus interface implements synchronization logic to transfer data between the PCI bus clock domain and the internal device clock.

PCI Transactions Supported

The on-chip PCI interface is capable of initiating the following commands:

- memory read
- memory write
- configuration read
- configuration write

The on-chip PCI interface supports the following PCI commands as a target:

- memory read
- memory write
- configuration read
- configuration write
- memory read multiple
- memory read line
- memory write and invalidate

PCI Vendor ID and Device Number

The vendor ID assigned to the ELAN 8x10 device (in the PCI configuration register space) is 11F8 hexadecimal, while the device number is 3350 hexadecimal. The class code for the ELAN 8x10 is set to 0x020000 hexadecimal.

Slave Read Prefetching

The PCI bus slave logic contains a 128-byte read FIFO buffer to speed up reads made from this ELAN 8x10 device over the PCI bus. This FIFO has a prefetch capability that is activated when accessing external memory space: it attempts to read ahead and speculatively obtain more data words than have been actually requested by the transaction master, thereby potentially increasing the efficiency of burst transfers.

The prefetch capability functions as follows. Initially, the PCI slave read FIFO is empty, and remains so until the PCI slave is idle. When a read transaction is initiated by an external bus master, the PCI slave logic will perform a disconnect with retry (after a configurable amount of cycles) because the read FIFO is empty and no data can be returned. The slave logic then decodes the target address of the read: if it corresponds to external memory space, then the prefetch capability is activated. The slave logic subsequently requests the memory controller to begin fetching from the target memory address; the data returned are placed into the slave read FIFO, and the PCI slave logic continues to fetch additional data words at consecutive addresses until the slave read FIFO is full. Up to 128 bytes of data may be fetched in this way and placed into the slave read FIFO.

At some point, the original PCI transaction initiator (bus master) is expected to retry the access. (According to the rules of the PCI bus, it is an error for the bus master to abandon an access that has been terminated with a disconnect-and-retry.) The PCI slave logic will compare the address being requested by the PCI bus master for the

retried transaction to the address from which it began the prefetch; if a match occurs, the slave logic will return the data present in the read FIFO in a continuous burst of back-to-back data phases on the PCI bus. The burst of transfers on the PCI bus will continue as long as (1) the read FIFO is not empty and (2) the bus master does not terminate the access.

If the read FIFO empties during a transfer (possibly because the memory controller cannot satisfy the requests from the PCI slave logic at a sufficient rate), the PCI slave logic will issue another disconnect with retry, and continue to request the memory controller for more data. The bus master is again expected to retry the access, and the cycle continues.

If the bus master terminates the access in any way, the PCI slave logic will stop placing data on the PCI bus and flush the read FIFO to discard any remaining unread data words. It will then proceed to fulfill the next PCI read or write access.

The use of the PCI slave read FIFO enhances the ability of the PCI slave logic to maintain the utilization of the PCI bus by transferring data in long bursts. If sufficient delay is inserted by the bus master between the initial disconnected access and the subsequent retry, the PCI slave will have time to read ahead by a substantial number of words (up to 32) and can therefore transfer data in a long burst when the bus master finally retries the access.

To further improve the efficiency of the PCI slave interface, the slave logic has the capability of latching and holding up to two different target read addresses from two different bus masters at a time. This allows the first bus master to make a read access, which will be disconnected with retry by the PCI slave logic after the target address has been latched; another bus master can then make another read access at a different address, which will also be disconnected with retry by the slave logic after the address has been latched. (Only two such addresses can be latched by the slave; if yet a third master makes another read access to another address, the slave logic will also disconnect this master with a retry, but will not latch the address internally.) The availability of the second read address means that the PCI slave logic can begin immediately fetching data from the second location into the read FIFO after the first bus master terminates its access, without waiting for the second bus master to retry the access, and hence the PCI slave logic can improve its utilization of the available memory and PCI bus bandwidth.

Note:

Writes are not allowed to be completed out-of-sequence with reads, and vice versa.

Watchdog Timer Facility

The ELAN 8x10 device incorporates a simple internal watchdog timer that optionally initiates an automatic system hardware reset if some catastrophic error occurs that causes the Switch Processor to lock up or enter an undefined state. The watchdog timer is built around the 16-bit WTIMER device control register (described in more detail in a later section).

The WTIMER register principally acts as a down-counter, decrementing its value by 1 every millisecond until it reaches zero. Firmware running on the Switch Processor will, under normal circumstances, periodically reload the WTIMER register with a non-zero value before it reaches zero. If however, the Switch Processor firmware encounters some serious system fault that prevents it from reloading the WTIMER register before it has counted down to zero, the watchdog facility will assert the ERST_ output LOW for one millisecond. If the ERST_ output is tied to the system reset line (this is made possible by the fact that ERST_ is an open-drain output), then the watchdog timer facility will effectively reset the entire system. Alternatively, the ERST_ output can tied to a resistive pull-up and simply monitored by an external system processor; a hardware or software reset of the ELAN 8x10 device should be performed if the ERST_ output goes LOW.

The value used by the Switch Processor firmware to reload the WTIMER register is a configurable parameter, and should be chosen to ensure that false system resets do not occur under high loads without simultaneously incurring an excessive system recovery time. The maximum reload interval is approximately 65 seconds; the minimum interval is about 2 milliseconds. A value of 1-2 seconds is recommended.

If the WTIMER register is loaded with all-ones (0xffff hex) the watchdog timer facility will be disabled, and the WTIMER register will be prevented from counting down and asserting the ERST_ output. (The watchdog facility can hence be disabled by the system implementer by setting the configurable reload value to 0xffff hex.) Note that the WTIMER register rolls over to 0xffff after it has counted down to zero, thereby automatically disabling itself after the 1 millisecond reset duration is over. If the WTIMER register is left untouched by the initialization process, then it will default to a disabled state, i.e., it will remain loaded with 0xffff and, as a result, will not count down.

If the Switch Processor itself detects an unrecoverable fault that requires a general system reset, then the Switch Processor firmware will write a value of zero to the WTIMER register under program control. This will cause the ERST_ pin to be immediately asserted (driven LOW), potentially causing a hardware reset or notifying the system processor that a fault has occurred.

Configuration and Initialization

Initialization and configuration can be considered as either a two step or four step process. The two-step process is the default, and used for most normal applications; the four-step process may be followed to over-ride the default configuration information for special purposes.

Two-step process

For a two-step process **all** of the basic device and system configuration information is taken from the local memory interface of this ELAN 8x10 chip using this sequence:

- Basic hardware configuration information, comprising a 32-bit device and memory configuration word, is latched by the ELAN 8x10 from the memory data lines into internal registers upon RST_ transitioning high (i.e., upon the trailing edge of a hardware reset). The 32-bit configuration word is held in the MCONFIG (Memory Configuration) and DCONFIG (Device Configuration) registers.

- The remaining device configuration and initialization information is obtained automatically by the ELAN 8x10 by scanning for an external EPROM or EEPROM, and reading a bootstrap image from it. The bootstrap image contains a number of system-dependent configuration parameters, the operating firmware for the Switch Processor, and any code required for a Real-Time Operating System, SNMP agent, spanning tree processing, etc.

Four-step process

For a four-step process **not all** of the basic device and system configuration is taken from the local memory of this ELAN 8x10 chip: an external host processor, or a master ELAN 8x10 device, is expected to modify or download configuration and bootstrap data.

1. Configuration information is latched from the memory data lines into internal registers upon RST_ transitioning high as described above. The RISCRUN bit (bit 30, see the next section) of the 32-bit configuration data word must be cleared to ensure that the Switch Processor enters a halt state immediately after reset. The PCIRUN bit (bit 31) of the configuration data word may also be cleared if the external host processor intends to carry out the normal PCI configuration register setup process as per the PCI specification.

2. The DCONFIG and MCONFIG registers of the device are accessible via the PCI bus interface, and may be modified if necessary by the system master. It is also possible for the system master to download a preformatted bootstrap image to the RAM interfaced to the device; the ELAN 8x10 will locate the bootstrap image and self-initialize from it.

3. A software reset is performed to the device using the HCTRL register, also accessible via the PCI bus, by writing the GRES bit first to a logic 1 and then to a logic 0. Note that GRES must remain asserted a minimum of 128 PCI clock cycles. The updated DCONFIG and MCONFIG register information will be internally latched upon de-assertion of the GRES bit.
4. The remaining device configuration and initialization will be obtained by scanning for and reading a bootstrap image contained in an external EPROM or EEPROM, or downloaded into system RAM by the master processor.

Device Configuration

Basic device and system configuration (i.e., memory types and speeds for various banks, the PCI base address for this ELAN 8x10 device, and auto-boot and master/slave enable flags) are supplied by means of resistor pull-ups and pull-downs connected to the 32-bit data bus. This configuration information is latched into internal registers upon the second SYSCLK rising edge after the RST_ input to the ELAN 8x10 transitions high, and sets up the ELAN 8x10 internal hardware. The 32 bits of configuration data presented on the memory data bus are latched into the 16-bit DCONFIG and MCONFIG registers internal to the ELAN 8x10; these registers may also be accessed by the Switch Processor and by external devices via the PCI bus.

As an alternative to resistor pull-ups and pull-downs, a tri-state buffer or tri-statable register may be used to drive configuration information on to the data bus during reset. Care should be taken to remove the data by tri-stating the buffer or register no earlier than 2 SYSCLK periods after the trailing edge of the RST* input, and no later than 10 SYSCLK periods after the latter (to prevent memory data bus contention).

The memory data bus is mapped to configuration bits as follows:

Device Pin	Register Bit	Description
MDATA[31]	PCIRUN	This input selects the default operating mode of the PCI interface. If logic 1: • The on-chip PCI interface latches its slave memory base address from the CHIPID configuration bits (MDATA[25:22]). • The PCI Command Register bits for "Bus Master" and "Memory Space" are set (1), thereby allowing the device to respond to PCI memory space accesses and to be a bus master. If logic 0 • The PCI interface has a memory base address of 0. • The PCI Command Register bits for "Bus Master" and "Memory Space" are cleared (0); the device is disabled from responding to PCI memory space accesses and will not be a bus master.
MDATA[30]	RISCRUN	A low on this signal halts the Switch Processor upon reset, effectively placing the device into stand-by mode.

MDATA[29]	RSTTM	For test purposes only. Pull low for correct operation.
MDATA[28]	IMDIS	Internal memory disable: if high, the internal Switch Processor ROM is disabled at initialization time.
MDATA[27]	PCI3V	If high, configures the PCI interface for the 3.3V signaling environment. If low, configures the PCI interface for the 5V signaling environment. Must be set to logic 0 since all AC parametric testing done with the 5V signaling conditions.
MDATA[26]	FIRM	Reserved for use by Switch Processor firmware.
MDATA[25:22]	CHIPID[3:0]	These bits determine the PCI memory base address at initialization time if the PCIRUN configuration bit is high. In this case, the CHIPID[3:0] inputs are zero-extended to 8 bits and loaded into the most significant byte of the Memory Base Address register in the PCI configuration register space.
MDATA[21:16]	RTCDIV[5:0]	Real time clock divider: selects the divide ratio used for the internal real-time clock prescaler. This field must be set numerically equal to the frequency, in megahertz, of the clock supplied on the SYSCLK input.
MDATA[15:14]	MXSEL[1:0]	These inputs select the row/column multiplexing used for EDO DRAM devices. MXSEL: 00 — Column Address Bits: 8 — DRAM Configurations Supported: 64K x N & 128K x N MXSEL: 01 — Column Address Bits: 9 — DRAM Configurations Supported: 256K x N & 512K x N MXSEL: 10 — Column Address Bits: 10 — DRAM Configurations Supported: 1024K x N & 2048K x N MXSEL: 11 — Column Address Bits: 11 — DRAM Configurations Supported: 4 Meg x N & 8 Meg x N
MDATA[13]	MSLO	The MSLO bit extends read and write cycles to accommodate slower local memory devices. If MSLO is high, memory accesses will be to 80ns DRAM. If MSLO is low, 60ns DRAM is expected. The PM3350 is intended to be used with 60ns EDO DRAM; hence, MSLO must be a logic 0. The access time for ROM is always 150ns, respectively, regardless of the state of the MSLO bit.
MDATA[12]	MDCAS	This bit identifies the type of DRAM connected to the memory interface. If MDCAS is high, the memory interface will generate control signals for 2-CAS DRAMs; otherwise, it generates signals for single CAS DRAMs.
MDATA[11:9]	MTYPE3[2:0]	Indicates the type of memory connected to the MCS[3]* output: MTYPE3[2:0] — Selected memory type 000 — Reserved 001 — Reserved 010 — Reserved 011 — Reserved 100 — Reserved 101 — 200ns (E)EPROM 110 — 60ns EDO DRAM 111 — Reserved
MDATA[8:6]	MTYPE2[2:0]	Indicates the memory type associated with MCS[2]*. The encoding is the same as for MTYPE3[2:0].

MDATA[5:3]	MTYPE1[2:0]	Indicates the memory type associated with MCS[1]*. The encoding is the same as for MTYPE3[2:0].
MDATA[2:0]	MTYPE0[2:0]	Indicates the memory type associated with MCS[0]*. The encoding is the same as for MTYPE3[2:0].

After the hardware configuration information has been latched from the data bus, it is loaded into the DCONFIG and MCONFIG registers. The lower 16 bits of the configuration word (i.e., bits 0 through 15, latched from MDATA[15:0]) are loaded into the MCONFIG register, with MDATA[0] being loaded into the LSB of MCONFIG. The upper 16 bits (i.e., corresponding to MDATA[31:16]) are loaded into the DCONFIG register in a similar fashion.

System Bootstrap Image

The ELAN 8x10 is designed to self-initialize upon power-up, using information and operating firmware supplied as a pre-determined image (referred to as the *boot image*) in external memory (typically, EPROM or EEPROM). The boot image may be located anywhere in the 16 MB address space, but must start on a 64 kB boundary. The ELAN 8x10 expects the boot image to be formatted in a predefined manner, as described below. The boot image consists of a boot header and a set of boot data blocks.

Boot Header

The boot image is distinguished by a special 32-bit signature followed by a predefined configuration header. The ELAN 8x10 will, therefore, perform some basic initialization indicated by the hardware configuration word loaded from the data bus after reset, and then begin scanning the entire memory space at 64 kB boundaries for the boot image signature. It expects to find the four bytes of the signature aligned on four consecutive 32-bit boundaries, as indicated in the following table:

Offset from 64kB Boundary	Expected Contents (hex)
+0	XXXXXXC8
+4	XXXXXXA8
+8	XXXXXX37
+12	XXXXXX59

The use of a signature to locate the boot image, rather than an explicit address, implies that it is not necessary to indicate the exact location of the configuration image to the ELAN 8x10. Instead, the boot image may be located anywhere throughout the 16 MB address space. In addition, a boot image need not even be supplied using an EPROM or EEPROM; it may also be downloaded to RAM by an external device or host processor.

The ELAN 8x10 expects to find executable firmware code within the boot image that will perform the actual initialization process. However, the boot image is only 8 bits wide, and thus the firmware code supplied within it cannot be directly executed by the ELAN 8x10 Switch Processor, which uses 32-bit instructions located on 32-bit boundaries. It is thus necessary for the ELAN 8x10 to copy the boot image to a block of RAM that is set aside for the purpose, and to convert the 8-bit boot image to a 32-bit version so that the boot firmware code can be directly executed.

When a proper boot image signature is found, therefore, the ELAN 8x10 will automatically go on to read a preformatted header within the boot image. This header should supply information required to copy the boot image to a pre-allocated block of RAM, and is formatted as follows:

Field Size, Bytes	Byte Offsets from Start	Mnemonic	Description
1	+16	HDRFLAGS	Boot image processing flags
3	+20,+24,+28	CPYTARGET	Target RAM block to copy 8-bit boot image to
2	+32,+36	CPYSIZE	Amount of data to copy in 64-byte blocks
3	+40,+44,+48	CPYFROM	Source of copy data within boot image
3	+52,+56,+60	BOOTSTART	Starting address of bootstrap firmware in target RAM block (valid after copy completed)
4	+64,+68,+72,+76	CHECKSUM	32-bit checksum, computed over entire boot image (including checksum field)
4	+80,+84,+88,+92	SPACER	Must be set to 0x00000000 hexadecimal

The HDRFLAGS field supplies some control bits that determine how the ELAN 8x10 will handle the boot image, and is formatted as follows:

7	6	5	4 ... 0
EightBit	CpyInt	JmpInt	reserved

EightBit:

If set, indicates that the boot image is formatted as an eight-bit-wide memory block (following the general format that has already been presented); otherwise, indicates a 32-bit boot image in a special format. This bit is intended for factory use only, and should always be set by the customer.

CpyInt:

If this bit is set, the contents of the boot image should be copied to the

internal instruction RAM of the Switch Processor rather than a block of external memory. This bit should be set only if the EightBit flag is also set (i.e., for a standard 8-bit-wide boot image).

JmpInt:

The JmpInt bit signifies, if set, that the BOOTSTART address indicates an address present in the instruction RAM of the Switch Processor, rather than an address in a block of external memory. This bit should be set only if the EightBit flag is also set (i.e., for a standard 8-bit-wide boot image).

In general, the HDRFLAGS field should be set to 0x80 hex, i.e., indicating an 8-bit-wide boot image with no special copy or branch options.

Note that the boot image header above is described as it would appear in the 8-bit image contained within an EPROM or EEPROM that is connected to the least-significant byte lane of the memory data bus. Each byte within the EPROM or EEPROM, therefore, will start on a 32-bit boundary and occupy the least-significant 8 bits of a memory word; the upper 24 bits of the word will be ignored by the ELAN 8x10. The ELAN 8x10 will copy data bytes from the EPROM or EEPROM boot image to consecutive byte addresses in a block of 32-bit wide RAM, thereby converting the 8-bit boot image to a 32-bit boot image. An alternative view of the boot header, giving the components of the header as they would appear after the boot image has been copied to 32-bit RAM, is given below:

0x59	0x37	0xa8	0xc8	0
CPYTARGET[23:0]			HDRFLAGS[7:0]	4
CPYFROM[15:0]		CPYSIZE[15:0]		8
BOOTSTART[23:0]			CPYFROM[23:16]	12
CHECKSUM[31:0]				16
SPACER[31:0]				20

The CPYTARGET, CPYSIZE and CPYFROM fields of the header define a block transfer that must be performed from the boot image to an area of RAM in order to convert the 8-bit boot image to a 32-bit image. After the copy and conversion has been done, the BOOTSTART field denotes a 24-bit address from which the ELAN 8x10 Switch Processor will begin executing code. The code at this location (presumably within the copied 32-bit boot image) is completely responsible for initializing the ELAN 8x10 system and environment, and for initiating normal operation.

The CHECKSUM field contains a 32-bit checksum computed over the entire boot image. The bootstrap firmware will recompute this checksum and compare it with the value in the CHECKSUM field. If a mismatch occurs, the ELAN 8x10 will consider the boot image as invalid, and will terminate the system initialization and startup process and report an error.

Boot Data

In addition to the signature, header and bootstrap firmware code, the boot image is expected to contain configuration information such as the sizes of memory buffer pools, buffer limits defined on a per-port basis, predetermined MAC addresses to be placed in the routing tables, the location of external ELAN 8x10s, the MAC address and IP address assigned to this ELAN 8x10, and so on. The boot image must also supply the operating firmware that is required by the ELAN 8x10 for packet switching and management. The general layout of the boot image is as shown below (note that the memory addresses increase downwards):

0	Boot Image Signature
	Boot Image Header
	Boot Tag Space
	Bootstrap Firmware Code
	Auxiliary Configuration Information
	Auxiliary Firmware Code
32 Kbytes	Master Operating Code
	RTOS Firmware Code (optional)
	UDP/IP Stack Firmware Code (optional)
256 Kbytes	SNMP Agent/MIB Firmware Code (optional)

As shown, the basic bootstrap code and the master switch operating firmware occupy 32 Kbytes of EPROM/EEPROM space; if an SNMP agent is required (with the associated UDP/IP stack and RTOS firmware), then the boot image size requirements rise to 256 Kbytes. Contact the factory for more information on the boot image, and the means of creating one.

The bootstrap firmware code performs a brief self-test, testing external memory and verifying the boot image checksum. The status of each stage of the self-test is output as writes of binary codes to the (configurable) memory location at which the optional LED register may be located. After the self-test completes, the ELAN 8x10 uses the information read from the boot image to set up the fixed and dynamic data structures in external and internal RAM, and then initiates normal operation.

In a multiple-ELAN 8x10 system, a single ELAN 8x10 may be designated as a master and possess a boot image in an EPROM or EEPROM. This ELAN 8x10 is then responsible for initializing and configuring all the other ELAN 8x10s in the system via the expansion port interface. This is facilitated by the RISCRUN configuration bit supplied on the memory data buses of all of the ELAN 8x10s in the system. In this application, the master ELAN 8x10 should have its RISCRUN bit pulled HIGH during reset, and all of the other slaves should have the corresponding bits pulled LOW, ensuring that they enter standby mode The master ELAN 8x10 can then configure itself, download boot information to the RAM interfaced to the remaining ELAN 8x10s, and finally enable all of the slave ELAN 8x10s to start running. The slave devices can then initialize themselves using the downloaded information.

Configuration Parameters

This section describes the various configuration parameters that can be adjusted by the system implementer to create a boot image for various system types and options. The configuration parameters are primarily located in a header file that is read as part of the bootstrap image creation process.

This section is TBD.

Stand-Alone System Boot

This section describes how a stand-alone ELAN 8x10 system (i.e., one containing only one device, or one where every ELAN 8x10 device possesses its own EPROM/ EEPROM that provides a boot image) initializes itself.

This section is TBD.

Master/Slave System Boot

The boot-up process implemented in a master/slave ELAN 8x10 system, consisting of a single master ELAN 8x10 device and one or more slave devices, is described here. In such a system, only the master device possesses an EPROM or EEPROM containing the primary boot image; the slaves receive their boot images in the form of downloads by the master device to predetermined areas of slave device external RAM.

This section is TBD

Host-Controlled System Boot

In a host-controlled system, it is expected that all of the ELAN 8x10 devices are configured as slaves, i.e., they do not possess EPROMs or EEPROMs containing boot images that permit them to self-configure at power-up. Instead, they enter a halt state after system reset and wait for the host (i.e., system master processor) to download the required boot image and enable them to begin executing it.

This section is TBD.

Self Test and Error Reporting

The bootstrap firmware code is expected to implement any required power-on self-test (POST) functions that are required by the system of which the ELAN 8x10 is a part. If any of the POST routines detects an error, it is expected to halt the bootstrap process and write a special code to the (optional) LED register that is mapped into the ELAN 8x10 memory address space. If the LED register is implemented, then the failure indication can be obtained for diagnostic purposes.

Currently, two primary types of self-test routines are implemented:

1. A checksum is computed over the complete boot image and compared to the pre-computed checksum in the boot image header. If a mismatch is detected, then the boot image is considered to be corrupted, and cannot be used for system initialization.
2. A destructive RAM test is performed over the entire RAM space with the exception of the space occupied by the boot image itself. The RAM test is quite simple, and consists of writing a known pseudo-random value to each location in the RAM and then reading the data back. If the data read is not equal to that written, then the RAM is considered to be defective, and the system cannot begin operation. (The RAM self-test is not intended to be an exhaustive device test aimed at unconditionally detecting a faulty RAM, but merely a fast and simple test for a gross go/no-go check.)

Additional self-test routines will be implemented in the bootstrap firmware code as developed.

In addition to the self-test functions performed upon system start-up, the ELAN 8x10 operating firmware also performs numerous checks of its internal state during normal system operation. If an unrecoverable error is detected, the ELAN 8x10 will output a status code to the LED register, and then attempt to restart itself (and possibly the entire system) via the internal watchdog reset facility. If the internal watchdog reset output (as driven onto the ERST* pin) is connected to the global system reset, then the ELAN 8x10 will reset the entire system; otherwise, the ERST* pin should be monitored by an external system master to determine when the ELAN 8x10 is halted due to some fatal error, and must be reset in order to continue.

In general, LEDs 6 and 7 (i.e., the most significant bits of the LED register) are intended to be used to provide a general failure indication; LED 5 indicates whether the failure occurred at self-test and initialization time, or whether the failure occurred during normal operation; and the rest of the LEDs supply a diagnostic code that can be used to identify the cause of the failure. The codes written out to the LED register upon detection of any device failure are defined below:

Failure	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
Boot Checksum	Blinking	On	Off	Off	Off	Off	Off	On
RAM test	Blinking	On	Off	Off	Off	Off	On	Off
Operational error	Blinking	On	On	Code[4]	Code[3]	Code[2]	Code[1]	Code[0]

If a failure is detected at any time that causes the ELAN 8x10 to halt normal operation, the most significant LED (connected to bit 7 of the LED register) will be made to blink on and off at a 0.5 second rate. The next-most-significant LED (i.e., bit 6 of the LED register) will be turned ON by writing a zero to this bit position of the LED register; this serves as a global failure indication, and may alternatively be polled by system hardware to determine whether a failure has occurred. LED number 5 indicates whether the failure occurred during system boot-up or normal operation: if it is lit, the system encountered an unrecoverable error during operation. The remaining five LEDs are used to signal an error-specific code that can be used for diagnostic purposes. The codes signaling errors during operation are TBD.

Data Structures

It is assumed that an earlier bootstrap initialization and configuration phase will have set up some predefined data structures in ELAN 8x10 local memory. These are the Switch Processor operating environment (stacks, memory pool, local variables, etc.), Data Descriptors, fixed-length Packet Buffers, the Port Descriptor Table (with the associated per-port section of the Management Information Base), the MAC address hash table (which also contains the per-host section of the Management Information Base), and data associated with the IEEE 802.1d spanning tree bridge configuration algorithm.

The data structures, their function, and the operations performed on them are described below. In addition, the memory requirements for each type of structure, as well as the general memory map expected by the Switch Processor operating firmware, are provided in this section.

Switch Processor Operating Environment

The Switch Processor requires a small operating environment (i.e., data structures and variables) for performing basic packet switching functions. This environment is set up in the external RAM during the bootstrap initialization and configuration phase. The entire operating environment occupies about 40 Kbytes of space, and must be located

starting at address 0x000000 hex in the external memory. Note that the operating environment excludes the space used by Ethernet frame buffers and the queuing structures used to track them, and also does not include the code, data or stack spaces that are required by the (optional) real-time OS, UDP/IP stack, or SNMP agent firmware.

The general layout of the operating environment is as follows (note that the memory addresses increase downwards):

0x000000 0x001FFF	Reserved for switching firmware
0x002000 0x00207f	Frequently used variables and parameters
0x002080 0x0020ff	Register save space for switching code
0x002100 0x00237f	Local Port Descriptors
0x002380 0x0023ff	Expansion Port Descriptors
0x002400 0x00247f	Dispatch Table
0x002480 0x0025ff	Miscellaneous variables and tables
0x002600 0x0029ff	Port Descriptor Error Counters
0x002a00 0x00a9ff	Hash Pointer Array

The components of the operating environment are as follows:

1. The first 8192 bytes of space are reserved for holding switching firmware code. This space is intended to serve as a backup for the internal 8192 byte instruction RAM present in the ELAN 8x10.

2. The next 128 bytes are used to hold various frequently referenced variables, such as the exception masks, broadcast counters, etc. These variables normally reside in the Switch Processor data cache, and hence their memory image may be inconsistent until the cache is flushed to memory.

3. A block of 128 bytes is reserved for use by the switching firmware as a register save space during interrupts.

4. A set of eight Local Port Descriptors is placed in the next 640 bytes. Note that only the first 512 bytes of this space is actually used; the remainder is reserved for future applications. The Local Port Descriptors usually reside in the Switch Processor data cache, and this region of memory will thus contain out-of-date values until the data cache is flushed. The Local Port Descriptors are described in further detail below.

5. A set of eight Expansion Port Descriptors occupies another 128 bytes. As in the case of the Local Port Descriptors, the Expansion Port Descriptors are normally cached and the memory region will not be updated until a data cache flush. The Expansion Port Descriptors are discussed in more detail below.

6. A dispatch table, used to hold addresses of handlers for special situations or for error recovery purposes. A block of 128 bytes is reserved for the dispatch table. Note that the dispatch table is not cached.

7. The next 384 bytes are reserved for miscellaneous variables utilized on an infrequent basis by the switching firmware. These variables include tables of constants, such as the pseudo-random number generator table used by the collision handling firmware. These variables are not cached.

8. 1024 bytes are reserved for a set of 8 local port error counter blocks to hold error and collision statistics for each of the local (MAC) ports. Each counter block requires 96 bytes of storage. Note that only the first 768 bytes of this space is actually used; the remainder is reserved for future applications. These counters are also described in more detail below.

9. The next 32768 bytes are reserved for the Hash Pointer Array. This array forms the base of the MAC address hash table; each 4-byte entry in the array contains a pointer to a chain of hash buckets that hold the actual per-MAC information required for packet switching. The entire array is cleared to zero (NULL) during

the initialization process. The Hash Pointer Array contains a total of 8192 pointers.

The first 1024 bytes of the operating environment are normally cached by the Switch Processor in the data cache. As the data cache uses a write-back policy, the actual memory locations corresponding to the cached structures may be out-of-date (i.e., not reflect the most recent information written to the locations by the firmware). For instance, the per-port SNMP counters contained within the Local Port Descriptors are cached, and hence the memory images of the per-port counters may not be up-to-date. Thus the Switch Processor must be forced to flush the data cache to update the external memory locations prior to reading them from an external CPU via the PCI bus. This can be accomplished via the messaging interface described further within this section.

It is the responsibility of the bootstrap and initialization firmware contained within the boot image to set up and initialize the entire operating environment described above.

Variables and Tables in Operating Environment

In addition to the major data structures (i.e., the register save space, the local and expansion port descriptors, the port descriptor error counters, and the hash pointer array), the Switch Processor operating environment contains a number of variables and tables used during normal operation. Some of these memory locations (those located between addresses 0x002000 and 0x0023ff in the memory map above) are normally expected to be cached in the Switch Processor data cache, while the remainder are never loaded into the data cache by the operating firmware. This section describes these variables and their expected values.

The remainder of This section is TBD.

Packet Buffers

Ethernet packets are stored in the external RAM by the ELAN 8x10 chip in small, fixed length *packet buffers*; multiple packet buffers are chained in a linked list to hold a complete Ethernet packet. The size of the packet buffers is determined at configuration time, and may range from 64 to 240 bytes; the default is 80 bytes.

Each packet buffer contains an 8-byte header holding various control information, and from 0 to at most (N - 8) bytes of payload, comprising Ethernet frame data. (In this context 'N' denotes the total size of each packet buffer: the default 80-byte packet buffers will contain at most 72 bytes of payload.) The Ethernet data stored in the packet buffers includes the Ethernet header and CRC fields.

A MAC channel byte-swap control bit is implemented on a channel-by-channel basis in the LWCTRL device control register (see register descriptions below). If the byte swap

control is set to the default of zero, indicating no byte swap, the packet buffers have the following format (where 'N' is the total size of the packet buffer in bytes):

31 24	23 16	15 8	7 0	Byte Offset
Size	NextPB			0
LastSize	RefCount	unused, reserved		4
PayloadByte0	PayloadByte1	PayloadByte2	PayloadByte3	8
PayloadByte4	PayloadByte5	PayloadByte6	PayloadByte7	12
....				....
PayloadByte(n-4)	PayloadByte(n-3)	PayloadByte(n-2)	PayloadByte(n-1)	(N-4)

If the MAC channel is set up to swap the incoming frame bytes, then the payload field of the packet buffers will be byte-swapped, but the headers will be left unchanged, as shown below:

31 24	23 16	15 8	7 0	Byte Offset
Size	NextPB			0
LastSize	RefCount	unused, reserved		4
PayloadByte3	PayloadByte2	PayloadByte1	PayloadByte1	8
PayloadByte7	PayloadByte6	PayloadByte5	PayloadByte4	12
....				....
PayloadByte(n-1)	PayloadByte(n-2)	PayloadByte(n-3)	PayloadByte(n-4)	(N-4)

NextPB:

24-bit pointer to next packet buffer in linked-list of packet buffers constituting a frame. If no next packet buffer exists (i.e., this is the tail of the linked-list), then this field is NULL (all-zeros).

Size:

8-bit size of packet buffer payload (excludes the 8-byte header), in bytes.

RefCount:

8-bit reference count associated with packet: gives the number of queues that are pointing to this packet buffer.

LastSize:

8-bit count of total number of valid bytes (including the 8-byte header) in the last packet buffer in the linked-list of buffers. Only valid if this is the first packet buffer in the linked-list, and there is more than one packet buffer in the linked-list.

PayloadByte0 - PayloadByte(n-4):

8-bit packet buffer payload bytes: contains Ethernet frame data for the frame contained within the linked-list of packet buffers.

The NextPB field in the packet buffer header contains a pointer to the next packet buffer in the chain of buffers that holds the entire Ethernet frame. If this is the last (or only) buffer in the chain, then this field is set to zero to indicate a NULL pointer. The Size field holds the number of valid payload bytes (i.e., excluding the 8 packet buffer header bytes) in the packet buffer payload field; for an 80-byte packet buffer, the value in this byte can range between 0 and 72. A value of zero indicates a completely empty packet buffer with no data, which will simply be skipped over by the hardware or firmware with no ill effects.

The RefCount field holds a reference count indicating the number of ports or devices to which the packet buffer contents must be transmitted, and is used to determine when the packet buffer may be freed. The reference count is normally 1 for unicast frames, and equals the sum of the number of destination ports and devices for broadcasts; it is only valid for the last packet buffer in a chain (i.e., when the next packet buffer pointer is NULL), and is ignored in other buffers.

The LastSize field in the header holds the total number of valid bytes, including the header, in the last packet buffer in a linked list of buffers. It should be placed in the first packet buffer in the chain, and is principally used to optimize processing of packet buffers by the hardware. This field is ignored if there is only one packet buffer in the chain, or if the buffer under consideration is not the first one in the chain.

The remaining bytes in the packet buffer contain the payload, consisting of Ethernet frame bytes received by one of the MAC channels of the ELAN 8x10 (or created via software). The bytes are placed into the packet buffer in the order depicted above, depending on the setting of the byte swap hardware configuration bit. Note that 'PayloadByte0' in the above diagrams refers to the first byte received from the medium for the payload of the given packet buffer; in the case of the first packet buffer in a chain, this byte will contain the LSB of the 48-bit MAC destination address in the Ethernet frame header.

Packet buffer chains may also be used by the various chips in the system to contain message data that is to be exchanged between devices interfaced to the PCI bus. If a packet buffer holds message data rather than Ethernet frame data, then the format of the header is unchanged, but the payload field is formatted in a message-specific manner.

Data contained within packet buffers is never cached by the Switch Processor, and hence the contents of any packet buffer may be read at any time via the PCI expansion port. In normal operation, packet buffers are primarily manipulated by the DMA hardware; the Switch Processor usually writes only the RefCount and LastSize fields.

Data Descriptors

Ethernet packets being buffered within the ELAN 8x10 system are tracked by means of 16-byte *data descriptors*. Data descriptors form the primary queuing element in the ELAN 8x10 device: they are used to form receive, transmit and expansion port queues, as well as to hold message information and special packet handling queues. All data descriptors have the same general format, and contain pointers to the first and last packet buffer of a packet buffer chain holding an Ethernet packet, flags fields that indicate the processing to be performed on the packet, and other information associated with the packet.

Data descriptors are formatted as follows:

31 24	23 16	15 8	7 0	Byte Offset
Flags	NextDD			0
NumPBs	FirstPB			4
SrcPort	LastPB			8
SrcChip	HashBkt			12

NextDD:
24-bit pointer to next data descriptor in linked-list of data descriptors forming queue.

Flags:
8-bit data descriptor flags field, formatted as below.

FirstPB:
24-bit pointer to first packet buffer in linked-list of packet buffers containing frame data. If NULL (zero), no packet buffer chain exists for this data descriptor; this is only valid for special message data descriptors.

NumPBs:
8-bit count of packet buffers in chain pointed to by FirstPB.

LastPB:
24-bit pointer to last packet buffer in linked-list of packet buffers pointed to by FirstPB.

SrcPort:
8-bit source port index within device: indicates the MAC channel upon which the packet was received. For the ELAN 8x10, this ranges from 0 through 7.

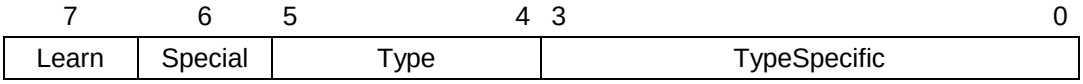
HashBkt:
24-bit pointer to source or destination hash bucket.

SrcChip:

8-bit index of source device that received packet. For the ELAN 8x10 system, this must range from 0 through 7.

The NextDD field in the data descriptor contains a pointer to the next data descriptor in a chain of descriptors that constitutes a queue of Ethernet frames or messages. If this is the last (or only) element in the queue or chain, then this field is set to zero to indicate a NULL pointer.

The Flags field holds a set of flag bits and bitfields that provide type and control information for the data descriptor, and is formatted as follows:



Learn:

If set, indicates that the source address (SA) field of the Ethernet frame pointed to by this descriptor contains a new MAC address that must be learned by the device. Only valid for Ethernet frames received by the ELAN 8x10 from the PCI expansion port; should not be set if the data descriptor points to a message.

Special:

If set, indicates that the data contained within the packet buffer chain pointed to by this data descriptor requires special processing by the firmware, and should not be treated as a normal Ethernet frame. This bit is typically set to indicate a message that is being passed between devices on the PCI expansion port.

Type:

This 2-bit field contains the type of frame pointed to by the data descriptor FirstPB field, and is interpreted as follows:

Type	Interpretation
00	Unicast frame from remote device
01	Unicast frame from local MAC channel
10	Broadcast frame from remote device
11	Broadcast frame from local MAC channel

The above definitions of the Type field are only valid if the Special bit is clear, indicating that this data descriptor points to a normal Ethernet frame.

TypeSpecific:

The 4-bit TypeSpecific field is used to hold information specific to the type of

information carried by the data descriptor, or the purpose for which the data descriptor is being used. In the case of message descriptors, this field is used carry a code that identifies the type of message being sent from one chip to another. For normal data descriptors that point to frame data being processed, this field is usually set to zero. The mark-and-sweep algorithm used to maintain the transmit queues will also utilize this field to determine when a particular transmit queue has been excessively blocked.

The FirstPB and LastPB fields contain the 24-bit addresses of the head and tail, respectively, of the linked-list of packet buffers that are associated with this data descriptor. If only one packet buffer is present in the linked-list, then the FirstPB and LastPB pointers contain the same value. It is an error for the FirstPB field to be zero in any valid data descriptor that points to an Ethernet frame; however, message data descriptors (i.e., those used for exchanging messages between ELAN 8x10 devices) may optionally have the FirstPB field set to zero, if the entire content of the message can be placed in unused fields of the data descriptor. The NumPBs field contains the count of the number of packet buffers contained within the linked-list of packet buffers, and may range between 1 and 255.

The SrcPort and SrcChip fields indicate the source port index and the source device index, respectively, on which the Ethernet frame entered the system. The values in the SrcPort field are directly derived from the hardware indices assigned to the various physical MAC channels of the ELAN 8x10; the value placed in the SrcChip field by any ELAN 8x10 device is assigned as a system parameter during the device boot-up and initialization process, using data obtained from the boot image. If the Learn bit is set in the data descriptor Flags field, then both of these fields are used to associate the source MAC address within the Ethernet frame with a particular device and physical port.

The HashBkt field holds the 24-bit local memory address of the address hash bucket associated with either the source or destination MAC address in the Ethernet frame header. If the Ethernet frame is to be broadcast (or the Learn bit is set, and the frame must be flooded), the HashBkt field holds the memory address of the hash bucket corresponding to the source MAC address in the source device's address space; if, on the other hand, the frame is a unicast, then the HashBkt field contains the memory address of the hash bucket for the destination MAC address in the destination device's address space. (The distinction between the source and destination device address spaces only applies to frame transfers across the PCI expansion bus; if the frame is being switched locally, or there is only one ELAN 8x10 device in the system, then the source and destination address spaces are the same.) The HashBkt field has a dual purpose: it is used to learn the memory address of a source hash bucket during broadcasts and floods, and is also used to indicate a target destination hash bucket to use in switching unicast frames once address learning is complete. More details on the use of the HashBkt field in various situations will be provided in subsequent sections.

Note that the HashBkt field is not used when the data descriptor does not point to an Ethernet frame.

A slightly different format is used for data descriptors when they are being transferred between internal firmware routines within the ELAN 8x10 during frame reception on the eight local MAC channels. This format is strictly transient, and is never visible when the data descriptor is presented to an external entity or packet driver firmware, but may be observed if the contents of the work queue (see below) are inspected directly via the PCI bus interface. The format is as follows:

31 24	23 16	15 8	7 0	Byte Offset
Flags	NextDD			0
NumPBs	FirstPB			4
SrcPort	LastPB			8
DstHBIndx		SrcHBIndx		12

NextDD:
24-bit pointer to next data descriptor in linked-list of data descriptors forming work queue.

Flags:
8-bit data descriptor flags field, formatted as below.

FirstPB:
24-bit pointer to first packet buffer in linked-list of packet buffers containing frame data.

NumPBs:
8-bit count of packet buffers in chain pointed to by FirstPB.

LastPB:
24-bit pointer to last packet buffer in linked-list of packet buffers pointed to by FirstPB.

SrcPort:
8-bit source port index within device: indicates the MAC channel upon which the packet was received. For the ELAN 8x10, this ranges from 0 through 7.

SrcHBIndx:
16-bit encoded index of source hash bucket found by address table lookup hardware.

DstHBIndx:

16-bit encoded index of destination hash bucket found by address table lookup hardware.

The NextDD, FirstPB, NumPBs, LastPB and SrcPort fields all correspond to the similarly named fields in normal data descriptors, and perform the same functions. The Flags field, however, is formatted in a slightly different manner, as shown below:

7	6	5	4	3	2	1	0
0	0	0	1	ARPT	Dribble	DMiss	SMiss

SMiss:

If set, indicates that the SrcHBIndx field is invalid (the address table lookup hardware did not find a match for the source MAC address).

DMiss:

If set, indicates that the DstHBIndx field is invalid (the address table lookup hardware did not find a match for the destination MAC address).

Dribble:

Indicates a dribble bit error during the reception of the frame (i.e., additional bits were received beyond the end of the frame, but the CRC was found to be valid).

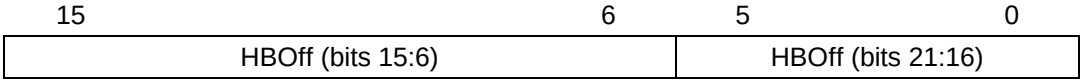
ARPT:

If set, denotes that the 16-bit EtherType field within the frame contains the type code assigned to Address Resolution Protocol (ARP) frames (0x0806 hexadecimal).

The SMiss and DMiss bits signify, if set, that the address lookup failed for the source and destination MAC addresses, respectively. This notification is supplied by the address table lookup hardware after it attempts to search the hash table for the 48-bit MAC addresses in the received frame. The Dribble bit is used to signal the switching firmware that a dribble error was detected during the reception of the frame, and the suitable dribble error counter should be updated. Finally, the ARPT bit is set whenever the EtherType field of the received frame (corresponding to the IEEE 802.3 Length field) equals the value assigned to ARP frames, or 0x0806 hex.

The SrcHBIndx and DstHBIndx fields contain the encoded 16-bit indices of the 24-bit source and destination hash buckets, respectively, that are returned by the address table lookup hardware after a search for the MAC addresses within the Ethernet frame. (If either of the searches fail, then the corresponding field contains invalid information, and should not be used.) The encoding is performed by first subtracting the 24-bit base address of the start of the hash bucket array to produce an offset into the array (rather than the absolute address returned by the address table lookup hardware), and then

shuffling the bits in order to compact the 24-bit offset into a 16-bit quantity. The resulting format is as follows:



HBOff:

24-bit offset of hash bucket or forwarding tag found during search of address table for given MAC address.

The compaction of the 24-bit offset into 16 bits takes advantage of the fact that all hash buckets (and consequently forwarding tags) are expected to be aligned on 64-byte boundaries, and hence the lower 6 bits of the offset will always be zero; also, the hash bucket array cannot span memory banks, and hence the uppermost 2 bits of the offset will also be zero. The bit field shuffling is done to minimize the computational requirements of the index generation. When the switching firmware removes the data descriptor from the work queue, it will immediately convert the encoded hash bucket indices into their 24-bit absolute address equivalents (assuming that the SMiss and/or DMiss bits are not set) and thereafter use only the 24-bit addresses, discarding the encoded forms.

As previously noted, the above data descriptor format is only used internally during a normally invisible portion of the firmware processing cycle; hence it is irrelevant to all but the most specialized applications.

Data descriptors are never cached by the Switch Processor, and hence the contents of any data descriptor that is not actually being processed by the Switch Processor may be read at any time via the PCI expansion port. The Switch Processor is usually the only entity that allocates, creates, manipulates and de-allocates data descriptors. (An exception is during frame transfer across the PCI bus: in this case, the DMA hardware is responsible for copying the contents of remote data descriptors to local memory.)

Local Port Descriptor Tables

A set of data structures, collectively referred to as the *Local Port Descriptor Table,* is associated with the eight MAC interfaces within the ELAN 8x10. The local port descriptor table contains three types of structures: a local port descriptor, which contains various control/status and packet statistics fields, and is updated whenever a packet is received or transmitted by that MAC interface; a chain of data descriptors (attached to the local port descriptor), referred to as the *transmit queue*, that holds Ethernet frames waiting to be transmitted out the MAC interface; and a set of packet error statistics fields, known as the *port descriptor error counters*, which tracks various error counts that are updated on an infrequent basis during normal operation. A separate copy of these three data structures is maintained for each of the eight MAC channel ports that are implemented by the ELAN 8x10. All of these data structures are

under the sole control of the Switch Processor, and never modified or read by the rest of the ELAN 8x10 hardware.

Local Port Descriptor Structure

Local port descriptors are associated on a one-to-one basis with the eight physical MAC channels (assigned indices from 0 through 7), with the port descriptor at the lowest address being assigned to MAC channel index zero, and so on. (The memory map already given shows the location of the local port descriptors in the external memory.) Each local port descriptor structure occupies 64 bytes, and is formatted as follows:

31 24	23 16	15 8	7 0	Byte Offset
NumCollide	FirstTDD			0
PortNum	LastTDD			4
MulticastMask		MaxBuffers		8
PortFlags	Reserved	BackoffMask		12
TXFrames				16
TXOctets				20
TXDeferred				24
Reserved				28
RXUcastFrames				32
RXUcastOctets				36
RXFrames64				40
RXFrames65-127				44
RXFrames128-255				48
RXFrames256-511				52
RXFrames512-1023				56
RXFrames1024-1518				60

FirstTXDD:

24-bit pointer to first data descriptor in transmit queue for this MAC channel.

NumCollide:

8-bit count of collisions experienced so far while attempting to transmit frame at head of transmit queue for this MAC channel; cleared to zero before starting transmit for each frame.

LastTXDD:

24-bit pointer to last data descriptor in transmit queue for this MAC channel.

PortNum:

8-bit logical port index assigned to the physical MAC channel with which this port descriptor is associated.

MaxBuffers:

16-bit count of packet buffers that remain to be allocated to hold incoming received frames; flow control or packet discard will be initiated when this count goes to zero.

MulticastMask:

16-bit mask used to restrict broadcasts and multicasts received on this port to a subset of the ports on this ELAN 8x10 and a subset of the devices in the system.

BackoffMask:

16-bit mask used to limit the range of collision backoff values generated by the truncated binary exponential backoff algorithm for this port. Note that only the lower 10 bits are significant; the upper 6 bits of this field should always be set to zero.

PortFlags:

8-bit per-port control flags.

TXFrames:

32-bit count of valid frames transmitted on this port.

TXOctets:

32-bit count of valid bytes transmitted on this port.

TXDeferred:

32-bit count of transmitted frames that experienced carrier deference during the transmission process.

RXUcastFrames:

32-bit count of valid unicast frames received on this port.

RXUcastOctets:

32-bit count of valid bytes from unicast frames received on this port.

RXFrames64:

32-bit count of frames received on this port that were 64 bytes in size.

RXFrames65-127:

32-bit count of frames received on this port that ranged between 65 and 127 bytes in size, inclusive.

RXFrames128-255:

32-bit count of frames received on this port that ranged between 128 and 255 bytes in size, inclusive.

RXFrames256-511:

32-bit count of frames received on this port that ranged between 256 and 511 bytes in size, inclusive.

RXFrames512-1023:

32-bit count of frames received on this port that ranged between 512 and 1023 bytes in size, inclusive.

RXFrames1024-1518:

32-bit count of frames received on this port that ranged between 1024 and 1518 bytes in size, inclusive.

The FirstDD and the LastDD fields serve to control the transmit queue for the MAC channel associated with this local port descriptor: the FirstDD contains the 24-bit memory address of the data descriptor at the head of the queue, while the LastDD field points to the data descriptor at the tail of the queue. If the queue is empty, the FirstDD and LastDD fields are set to NULL; otherwise, the fields define a linked-list of data descriptors that in turn point to the chains of packet buffers containing the Ethernet frame data that must be transmitted out this physical port. Initially, the FIrstDD and LastDD fields are set to NULL (i.e., representing an empty transmit queue.) As frames are queued for transmit, these are changed to point to the respective data descriptors; as frames are transmitted, they are updated, until finally they return to holding NULL pointers when no more frames are present on the transmit queue. It is illegal for anything other than valid Ethernet frames to be queued on the transmit queues maintained by the local port descriptors.

The NumCollide field is used to track the number of consecutive collisions encountered when attempting to transmit a given frame. It is cleared to zero prior to starting the transmit of every frame. If a collision terminates the frame transmission attempt, this field is incremented by one; if the count of collisions for this frame exceeds the pre-defined maximum (generally, a default value of 16, according to the IEEE 802.3 standard), then the frame is discarded and an error is reported. The value of the NumCollide field is also used in generating the backoff counter value according to the IEEE 802.3 collision backoff algorithm.

The PortNum field contains the logical port number assigned to the physical MAC port corresponding to this local port descriptor. This field must be set to the physical index of the MAC port at startup time. The SrcPort field in data descriptors pointing to frames received on this MAC port is initialized from the PortNum field.

MaxBuffers is expected to be initialized (during the system boot-up process) with the maximum number of packet buffers that may be used by the MAC port associated with this local port descriptor during the reception of Ethernet frames. This field is decremented (by the firmware) with the number of packet buffers used by each received Ethernet frame; when it becomes less than or equal to zero, the switching firmware will refuse to accept any more frames, and will discard frames if received. The field is incremented whenever an Ethernet frame that was received on this MAC port has been completely transmitted or transferred, and its packet buffers are freed. If backpressure is enabled for this port and the value of the MaxBuffers field becomes less than zero, backpressure is started on the MAC channel preventing any more frames from entering on the port. Once the MaxBuffers becomes greater than zero, backpressure is halted allowing frames to enter the port again. More on backpressure is TBD.

The 16-bit MulticastMask field is used to restrict broadcasts, multicasts and floods of packets received on the MAC channel with which this local port descriptor is associated. The lower 8 bits of this mask correspond to the eight local MAC channels, while the upper 8 bits of the mask correspond to the eight possible devices in the system. The MulticastMask field is logically ANDed with the global restriction mask managed by the spanning tree protocol entity to permit broadcasts to be further restricted on a port-by-port basis.

A 16-bit BackoffMask is provided to allow the range of collision backoff values to be adjusted on a port-by-port basis. The BackoffMask field contents are logically ANDed with the standard random backoff value generated according to the IEEE 802.3 backoff timer computation algorithm. The uppermost 6 bits of the BackoffMask must be set to zero. The lower 10 bits should normally be set to all-ones, if the IEEE 802.3 standard backoff algorithm (which specifies a range of 0 to 1023 slot times, inclusive) is to be adhered to; however, smaller ranges of backoff values may be generated by reducing the number of '1' bits set in this mask.

The PortFlags field supplies several port-specific control bits that are used to determine the operating mode of the switching firmware when handling Ethernet frames that are received from or transmitted to this port. This field is formatted as:

7	6	5 4	3	2	1	0
BackPEn	BackPRn	Type	reserved	TXBlkd	RXBlkd	Special

BackPEn:

If set, indicates that backpressure flow control is enabled for this port. This bit is normally set via a configuration parameter at system initialization time.

BackPRn:

If set, indicates that backpressure flow control is running for this port.

Type:

This 2-bit field is used to initialize the 2-bit frame Type subfield in the Flags fields of data descriptors corresponding to Ethernet frames received on this MAC channel. Normally set to 01 binary, corresponding to unicast frames received on a local MAC channel.

TXBlkd:

If set, indicates that the MAC channel is blocked to transmit; i.e., no frames can be transmitted out this MAC channel. Normally used by the spanning tree algorithm.

RXBlkd:

If set, indicates that the MAC channel is blocked to received; i.e., frames received on this MAC channel must be discarded. Normally used by the spanning tree algorithm.

Special:

If set, indicates that the data descriptors created for Ethernet frames received on this MAC channel should have the Special flag bits set in their Flags fields, causing them to be specially handled by the switching firmware. Normally set to zero; usually set to a '1' by the spanning tree algorithm.

The remainder of the local port descriptor holds per-port counters that are maintained and updated by the switching firmware in order to support the SNMP and RMON MIB statistics. A total of 11 32-bit counters are implemented. Note that the TXDeferred counter tracks the number of frames whose transmission was delayed due to the need to defer to received frames as stipulated by the Carrier Sense Multiple Access protocol of IEEE 802.3. (This counter is part of the standard Ethernet MIB, and is typically used to track the utilization of the link.)

The Switch Processor caches all of the eight local port descriptors during normal operation. The in-memory copy of the local port descriptors, therefore, is likely to be 'stale' (i.e., outdated by information in the Switch Processor's data cache). The local port descriptors should preferably be read, if desired, by using the message interface maintained by the Switch Processor. If the contents of the local port descriptors must be directly read over the PCI bus, the Switch Processor data cache must be flushed prior to the read, again via the message interface. (The operation of the message interface is described below.)

Transmit Queue

Each local port descriptor maintains, as already described, a queue of frames that are awaiting transmission out the MAC channel. These frames have typically been received on some other port, and are directed to the given MAC channel during the switching process.

The transmit queue is constructed out of a linked-list of data descriptors, with each descriptor pointing to an Ethernet frame to be transmitted. The FirstDD and LastDD pointers in the local port descriptor point to the head and the tail of the transmit queue, respectively. The transmit queue imposes a First-In-First-Out (FIFO) discipline: new frames to be transmitted are always added to the tail of the queue, and the actual transmit process always removes frames from the head of the queue in order to transmit them. Note that a frame will remain in the transmit queue until its transmission has been completed successfully, or it is discarded for some reason.

The data descriptors and packet buffers in the transmit queue are never cached, and hence they can be inspected directly via the PCI bus interface. Note, however, that the head and tail pointers of the queue are located in the local port descriptor, which is itself a cached structure, as already mentioned.

Port Descriptor Counter Structure

The eight port descriptor counter structures are considered to be auxiliary data structures to the local port descriptors. Each local port descriptor is associated on a one-to-one basis with a 64-byte port descriptor counter structure (previously described), and contains various error and collision counters that are infrequently updated during normal operation. The purpose of separating these counters from the local port descriptor structures is to reduce the amount of data that must be cached in the Switch Processor data cache: the port descriptor counter structures are not cached by the Switch Processor.

Each port descriptor counter structure is formatted as follows:

31 24	23 16	15 8	7 0	Byte Offset
SingleCollide				0
MultCollide				4
LateCollide				8
CollideAbort				12
RXErrorOctets				16
AlignErrors				20
CRCErrors				24
DribbleErrors				28
MACErrors				32
LongErrors				36
JabberErrors				40
ShortErrors				44
RuntErrors				48
DropErrors				52
TXErrors				56
Reserved				60
RXBcastFrames				64
RXBcastOctets				68
RXFloodFrames				72
RXFloodOctets				76
RXMcastFrames				80
RXMcastOctets				84
Reserved				88
Reserved				92

SingleCollide:

32-bit count of single collisions encountered when attempting to transmit frames out this MAC port.

MultCollide:

32-bit count of multiple collisions encountered when attempting to transmit frames out this MAC port.

LateCollide:

32-bit count of late collisions encountered when attempting to transmit frames out this MAC port.

CollideAbort:

32-bit count of transmit frames that were discarded due to excessive collisions for this MAC port.

RXErrorOctets:

32-bit count of bytes in errored frames received on this MAC port.

AlignErrors:

32-bit count of frames received on this MAC port with alignment errors greater than 7 bits (DribbleError).

CRCErrors:

32-bit count of frames received on this MAC port with CRC errors.

DribbleErrors:

32-bit count of frames received on this MAC port with extra bits (1-7).

MACErrors:

32-bit count of frames received on this MAC port that were discarded due to internal MAC errors.

LongErrors:

32-bit count of frames received on this MAC port with valid CRCs but longer than the preset maximum frame length.

JabberErrors:

32-bit count of frames received on this MAC port with invalid CRCs and longer than the preset maximum frame length.

ShortErrors:

32-bit count of frames received on this MAC port with valid CRCs but below the preset minimum frame length.

RuntErrors:

32-bit count of frames received on this MAC port with invalid CRCs and below the preset minimum frame length.

DropErrors:

32-bit count of frames received on this MAC port that were dropped due to lack of buffer space to hold them (i.e., due to congestion).

TXErrors:

32-bit count of frames that were dropped while being transmitted out this MAC port due to internal errors (transmit FIFO underflow).

RXBcastFrames:
32-bit count of frames received on this MAC port which were broadcast.

RXBcastOctets:
32-bit count of bytes in frames received on this MAC port which were broadcast.

RXFloodFrames:
32-bit count of frames received on this MAC port which were flooded.

RXFloodOctets:
32-bit count of bytes in frames received on this MAC port which were flooded.

RXMcastFrames:
32-bit count of frames received on this MAC port which were multicast.

RXMcastOctets:
32-bit count of bytes in frames received on this MAC port which were multicast.

The counters maintained within the port descriptor counter structures correspond to those required by the RMON and SNMP MIBs.

The port descriptor counter structures occupy the memory locations described in the address map previously given, with the port descriptor counter structure for physical port 0 occupying the lowest address, and so on. As the port descriptor counter structures are not cached, they may be read at any time via PCI bus accesses.

Expansion Port Descriptor Tables

A set of data structures, collectively referred to as the *Expansion Port Descriptor Table*, is associated with the PCI expansion port. The Expansion Port Descriptor Table consists of eight 16-byte *expansion port descriptors*, each representing one of the (at most) eight ELAN 8x10 (or compatible) devices in the system, along with an associated set of eight *frame transfer queues* that contain frames pending to be transferred to the particular device. As noted, a single expansion port descriptor and the corresponding frame transfer queue is assigned to each device present on the PCI bus; as there can be at most seven external devices (i.e., excluding the device being considered), only seven of the eight expansion port descriptors are used, and the eighth descriptor is left untouched.

Note that the use of the expansion port descriptors is not limited to communicating with ELAN 8x10 devices. Any device that implements the same transfer protocol can be interfaced to the ELAN 8x10 PCI bus interface and assigned an expansion port descriptor (with the associated frame transfer queue), and the ELAN 8x10 device will transfer frames to this device without any special considerations.

Expansion Port Descriptor Structure

Each expansion port descriptor is assigned to a logical device, with indices ranging from 0 through 7. The expansion port descriptor structures are mapped into the memory locations as given in the foregoing address map, with the structure corresponding to logical device index zero being mapped into the lowest memory address, and so on. An expansion port descriptor occupies 16 bytes, and is formatted as follows:

31 24	23 16	15 8	7 0	Byte Offset
ChipNum	FirstDD			0
CtrMask	LastDD			4
reserved		MaxBuffers		8
RemoteDD				12

FirstDD:

24-bit pointer to first data descriptor waiting in transfer queue to be transferred to remote device.

ChipNum:

8-bit index assigned to remote chip corresponding to this expansion port descriptor; must be 0 for expansion port descriptor zero, and so on.

LastDD:

24-bit pointer to last data descriptor in transfer queue for this remote device.

CtrMask:

8-bit mask passed to DMA hardware when attempting to increment a request or acknowledge counter in the remote device; must correspond to ChipNum.

MaxBuffers:

16-bit count of packet buffers that remain to be allocated to hold incoming frames transferred from the remote device; when this count goes to zero, no more frames will be transferred.

RemoteDD:

32-bit PCI address of next data descriptor pointing to frame to be transferred from remote device. Generally, the uppermost 8 bits of this field remain constant, and give the upper 8 bits of the PCI base address set for the remote device; the lower 24 bits vary, and give the offset of the data descriptor with reference to the PCI base address.

The FirstDD and the LastDD fields serve to control the transfer queue for frames waiting to be transferred to the remote device associated with this expansion port descriptor. FirstDD contains the 24-bit memory address of the data descriptor at the

head of the transfer queue, while the LastDD field points to the data descriptor at the tail of the queue. The fields define a linked-list of data descriptors that in turn point to the chains of packet buffers containing the Ethernet frame data that must be transferred to the remote device. They are handled in a special manner, as follows.

Initially, the FirstDD and LastDD fields are set to point to a single empty data descriptor, i.e., one containing a NULL NextDD field and a NULL FirstPB field. When a frame needs to be transferred to the remote device, the following steps are performed:

1. The blank data descriptor in the transfer queue is updated to point to the frame, i.e., the FirstPB, LastPB, Flags, SrcPort, SrcChip and HashBkt fields are set up properly.
2. A **new** empty data descriptor is allocated and attached to the former (by setting the NextDD field of the former to point to the new empty data descriptor).
3. The LastDD field in the expansion port descriptor is updated to point to the new empty data descriptor.

The process is repeated as additional frames are queued for transfer to the remote device. (Note that the LastDD field always points to an empty data descriptor, which will be filled in when the next frame is queued.)

When the remote device acknowledges that it has completed the transfer of a given frame, the data descriptor at the head of the transfer queue (pointed to by FirstDD) is removed and freed, and the FirstDD pointer is advanced to the next data descriptor in the queue. It is illegal for either the FirstDD or the LastDD fields to be NULL.

The ChipNum and CtrMask fields are statically configured during initialization. ChipNum should be loaded with the index of the remote device, and is used by the firmware during consistency checks. The CtrMask field contains an 8-bit mask with the bit corresponding to the numeric index of the remote device (i.e., the least significant bit corresponds to device zero, and so on) set, and all of the other bits cleared. The mask is loaded into the DMA Controller when the firmware desires to increment either the request or acknowledge counter in the remote device during expansion port frame transfers.

MaxBuffers serves to impose a limit on the number of packet buffers that can be consumed by frame transfers from the given remote device into local memory. This field is set up during initialization to the desired (configurable) value, and is decremented for each packet buffer used to hold frames copied over the PCI bus from the remote device. (It is incremented as these packet buffers are freed after the frames have been completely transmitted, or discarded.) If this field becomes equal to or less than zero, the firmware will halt further transfers from the specific remote device, preventing it from unfairly using up all of the local buffer space. All subsequent requests from the same device, will be ignored. A special 'holdoff' mechanism is implemented in the expansion

port logic to reduce the frequency of request interrupts upon request by the firmware when MaxBuffers goes to zero.

The RemoteDD field holds the 32-bit PCI address of the data descriptor at the head of the remote device's frame transfer queue assigned to this ELAN 8x10 (i.e., the counterpart of the local frame transfer queue maintained by the FirstDD/LastDD pair in this expansion port descriptor.) This field is set up during system initialization. The uppermost 8 bits are loaded with the 8 MSBs of the PCI base address of the remote device. The lower 24 bits are loaded with the FirstDD field of the expansion port descriptor in the remote device (i.e., the address of the empty data descriptor initially allocated to the remote device's transmit queue.) After frame transfer from the remote device begins, the uppermost 8 bits of the RemoteDD field remain the same, but the lower 24 bits are progressively updated by the firmware as frames are copied from the remote device. The frame transfer queue discipline described above makes the updating of this field simple: the NextDD field of the last data descriptor to be copied from the remote device is simply loaded into the lower 24 bits of the RemoteDD field. (It will be obvious that the 24 LSBs of the RemoteDD field in this ELAN 8x10 device will always equal the FirstDD field in the expansion port descriptor of the remote device, i.e., the head of the frame transfer queue.)

As already noted, the expansion port descriptor corresponding to the index assigned to the ELAN 8x10 device itself (i.e., the device implementing the expansion port descriptor) is not used. Thus expansion port descriptor 0 in device 0 is not used, expansion port descriptor 1 in device 1 is not used, and so on. (Obviously, it is meaningless for a device to transfer frames to *itself* across the PCI bus.)

Unlike local port descriptors, the data queued on an expansion port descriptor need not be restricted to Ethernet frames. Inter-device messages are exchanged by formatting them into packet buffers and then placing them on the expansion port descriptor transfer queues corresponding to the target devices. The data descriptor flags should be set up properly to ensure that the messages are handled specially by the remote devices, and not treated as normal Ethernet frames. If the message is sufficiently compact, the packet buffers may be dispensed with and the entire message can be packed into the data descriptor; in this case, the FirstPB field of the data descriptor should be set to a NULL.

The expansion port descriptor structures are all cached by the Switch Processor in its data cache during normal operation; hence the in-memory copies of the expansion port descriptors may be out-of-date, and cannot be read directly from the PCI bus interface in a reliable manner unless a data cache flush is performed first via the message interface.

Frame Transfer Queue

A set of eight frame transfer queues are used to exchange forwarded packets and messages with up to seven external ELAN 8x10 devices that are part of the same switch. On a given ELAN 8x10 device, each frame transfer queue is dedicated to transferring data to a single external device (another ELAN 8x10, or any other device implementing the same transfer protocol).

The frame transfer queues are maintained by means of the FirstDD and LastDD pointers in the expansion port descriptors corresponding to the external devices. Each queue consists of a linked list of data descriptors, with each data descriptor pointing in the usual manner to the start and end of a list of packet buffers to be transferred from the local ELAN 8x10 to the remote device.

When an Ethernet frame or a message is to be forwarded to a remote device, the Switch Processor in the ELAN 8x10 will, as already described, initialize the blank data descriptor at the tail of the corresponding frame transfer queue to point to the packet buffers containing the Ethernet frame or message, and allocate a new blank descriptor to form the new tail of the transfer queue. It will then notify the remote device that data are available to be read from the frame transfer queue. The remote device is responsible for reading the contents of the data descriptor and all of the associated packet buffers from the local ELAN 8x10 memory space over the PCI bus. When the read is complete, the remote device must signal the local ELAN 8x10 that the data have been transferred and the queued packet and data descriptor can be freed.

None of the frame transfer queue data structures are cached by the Switch Processor; thus they may be read at will over the PCI bus interface without difficulty.

Address Hash Table

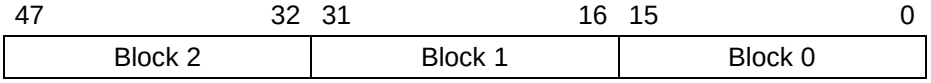
The ELAN 8x10 stores all learned or pre-configured Ethernet addresses using a *hash table* approach. Each entry in the address hash table is associated with a particular 48-bit IEEE MAC address. The table contains all of the information required to accept, process and switch packets to known (i.e., previously learned or set up) addresses. (In consonance with standard bridging practice, packets destined for unknown (not previously seen or set up) destinations must be flooded to all ports, or those determined using the standard IEEE 802.1d spanning tree.) The hash table also serves to hold per-host statistics information.

Note that when multiple ELAN 8x10 devices are present in a system the hash table becomes a **distributed** data structure, with a separate table being maintained by each device and kept consistent by exchanging information among the different ELAN 8x10 devices according to a predefined protocol.

The hash table consists of three types of data structures: an array referred to as the hash pointer array of between 256 and 8,192 pointers, a set of 64-byte data structures called hash buckets which hold information for locally reachable hosts on a per MAC address basis, and a set of 16-byte forwarding tags which hold abbreviated information for hosts reachable via external devices on a per-MAC address basis. (Note that the forwarding tags are only 16 bytes in terms of formatted data, but are actually stored into 64-byte blocks of memory for simplicity.)

Each pointer in the hash array points to a linked-list of zero or more hash buckets and/or forwarding tags. Each hash bucket or forwarding tag corresponds to a particular MAC address; only one hash bucket or forwarding tag may be present for a given MAC address in any device. (In addition, only one hash bucket may be present for a given MAC address in a complete system; however, multiple forwarding tags may be present in the various devices that point to this hash bucket.)

To access the hash bucket for a specific MAC address, the hash lookup engine in the DMA Controller first creates a *hash key* and uses it to generate an index into the hash pointer array. The hash key is obtained by dividing the 48-bit MAC address into three 16-bit blocks:



The three blocks are logically XORed together to create a single 16-bit value, which is then logically ANDed with a configurable 16-bit bitmask to obtain the index into the hash pointer array. The purpose of the bitmask is to limit the range of the indices to the actual size of the hash table. If there are 8,192 entries in the table, the bitmask should be defined such that the 13 LSBs are set to '1', and the 3 MSBs are set to '0'.

The hash lookup engine uses the index to locate the corresponding pointer within the hash pointer array, and reads the pointer to obtain the first element of the linked-list of hash buckets and/or forwarding tags that must be searched. It then scans the linked-list, comparing the complete 48-bit MAC address to be resolved with the MAC address fields within the hash buckets or forwarding tags, until the required hash bucket or forwarding tag is found (or no more entries are present in the linked-list). If a match is found, then the search is declared successful, and the memory address of the hash bucket or forwarding tag is passed to the Switch Processor. If the complete linked-list is traversed without finding a match, then the search is considered to have failed (i.e., the specified source or destination MAC address is not present in the address table), and an indication is accordingly passed to the Switch Processor by the hardware.

Hash Pointer Array

The hash pointer array is simply a linear array of 4-byte elements; each element contains a pointer to a linked list of hash buckets and/or forwarding tags.

31 24	23 0	
Unused	Pointer to bucket/tag (NULL if none)	...
Unused	Pointer to bucket/tag (NULL if none)	4N
Unused	Pointer to bucket/tag (NULL if none)	4N+4
Unused	Pointer to bucket/tag (NULL if none)	4N+8
Unused	Pointer to bucket/tag (NULL if none)	4N+12
Unused	Pointer to bucket/tag (NULL if none)	...

The hash pointer array occupies the locations in memory as described by the address map given previously. Initially, all the pointers in the array are set to zero (NULL), indicating that no MAC address is present in the hash table. As MAC addresses are learned (or statically created), the hash buckets or forwarding tags corresponding to these addresses are inserted into the hash table, and the relevant pointers in the array are updated to point to the linked-lists of hash buckets and/or forwarding tags. Note that newly learned addresses are always placed at the **head** of any linked-list thus created.

The size of the hash array, which determines the probability of collisions (i.e., the probability that multiple MAC addresses will be associated with a single hash pointer array index, leading to the need for traversing a linked-list), is a configurable parameter that may be set at initialization.

The contents of the hash array are never cached by the Switch Processor in its data cache; thus the hash array may be read via the PCI bus interface without data consistency issues.

Hash Buckets

Hash buckets are created to store frame handling and statistics information for all MAC addresses that are discovered to be directly reachable via a local MAC channel on the ELAN 8x10 device. Each hash bucket contains, in addition to the IEEE 802.3 MAC address with which it is associated, various fields used in forwarding the incoming frame, controlling the aging process, and maintaining per-host statistics. The format of a 64-byte hash bucket is given below:

31 24	23 16	15 8	7 0	Byte Offset
reserved	NextHB			0
MACAddr[23:16]	MACAddr[31:24]	MACAddr[7:0]	MACAddr[15:8]	4
MulticastMask		MACAddr[39:32]	MACAddr[47:40]	8
HBFlags	LPDPtr			12
TXFrames				16
TXOctets				20
RXErrorFrames				24
RXMcstFrames				28
RXFloodFrames				32
RXBcstFrames				36
RXUcstFrames				40
RXOctets				44
AgeControl				48
CreateTime				52
Reserved				56
Reserved				60

NextHB:

24-bit pointer to next hash bucket or forwarding tag in linked-list; NULL if none.

MACAddr:

48-bit IEEE 802.3 MAC address associated with this hash bucket.

MulticastMask:

16-bit mask used to restrict multicasts directed to this MAC address to a subset of the ports on this ELAN 8x10 and a subset of the devices in the system. Only valid for multicast MAC addresses, as indicated by the proper setting of the HBFlags field.

LPDPtr:
24-bit address of the local port descriptor assigned to the physical ELAN 8x10 port associated with this MAC address.

HBFlags:
8-bit per-MAC control flags, formatted as described below.

TXFrames:
32-bit count of total number of frames transmitted to this MAC address.

TXOctets:
32-bit count of total number of bytes transmitted to this MAC address.

RXErrorFrames:
32-bit count of frames received from this MAC address with errors.

RXMcstFrames:
32-bit count of valid multicast (i.e., IEEE 802.3 group/functional address) frames received on this port.

RXFloodFrames:
32-bit count of frames received on this port which need to be flooded.

RXBcstFrames:
32-bit count of valid broadcast frames received on this port

RXUcstFrames:
32-bit count of valid unicast frames received on this port.

RXOctets:
32-bit count of total bytes (errored or otherwise) received on this port.

AgeControl:
32-bit timestamp (obtained from the CLOCK register pair within the Switch Processor) when a frame was last received from this MAC address.

CreateTime:
32-bit timestamp (obtained from the CLOCK register pair within the Switch Processor) when this hash bucket was created.

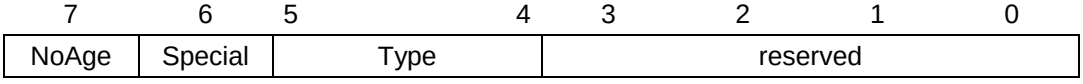
The NextHB field contains a 24-bit pointer to the next hash bucket (or forwarding tag) in the linked-list of hash buckets. It is NULL (all-zeros) if this hash bucket forms the end of the chain. The MACAddr field spans two consecutive words, and contains the complete 48-bit MAC address associated with this hash bucket; it is compared to the address extracted from the Ethernet packet during the address table lookup process to ensure that the proper hash bucket has been found.

The 16-bit MulticastMask field is used to restrict broadcasts, multicasts and floods of packets directed to a given multicast MAC address (i.e., when the contents of the MACAddr field is an IEEE group/functional address). The lower 8 bits of this mask correspond to the eight local MAC channels, while the upper 8 bits of the mask correspond to the eight possible devices in the system. The MulticastMask field is logically ANDed with the global restriction mask managed by the spanning tree protocol entity to implement multicasts on a per-MAC-address basis. It is ignored for unicast frames, or for hash buckets corresponding to source MAC addresses in received Ethernet frames.

The LPDPtr field contains a 24-bit pointer to the local port descriptor associated with this MAC address, i.e., corresponding to the physical port from which the given MAC address is reachable, and on which frames destined for this MAC address must be transmitted. This field is set up during the learning process (when the hash bucket is created) to point to the local port descriptor for the port on which the packet with the unknown source address was received. This address is the primary switching control value, and is used to locate the transmit queue on which the frame must be placed.

The LPDPtr field of hash buckets that are resolved during the source MAC address lookup process will be compared by the Switch Processor to the address of the local port descriptor corresponding to the physical MAC channel on which the Ethernet frame actually arrived. It is an error for this compare to indicate a mismatch, as this implies that the host corresponding to the source MAC address has apparently moved from one physical port to another within this ELAN 8x10 device. The Switch Processor will then declare a topology change situation, and perform the required topology change update process.

The HBFlags field supplies several MAC-address-specific control bits that are used to determine the operating mode of the switching firmware when handling Ethernet frames that are received from or transmitted to this MAC address. This field is formatted as:



NoAge:

If set, denotes a permanent hash bucket (i.e., one that the aging process is not permitted to remove). Normally cleared for addresses learned by the ELAN 8x10.

Special:

If set, indicates that the frame data must not be processed by the normal switching firmware, but must be handled specially by external code. Normally zero.

Type:

This 2-bit field denotes the type of MAC address (and hence the disposition of the Ethernet frame directed towards the MAC address). It is interpreted as follows:

Type	Interpretation
00	Invalid for hash buckets, reserved for forwarding tags
01	Unicast MAC address reachable via a local MAC channel
10	Invalid for hash buckets, reserved for forwarding tags
11	Broadcast MAC address reachable via a local MAC channel

The Type field is only valid if the Special bit is clear, indicating that the frame can be treated as a normal Ethernet frame. The Type field of the hash bucket corresponding to the destination MAC address is generally used to set up the Type subfield of the Flags field of the data descriptor (and thus to switch the frame).

The AgeControl field holds a 32-bit timestamp which records the absolute time a frame was last received from the source host with a MAC address corresponding to this hash bucket. The timestamp has a resolution of 1 microsecond and a range of approximately 1.12 hours. It is used during the aging process to locate hash buckets that have not been refreshed for some time, and are thus candidates for removal. The Switch Processor updates the AgeControl field with the contents of its built-in CLOCK real-time clock register whenever it receives a valid Ethernet frame, and the hash lookup engine resolves the source MAC address in the frame to the given hash bucket.

The CreateTime field is similar to the AgeControl field, but holds the time of creation for this hash bucket for RMON and SNMP counter purposes.

The remainder of the local port descriptor holds per-MAC counters that are maintained and updated by the switching firmware in order to support the RMON MIB statistics. A total of seven 32-bit counters are implemented. Note that the RXErrorFrames counter is only updated if sufficient bytes of the frame are received to permit the hash lookup engine to successfully resolve the source hash bucket.

The Switch Processor does not cache any hash bucket at any time in its internal data cache. It is therefore possible to read the contents of any hash bucket via the PCI bus interface without data consistency issues.

Forwarding Tags

A 16-byte forwarding tag (occupying the first 16 bytes of a 64-byte memory block, with the remaining 48 bytes being unused) is created whenever it is necessary to record that a particular MAC address is reachable by an external device (such as another ELAN

8x10), and frames directed to this MAC address should be directed to that device via the PCI expansion port. Forwarding tags have the following format:

31 24	23 16	15 8	7 0	Byte Offset
ChipNum	NextFT			0
MACAddr[23:16]	MACAddr[31:24]	MACAddr[7:0]	MACAddr[15:8]	4
MulticastMask		MACAddr[39:32]	MACAddr[47:40]	8
FTFlags	SrcHashBkt			12

NextFT:
24-bit pointer to next forwarding tag (or hash bucket) in linked-list; NULL if none.

ChipNum:
8-bit index assigned to remote device corresponding to this forwarding tag.

MACAddr:
48-bit IEEE 802.3 MAC address associated with this forwarding tag.

MulticastMask:
16-bit mask used to restrict multicasts directed to this MAC address to a subset of the devices in the system. Only valid for multicast MAC addresses, as indicated by the proper setting of the HBFlags field.

SrcHashBkt:
24-bit address of the target hash bucket within the remote device that contains the primary frame handling and statistics information for this MAC address.

FTFlags:
8-bit per-MAC forwarding control flags, formatted as described below.

The NextFT field contains a pointer to the next forwarding tag (or hash bucket) in the linked-list of forwarding tags. It is NULL (all-zeros) if this forwarding tag forms the end of the chain. The MACAddr field spans two consecutive words, and contains the complete 48-bit MAC address associated with this forwarding tag; as in the case of the hash bucket, the MACAddr field is compared to the address extracted from the Ethernet packet during the address table lookup process to ensure that the proper forwarding tag has been found.

The ChipNum field holds the index of the external device corresponding to this forwarding tag, i.e., the device to which frames directed to this MAC address should be sent. The ChipNum field is set up when the forwarding tag is inserted into the address table with the ID of the device requesting that the tag be created.

The 16-bit MulticastMask field is used to restrict broadcasts, multicasts and floods of packets directed to a given multicast MAC address (i.e., when the contents of the MACAddr field is an IEEE group/functional address). The lower 8 bits of this mask are unused, and set to zeros during the forwarding tag creation process. The upper 8 bits of the mask correspond to the eight possible devices in the system. The MulticastMask field is logically ANDed with the upper 8 bits of the global restriction mask managed by the spanning tree protocol entity to implement multicasts on a per-MAC-address basis. It is ignored for unicast frames.

The SrcHashBkt field contains the 24-bit memory address of the actual hash bucket that holds the necessary physical port routing information and the per-MAC statistics associated with this MAC address. This address is valid only in the memory address space of the **remote device**, and has no significance to the ELAN 8x10 containing the forwarding tag.

The remote device supplies the information placed in the SrcHashBkt field when it requests that a forwarding tag be created during the learning process. If the destination MAC address of a received Ethernet frame resolves to a forwarding tag, then the ELAN 8x10 device will transfer the frame to the remote device indicated by the ChipNum field in the forwarding tag, and will also place the SrcHashBkt value in the HashBkt field of the data descriptor associated with the frame. The remote device is expected to use the hash bucket address passed by this ELAN 8x10 to avoid having to perform yet another hash lookup in its own version of the address table; instead, it will locate the hash bucket indicated by the HashBkt field of the data descriptor, verify that it is indeed valid, and use the information within the hash bucket to switch the frame.

FTFlags supplies control bits that are used to determine the operation of the switching firmware when handling Ethernet frames that are directed to this MAC address. This field is formatted as:

7	6	5 4	3 2 1 0
reserved	Special	Type	reserved

Special:

If set, indicates that the frame data must not be processed by the normal switching firmware, but must be handled specially by external code. Normally zero.

Type:

This 2-bit field denotes the type of MAC address (and hence the disposition of the Ethernet frame directed towards the MAC address). It is interpreted as follows:

Type	Interpretation
00	Unicast MAC address reachable via a remote device
01	Invalid for forwarding tags, reserved for hash buckets
10	Broadcast MAC address reachable via a remote device
11	Invalid for forwarding tags, reserved for hash buckets

The Type field is only valid if the Special bit is clear, indicating that no special processing is required for frames directed to this forwarding tag. The Type field of the forwarding tag corresponding to the destination MAC address is generally used to set up the Type subfield of the Flags field of the data descriptor (and thus to switch the frame).

Note that it is an error for a source MAC address to resolve to a forwarding tag during the address lookup process. This indicates that the owner of the MAC address has apparently moved from a port on a remote device to a port on this ELAN 8x10. The Switch Processor will, in this case, announce a topology change situation, and go through the topology change update process required.

The Switch Processor does not cache any forwarding tag at any time in its internal data cache. It is therefore possible to read the contents of a forwarding tag via the PCI bus interface without data consistency issues.

Work Queue

The ELAN 8x10 maintains a special linked-list data structure known as the *work queue*. This queue buffers packets received from the various physical MAC ports and the PCI expansion port prior to processing and forwarding them to the appropriate entities. The primary purpose of the work queue is to free the low-level hardware interrupt service routines running on the Switch Processor from having to implement time-consuming and high-overhead switching functions, keeping them short and minimizing critical interrupt service latency. Each interrupt service routine accepts Ethernet frames from the hardware, either received over the local MAC channels or transferred across the PCI bus, and places them on the work queue. A less critical background task can then process the work queue in FIFO order when Switch Processor capacity is available to do so. The work queue hence permits the ELAN 8x10 system to absorb short periods of overload (in terms of frame processing requirements) without causing frame loss.

The work queue consists merely of a linked list of data descriptors, with each data descriptor pointing to a completely received and error-free Ethernet frame or message

(in the form of a linked list of packet buffers). After each frame is received from a local MAC channel or transferred over the PCI expansion bus, a data descriptor is created by the reception or transfer process running on the Switch Processor to point to the packet buffer chain and attached to the tail of the work queue. When frame or message processing is taking place, data descriptors (and the corresponding packet buffers) are removed from the head of the work queue, processed, and attached to the various transmit or transfer queues as determined by the switching algorithm. This ensures that frames are always transmitted in the order they were received.

A special hardware mechanism, consisting of a 16-bit counter along with some interrupt generation logic, is provided to simplify and speed up the maintenance of the work queue by the Switch Processor. This mechanism is referred to as the *work counter*. The work counter is expected to be incremented by 1 every time an entry is added to the work queue, and decremented by 1 every time an entry is removed from it. Whenever the work counter is non-zero, the interrupt generation logic generates a low-priority interrupt to the Switch Processor, indicating that items remain in the work queue that need to be processed. The Switch Processor may thus asynchronously add and remove entries from the work queue; the work counter will track the occupancy of the queue, and ensure that the Switch Processor never has to poll the queue to determine if entries need to be processed. The width of the work counter sets the upper limit on the size of the work queue (i.e., 65,535 entries); there is no other limit (beyond the availability of system resources, such as data descriptors and packet buffers) upon the depth of the queue.

Free Pools

During normal frame switching operations, the ELAN 8x10 must dynamically allocate blocks of memory to hold packet buffers, data descriptors, hash buckets and forwarding tags. (The memory allocation required to support the RTOS, and SNMP and RMON agents is not considered here; this type of memory allocation is expected to be handled as part of the RTOS.) These memory blocks are drawn from three free memory pools: a packet buffer pool, a data descriptor pool, and a hash bucket pool. All the pools consist of constant-sized blocks of memory linked together by means of pointers in the normal fashion.

The packet buffer pool consists of a linked-list of (nominally) 80-byte memory blocks on 16-byte boundaries, and is used to supply packet buffers for holding received Ethernet frames. Note that the block size may be changed between 64 and 240 bytes, in increments of 16 bytes, via a configurable parameter at initialization time. The head of the packet buffer pool is actually maintained by the DMA Coprocessor; an internal hardware register in the DMA Coprocessor is set up at initialization time to point to the first free buffer in the linked-list, and the DMA allocates blocks from the list as required. De-allocation of the packet buffers to the free pool is performed by the Switch Processor firmware after frames have been completely transmitted or transferred across the PCI bus, and is done to the tail of the pool. It is illegal for the packet buffer

pool to be completely empty at any point. The Switch Processor will declare a fatal system error and attempt to initiate a hardware reset if this occurs.

The data descriptor pool is used for supplying data descriptors. It consists of a linked-list of 20-byte fixed-size memory blocks, and is maintained entirely by the Switch Processor firmware. Blocks are allocated from this pool as required when a data descriptor must be created to hold the information for received frames or messages.

The hash bucket pool consists, in a similar manner, of a linked-list of 64-byte blocks that are intended to be used to create hash buckets and forwarding tags when MAC addresses are learned. One minor difference between this pool and the others is that the blocks in this pool are pre-formatted for rapid learning; all of the fields of the hash buckets in this pool, with the exception of the MAC address and the local port descriptor address, are initialized to their default values. This allows the learning process to simply allocate a free hash bucket, fill in the MAC address and local port descriptor pointer, and place the hash bucket into the address table, thereby minimizing the time spent formatting the hash bucket. When hash buckets are de-allocated to the free hash bucket pool (during aging, or topology changes), they are again re-formatted before being placed on the free pool. As the aging and topology change handling are principally non-time-critical background tasks, the additional time required to pre-format the hash buckets is not a significant issue.

When forwarding tags are created, only the first 16 bytes of the memory block is used; the remaining 48 bytes are left untouched (and preserve their original formatting). Forwarding tags are required during the learning of MAC addresses reachable by remote devices: they are allocated by the Switch Processor firmware during the learning process carried out when specially marked frames are received over the PCI expansion bus, and de-allocated (again by the Switch Processor firmware) when the corresponding MAC addresses are aged out. Removal of forwarding tags takes place only after their parent source hash buckets (on the remote devices) are removed. More details on the creation and removal of forwarding tags and hash buckets are provided in the sections dealing with learning and aging below.

The sizes and locations of the various free pools are completely configurable via parameters in the boot image, and can be adjusted by the system implementer as desired. These parameters implicitly determine the total amount of buffering in the system, the broadcast frame reception rate, the number of MAC addresses supported, and so on.

Spanning Tree Support Data Structures

Upon start-up, if spanning tree is enabled, the ELAN 8x10 creates a set of data structures required for implementing the IEEE 802.1d spanning tree algorithm. (In a multiple-ELAN 8x10 system, only the master device constructs the spanning tree data structures and implements the spanning tree algorithm; the slave devices exchange

data with the master device and transmit packets on its behalf, but do not maintain the spanning tree data structures.) Two different data structures are used by the spanning tree algorithm: a per-bridge data structure, only one of which is present in a system, and a per-port data structure, which is replicated for each port in the system.

Further description of the spanning tree data structures is TBD.

Broadcast Restriction Mask

A special data structure, referred to as the *broadcast restriction mask*, is used by the ELAN 8x10 to limit packet broadcasts in response to the results of the spanning tree algorithm. This mask consists of sixteen bits: eight bits correspond to the physical MAC ports implemented within the ELAN 8x10 itself, and the remaining eight bits are assigned to the external ELAN 8x10 devices connected to the PCI expansion bus, as shown below:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D7	D6	D5	D4	D3	D2	D1	D0	P7	P6	P5	P4	P3	P2	P1	P0

The eight port-related bits are indicated as P0 through P7, corresponding to physical ports 0 through 7 respectively, while the eight device-related bits are denoted as D0 through D7. Note that the device-related bit corresponding to the given ELAN 8x10 device itself must always be zero, as it is not permissible for an ELAN 8x10 device to send broadcasts to itself across the PCI bus. In addition, further device-related bits will be set to zero if there are less than eight devices in the system.

When a broadcast or multicast frame, or one directed to an unknown MAC address (which must be flooded) is received, the ELAN 8x10 inspects the broadcast restriction mask to determine the permissible destinations to which the packet may be sent. The mask is initialized as required upon power-on, and subsequently modified by the spanning tree algorithm or system manager to remove specific ports and devices from the broadcast topology. Note that hash buckets for multicast MAC addresses contain their own restriction masks, which are logically ANDed with the global broadcast restriction mask to further limit the broadcast topology; this feature may be used to implement multicast groups. In addition, broadcast restriction on a per-source-port basis is made possible by a similar multicast restriction mask in each of the local port descriptors, which is also ANDed into the global broadcast restriction mask.

RTOS Requirements

This section details the general requirements and specific data structures of interest that are implemented by the optional Real-Time Operating System.

This section is TBD.

UDP/IP Protocol Stack Requirements

This section describes the data structures created and maintained in order to support the UDP/IP protocol stack that is required by the SNMP agent to communicate with management platforms. It also describes the communication mechanisms used to exchange Service Data Units (SDUs) with the UDP protocol layer.

This section is TBD.

SNMP Requirements

This section gives a brief overview of the general requirements of the SNMP agent with regard to data structures used to support the MIB and the agent itself.

This section is TBD.

Storage Requirements

The local RAM storage requirements of the ELAN 8x10 are highly dependent on the system configuration (e.g., the amount of buffering required per port) as well as the various options selected, such as whether an SNMP agent is implemented or not. Some guidelines that may be used for estimating the amount of RAM for various system configurations are presented here.

The storage requirements may generally be divided into two categories: RAM space required for basic Ethernet frame switching operations, including operating data areas and memory pools; and the RAM space required to support an SNMP agent and the associated MIB, RTOS and UDP/IP stack.

General Frame Switching Requirements

For general frame switching operations, the storage requirements are limited to the switch processor operating environment and the free pools (i.e., the free packet buffer pool, the free data descriptor pool, and the free hash bucket / forwarding tag pool). In addition, a small amount of memory is required in order to copy the system boot image from EPROM or EEPROM to working RAM. The requirements for each type of structure are summarized in the following table:

Structure	Size	Quantity Required
Operating Environment	40 kB	Single block of memory starting at address 0x000000
Packet Buffers	80 bytes	Buffering per port multiplied by number of ports and divided by packet buffer size
Data Descriptors	20 bytes	Number of packet buffers plus (7*broadcast limit) plus (number of devices * broadcast limit)
Hash Buckets / Forwarding Tags	64 bytes	Number of MAC addresses to be supported in the system
System Boot Image Copy	32 kB	Temporary use; held reserved for simplicity

Of the different structures, the computation of how many data descriptors are required constitutes the most difficult, principally because of the differing requirements for unicast and multicast frames. Unicast frames require only a single data descriptor per frame, because, by definition, they can be placed on only one queue at a time. Broadcast/multicast frames, however, are queued to multiple destinations and thus require more than one data descriptor per frame. An upper limit on the number of data descriptors may be computed by observing that a given frame can be queued to at most (P -1) + (D - 1) queues, where P is the number of ports on a particular device, and D is the number of devices in the system. (The computation can be easily derived by noting that a broadcast frame cannot be retransmitted out the port it came in on, and also cannot be transferred over the PCI bus to the device that originally received it)

A good upper bound on the number of data descriptors required in the system may therefore be obtained by assuming minimum-sized frames (i.e., every packet buffer contains a complete Ethernet frame), and then taking into account the configurable system broadcast limit set up at initialization time to determine the additional number of data descriptors required to handle broadcast frames. The maximum number of data descriptors required is then given by:

$$N + (P - 1) * B + (D - 1) * B + E$$

where N = number of packet buffers in free packet buffer pool, P = number of ports in ELAN 8x10 (i.e., 7), B = broadcast limit in frames, D = number of devices in system including this ELAN 8x10, and E is the number of additional data descriptors that are required to support the messaging system. (A reasonable value for E is about 8 per port.)

As an example, consider a minimum system with two ELAN 8x10 devices, using 64 kB of buffering per port and supporting up to 2048 MAC addresses in the system. Assume that the broadcast limit is set to 1400 broadcast frames outstanding at a time, and about 128 data descriptors are reserved for holding system messages. The parameters for such a system would be:

Structure	Qty Needed	Storage Required
Operating Environment	1	40 kB
Packet Buffers	6550	512 kB
Data Descriptors	17878	280 kB
Hash Buckets	2048	128 kB
Boot Image Copy Area	1	32 kB
	TOTAL	1024 kB

This amount of memory fits well into a base configuration with 1 MB of DRAM. An example memory map is as follows (note that addresses increase downwards):

0x000000 0x009fff	Switch Processor Operating Environment
0x00a000 0x089fff	Packet Buffers
0x08a000 0x0cffff	Data Descriptors
0x0d0000 0x0effff	Hash Buckets
0x0f0000 0x0f7fff	Forwarding Tags
0x0f8000 0x0fffff	Boot Image Copy Area

The various start addresses of the memory blocks (with the exception of the Switch Processor operating environment) are entirely arbitrary, and can be modified at will via the configuration header file supplied during the boot image creation process.

Managed System Requirements

This section discusses the memory requirements for supporting the RTOS, the UDP/IP stack, the SNMP agent, and the MIB implemented by the ELAN 8x10.

This section is TBD.

Switching System Initialization

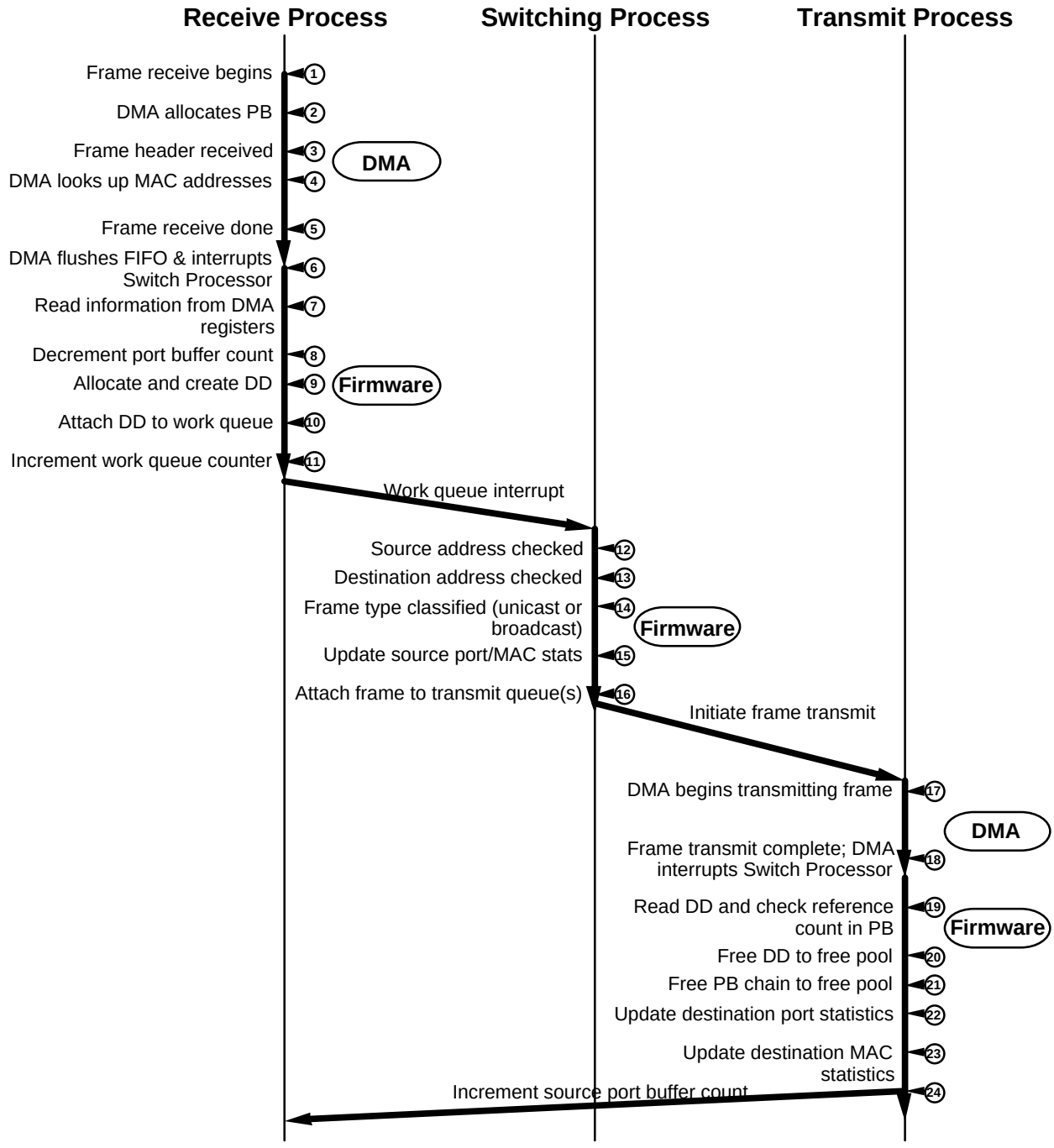
Prior to beginning Ethernet frame switching operations, it is necessary to initialize the various data structures (primarily the variables in the Switch Processor operating environment) and internal device registers to the appropriate values, and also to set up the free pools required for dynamic memory allocation. This section describes the initialization process in detail.

This section is TBD.

Switching Between Local Ports

The following describes the sequence of events that take place whenever an Ethernet frame is switched from one port to another through the local RAM. Note that the discussion focuses mainly on the actions performed by the Switch Processor firmware and the transactions performed by the frame switching processes, and only briefly describes the functions implemented by the hardware.

The switching sequence followed by the ELAN 8x10 upon the reception of a valid frame that will be transmitted out another of the ports implemented on the same ELAN 8x10 device is depicted in the following trellis diagram:


Receive Process
Switching Process
Transmit Process
Frame receive begins
1
DMA allocates PB
2
Frame header received
3
DMA
DMA looks up MAC addresses
4
Frame receive done
5
DMA flushes FIFO & interrupts Switch Processor
6
Read information from DMA registers
7
Decrement port buffer count
8
Allocate and create DD
9
Firmware
Attach DD to work queue
10
Increment work queue counter
11
Work queue interrupt
Source address checked
12
Destination address checked
13
Frame type classified (unicast or broadcast)
14
Firmware
Update source port/MAC stats
15
Attach frame to transmit queue(s)
16
Initiate frame transmit
DMA begins transmitting frame
17
DMA
Frame transmit complete; DMA interrupts Switch Processor
18
Read DD and check reference count in PB
19
Firmware
Free DD to free pool
20
Free PB chain to free pool
21
Update destination port statistics
22
Update destination MAC statistics
23
Increment source port buffer count
24

With reference to the numbers in the diagram above, the following steps are followed:

1. An Ethernet frame is transmitted by a source entity (DTE or switch) connected to one of the ELAN 8x10 ports for delivery to a destination entity connected to another ELAN 8x10 port. The ELAN 8x10 MAC channel begins receiving the frame data, converting it into 32-bit parallel data words, and places the data words into the 32-byte MAC interface FIFO. The MAC interface continues to receive and store data into the FIFO until the end of the Ethernet frame has been reached.

2. The DMA Controller allocates a packet buffer from the free packet buffer pool and transfers the MAC FIFO data into the buffer. When the packet buffer is filled, it allocates a new packet buffer, chains the new buffer to the old one, and continues. A 32-bit CRC is computed over the frame at the same time.

3. Reception and transfer to in-memory packet buffers continues in this manner until the 14-byte Ethernet frame header has been received.

4. When the Ethernet header has been completely received, the DMA Controller uses its hash table lookup hardware to scan the address table and resolve the source and destination MAC addresses into actual pointers to hash buckets.

5. The frame reception process continues until the frame has been completely received and transferred to packet buffers.

6. When frame reception terminates (as signaled by the MAC channel logic), the DMA Controller flushes any remaining bytes in the MAC interface FIFO to the packet buffers in external memory. It then notifies the Switch Processor of the presence of a completely received frame by asserting an interrupt. To ensure that the channel state is not altered by the reception of a new frame before the Switch Processor firmware has been able to read out the channel registers, the DMA Controller freezes the channel registers and prevents further activity on that particular channel until the Switch Processor notifies it that the channel state has been read and saved. (If any new frames arrive for that channel during this interval, they will be lost due to MAC FIFO overflows. Other channels are not affected, however.)

7. The Switch Processor begins executing the receive interrupt service firmware in response to the DMA Controller interrupt. The first operation by the firmware is to read out the contents of the DMA Controller channel registers corresponding to the MAC channel. It then notifies the DMA Controller that the registers have been read and reception of the next frame may proceed if necessary. In addition, the interrupt service routine also checks to see if a frame has been queued for transmit out this port; if so, it sets up the DMA Controller to begin transmitting the frame.

The receive interrupt service firmware subsequently checks the status code returned by the DMA to determine whether the frame is errored (due to a CRC, alignment, or other error) or valid. If the frame is errored, then the interrupt service firmware simply discards the packet buffers holding the frame data and updates the appropriate statistics counters. If the frame is valid, however, the firmware proceeds to further process the frame.

8. If a valid frame has been received, the firmware then proceeds to read the configured port buffer allocation limit (in the MaxBuffers field) for the source MAC channel from the local port descriptor, and subtracts the number of packet buffers consumed by the DMA Controller in order to hold the received frame. If the result is greater than zero, the firmware writes it back to the MaxBuffers field in the local port descriptor and accepts the frame; on the other hand, if the result is equal to or below a pre-set threshold, the firmware still accepts the received frame, but also invokes the backpressure function to begin flow-controlling the channel, if backpressure is enabled; if the result is equal to or less than zero, the buffer limit has been exceeded, and the firmware discards the frame.

9. Assuming that the buffer allocation limit has not been exceeded, the firmware allocates a 16-byte data descriptor and initializes it to point to the packet buffer chain that was read from the DMA Controller channel registers. The data descriptor is initialized to the default type defined by the Flags field in the local port descriptor (normally that of a Local Unicast frame). In addition, the 24-bit pointers to the address table entries (if any) for the source and destination MAC addresses that were located by the address table lookup hardware in Step 4 are encoded into 16-bit indices and placed into the data descriptor, and the data descriptor Flags field is set to indicate whether the address lookups were successful, as well as to reflect some status indications returned by the DMA Controller. (The format of these data descriptors has already been supplied in a preceding section.)

10. The newly formatted data descriptor is placed on the tail of the work queue, and internal pointers to the work queue are then updated.

11. Finally, the hardware work counter (which tracks the number of entries in the work queue) is incremented to indicate that another frame has been added to the queue. The receive interrupt service routine has now completed operation. As the work counter is non-zero, it automatically generates an interrupt to the Switch Processor (as already described).

12. When the Switch Processor receives the work interrupt, it begins executing the work queue interrupt service routine. This routine will remove the data descriptor (and the associated packet buffer chain holding the frame data) at the head of the queue, and decrement the work counter. The frame will then be processed to perform the necessary switching functions.

    The first action of the work queue interrupt service firmware is to check the data

descriptor flags to determine whether the source and destination MAC address lookups have been successful. If valid 16-bit encoded source and/or destination hash bucket indices are present, the firmware will convert these to the actual 24-bit memory addresses of the corresponding hash buckets. The source address checking and learning process is then carried out.

If the source address lookup has failed, the frame contains a new, un-learned MAC address; the firmware will then execute the address learning function, as described in a succeeding section. If the source address lookup has succeeded but the hash bucket is inconsistent with the expected address table entry (e.g., it is a forwarding tag rather than a hash bucket, or the hash bucket indicates an improper source physical port), then the firmware will declare a topology change and invoke the spanning tree functions. If the source address lookup has returned a valid hash bucket, the firmware will write a 32-bit timestamp (obtained from the CLOCK register within the Switch Processor) into the hash bucket for aging purposes, and then proceed to the next step.

13. The firmware now verifies that the destination address lookup was successful; if not, the firmware invokes the frame flooding function to broadcast the frame to all ports that are on the spanning tree. If the destination address lookup succeeded, but locates a forwarding tag, this indicates that the firmware must transfer the frame to a remote device; this is described in the next section. Finally, the physical ports referenced by the source and destination hash buckets are compared to determine whether the frame should be filtered (i.e., the source and destination ports are the same). The frame is forwarded to a single local port only if the destination address lookup succeeded and returned a valid hash bucket referencing a different physical port present on this ELAN 8x10. Note that the Flags field in the destination hash bucket is also checked to verify whether the frame requires special handling.
14. The frame type is classified according to the Type subfield in the Flags field of the destination hash bucket, i.e., whether the frame is to be treated as a unicast or broadcast/multicast. If the frame is to be a broadcast/multicast, the MulticastMask field in the hash bucket is consulted to determine the ports and devices to which the frame should be sent.
15. The frame has been successfully processed and classified; the source MAC address statistics (i.e., the per-host statistics in the source hash bucket) as well as the source per-port statistics are updated appropriately at this time.
16. The firmware now queues the frame for transmission or transfer (or both, in the case of broadcasts). It is assumed here that the frame is intended to be transmitted out one or more local MAC channels. In this case, the firmware identifies the physical port(s) on which the frame must be transmitted, locates the local port descriptor(s), and obtains the transmit queue head and tail pointers from these local port descriptor(s). The firmware then enters the frame

into the appropriate transmit queue(s). Note that entry into multiple queues - i.e., a broadcast - entails the allocation of additional data descriptors to act as queue entries, all set up to point to the same linked-list of packet buffers. The reference count in the packet buffer will be updated as required for broadcast frames.

When the frame has been placed on any transmit queue, the firmware also inspects the status of the corresponding DMA Controller channel . If the channel is not busy (either receiving a frame, or transmitting a previously queued frame), then the firmware will load the channel registers with the address of the first packet buffer in the transmit queue entry at the head of the queue, and enable the DMA Controller to begin transmitting data on that channel..

17. When the DMA Controller channel is enabled for transmit, the associated MAC logic attempts to seize the medium and send the data: it defers to passing frames, if any, waits for the required interframe gap and then starts sending the 802.3 preamble, followed by the frame data. The DMA Controller will read data out of the packet buffers in memory to fill the MAC channel FIFO with frame data, traversing the linked-list of buffers until it has reached the end.

    If a collision occurs at any time during frame transmit, the MAC channel logic will terminate the transmission, cause a jam pattern to be sent, and then start the backoff timer with the pre-loaded backoff value. In the mean time, the DMA Controller will signal a collision interrupt to the Switch Processor; the Switch Processor firmware will then re-compute the backoff timer for the next possible collision and set up the DMA Controller channel registers to re-transmit the frame after the backoff period ends.

18. The transmit process continues until no more packet buffers remain to be transmitted. The DMA Controller detects that all the frame data has been transmitted when it reaches the end of the linked-list of buffers (as indicated by a NULL NextPB pointer) and the MAC channel FIFO is empty, signifying that the data have been placed on the medium. At this point, the DMA Controller interrupts the Switch Processor to notify it that the frame transmit has completed successfully.

    Note that the DMA Controller computes a running CRC over the transmitted data. If the CRC is found to be in error, the DMA Controller will optionally append a correct CRC at the end of the frame, if the appropriate mode bit is set in a configuration register.

19. The Switch Processor responds to the DMA Controller end-of-transmit interrupt and begins executing the end-of-transmit interrupt service routine firmware. The firmware first reads the data descriptor at the head of the transmit queue for the port (which is associated with the frame that was just transmitted), obtains a pointer to the last packet buffer in the linked-list of buffers holding the

transmitted frame, and reads out the reference count from the RefCount field in the packet buffer. The reference count is decremented, and the result inspected: if the count is non-zero, then additional queue entries are pointing to the frame data, and the packet buffer chain should not be freed; instead, the new reference count is written back to the packet buffer. The Flags field of the data descriptor is also checked to see if the frame requires special handling.

20. The data descriptor read in the previous step is freed to the free data descriptor pool, and the transmit queue maintained by the local port descriptor is advanced to the next entry (if any).

21. If the reference count checked in the preceding step went to zero (or the frame was a unicast), then the frame does not have to be transmitted out any more ports; the firmware will then de-allocate the packet buffers holding the frame back to the free packet buffer pool.

22. The firmware now reads and updates the per-port transmit statistics stored in the local port descriptor.

23. If the frame was a unicast, then the HashBkt pointer field in the data descriptor will denote a valid destination hash bucket. The firmware will use this pointer to access and update the per-MAC transmit statistics in the hash bucket as well. Note that transmit statistics are not updated on a per-MAC basis in the case of broadcasts or multicasts.

24. When the above processing has been completed, the firmware will (if it freed up the packet buffer chain in step 21 above) increment the source port buffer count by the number of buffers freed in order to restore the buffer limit to its original value. The source port is obtained via the SrcPort entry in the data descriptor, which holds the index of the physical port on which the frame arrived (and hence indicates the local port descriptor containing the MaxBuffers field that must be incremented). This completes the processing of the frame.

It should be noted that the Switch Processor firmware is actually responsible for re-transmitting a frame over a local port if the transmission attempt is aborted due to collision or the need to defer to an incoming frame. This is done automatically by simply checking the transmit queue associated with that local port whenever it is known that the DMA Controller channel for that port is idle; if one or more frames are present in the transmit queue, then the frame at the head of the queue is passed to the DMA Controller for transmission. This operation is performed at three points in the course of frame switching: when a frame has just been received (and presumably the MAC logic is counting out the inter-frame gap), when a frame has just been transmitted, or a transmit attempt was terminated by a collision, and when a frame is queued for transmission out the given port.

Switching Via The Expansion Bus

If a frame received on a physical port of any ELAN 8x10 device is addressed to a destination host connected to a MAC port of an external (remote) ELAN 8x10 device, then the frame will be passed to the external device via the PCI expansion bus interface.

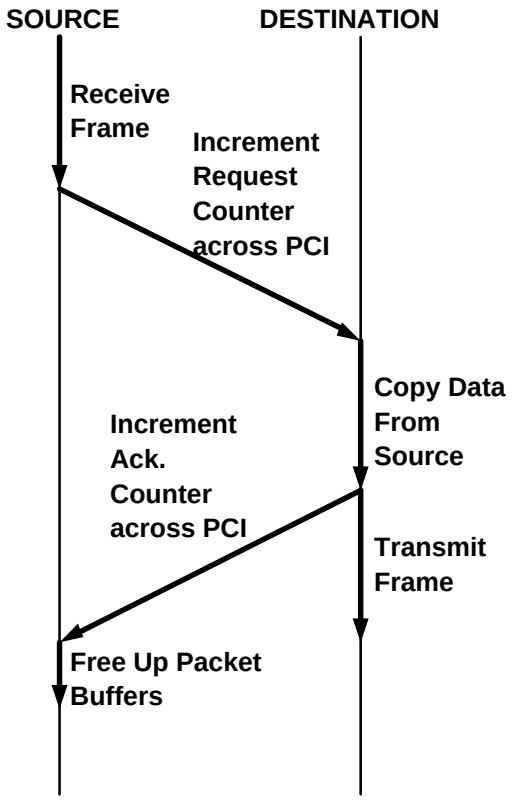
Frame switching via the PCI expansion bus is carried out in two stages: the source ELAN 8x10 receiving the frame appends the frame to the frame transfer queue that it maintains for the destination device (and signals the destination device that another frame is available); the destination device is then expected to copy the data from the source ELAN 8x10 to its own memory space, and notify the source ELAN 8x10 when the transfer is complete (i.e., acknowledge the transfer). The destination device is then completely responsible for subsequently transmitting the data on the correct Ethernet link.

Note that broadcast or multicast transfer to multiple remote devices is done by having each destination device separately copy the same frame across the PCI bus to their own memory spaces, and separately acknowledge the copy to the source of the frame. This will be followed by multiple transmissions of the frame over the local ports within the destination devices, without any further use of the PCI bus. A bandwidth multiplication effect is thus obtained when the destination devices (such as other ELAN 8x10 chips) transmit the frame over multiple ports, as a single copy of the frame over the PCI bus can result in multiple transmissions within the destination devices.

Request and Acknowledge Counters and Protocol

The ELAN 8x10 (and other devices that expect to communicate with the ELAN 8x10) implements a set of hardware counters that assist in the signaling across the PCI bus. A total of eight request counters and eight acknowledge counters are implemented, one corresponding to each device in the system. These counters are mapped into the upper 64 kB of the 16 MB address space assigned to each ELAN 8x10 device, and are incremented as required to notify target devices that frames are available to be transferred, or that frames have been transferred and the memory resources used by them may be freed.

A relatively simple protocol is followed when transferring frames between ELAN 8x10 devices across the PCI bus. The following figure shows a high-level view of the handshake employed:



The frame transfer handshake is initiated when a source device receives a valid frame from the MAC channel and determines that it is destined for an external destination device. At this point, it performs a single-word write across the PCI bus to increment a special counter in the destination device. This counter is referred to as the "request counter", and serves to notify the destination device that a frame (or message) is ready to be transferred across the PCI bus from the source device.

When the destination device detects that the request counter has been incremented, it copies the frame data from the head of the transfer queue maintained for it by the source device across the PCI bus to buffers in its local memory using a DMA channel. The destination then performs another single-word write across the PCI bus to another special counter in the source, referred to as the "acknowledge counter". This write automatically increments the acknowledge counter. Once this operation is done, the destination device is free to go ahead and transmit the newly copied frame from the buffers in its local memory space.

When the source device detects that the acknowledge counter has been incremented, it knows that the destination device has successfully copied the frame data out of the source buffers, across the PCI bus, and into local buffers at the destination. The source buffers are no longer needed; the source device can then free the buffers to the free pool, to be used for the storage of some subsequent frame.

Note that the use of counters rather than flags or registers permits multiple requests or acknowledges to be outstanding at any device without problems. Each destination device must thus maintain a request counter that the source device can increment, and each source device must maintain a counter that the destination can increment in acknowledgement. If the "destination" device wishes to send frames back to the "source" device, then a mirror set of request and acknowledge counters must be implemented in the devices, and the same protocol works but in mirrored fashion.

To extend the protocol to support multiple devices in the system, a bank of request counters and a bank of acknowledge counters are built into each device: each device in the system contains eight request counters and eight acknowledge counters, thereby permitting systems to be built with up to eight inter-communicating devices. The counters are mapped into the PCI memory address space starting at offset 0xffff00 hex from the base address assigned to the device, as follows:

0xzzffffff	Internal registers, etc.
0xzzffff40	
0xzzffff3c	Acknowledge Counter #7
0xzzffff38	Acknowledge Counter #6
0xzzffff34	Acknowledge Counter #5
0xzzffff30	Acknowledge Counter #4
0xzzffff2c	Acknowledge Counter #3
0xzzffff28	Acknowledge Counter #2
0xzzffff24	Acknowledge Counter #1
0xzzffff20	Acknowledge Counter #0
0xzzffff1c	Request Counter #7
0xzzffff18	Request Counter #6
0xzzffff14	Request Counter #5
0xzzffff10	Request Counter #4
0xzzffff0c	Request Counter #3
0xzzffff08	Request Counter #2
0xzzffff04	Request Counter #1
0xzzffff00	Request Counter #0
0xzzfffeff	External local memory and internal registers
0xzz000000	

Note that the component of the 32-bit PCI address marked "zz" (as in 0xzz000000, for example) represents the base address loaded into the uppermost 8 bits of the memory base address register.

As shown, the eight request counters occupy consecutive 32-bit locations from an offset of 0xffff00 to 0xffff1c from the base; the eight acknowledge counters occupy locations from an offset of 0xffff20 to 0xffff3c. The address of acknowledge counter #0 may thus be obtained by adding 0x20 (i.e., 32 decimal) to the address of request counter #0, and so on.

Each pair of request and acknowledge counters marked with a given index is dedicated to a particular device in the system. If the devices are numbered from 0 through 7, and

referred to as device #0, device #1, and so on, then request counter #0 and acknowledge counter #0 are dedicated to device #0 in **every** device; request counter #1 and acknowledge counter #1 are dedicated to device #1; and so on.

The use of the counters is as follows. If, for example, chip #3 wishes to transfer a frame to chip #5, it will increment request counter #3 implemented by chip #5 (signaling to chip #5 that chip #3 has a frame pending for transfer). Chip #5 determines by the index of the request counter the source of the data (i.e., that it comes from chip #3), performs the necessary transfer across the PCI bus, and then acknowledges the transfer by incrementing acknowledge counter #5 implemented by chip #3. This signals to chip #3 that chip #5 has read a frame from the former's local memory, and the packet buffers may be freed.

Note that one of the eight pairs of request and acknowledge counters will always be unused in each device in the system. In chip #0, for instance, request and acknowledge counters #0 will never be incremented (incrementing request counter #0 in chip #0 would imply that chip #0 was attempting to send a frame to itself across the PCI bus, which is clearly redundant). Essentially, the pair of request and acknowledge counters with an index that equals that of the chip itself will never be incremented.

If bilateral and uniform exchanges of frames are required in the system, then it is obvious that at most eight devices (one for each pair of request/acknowledge counters) can be supported. The following table shows, as an example, a three-device system with devices numbered from 0 through 2, and identifies the request and acknowledge counters that are incremented when a frame is transferred from any source device to any destination device:

	Dest = chip #0		**Dest = chip #1**		**Dest = chip #2**	
Src=chip #0	N/A	N/A	Req ID = 0	Ack ID = 1	Req ID = 0	Ack ID = 2
Src=chip #1	Req ID = 1	Ack ID = 0	N/A	N/A	Req ID = 1	Ack ID = 2
Src=chip #2	Req ID = 2	Ack ID = 0	Req ID = 2	Ack ID = 1	N/A	N/A

In the table above, "Req" refers to the index of the request counter, and "Ack" refers to the index of the acknowledge counter. Note that the request counter is always incremented in the **destination** device, and the acknowledge counter is always incremented in the **source** device. The index of the chip making the request or returning the acknowledge can be obtained by simply inspecting the index of the incremented counter.

An inspection of the table above will reveal that the indices of the request and acknowledge counters that are incremented by any particular chip are always the same. Thus, for example, chip #1 will always increment request counter #1 in any other device when acting as a source of data, and will always increment acknowledge counter #1 in

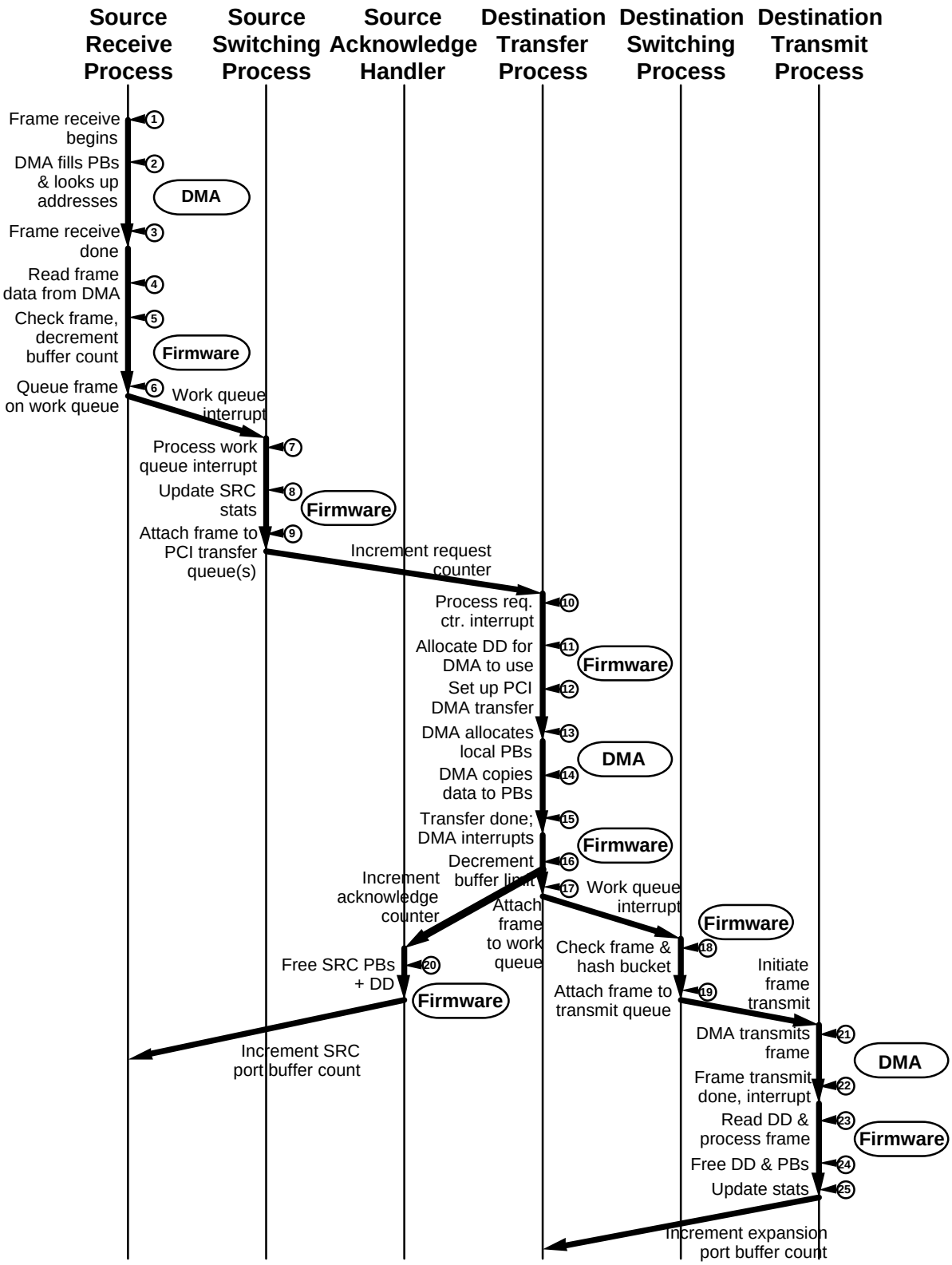
another device when acting as a destination and acknowledging the receipt of data. This fact is taken advantage of by the ELAN 8x10 hardware: the CTRBASE register in the DMA Controller is set up at configuration time in each ELAN 8x10 device with the offset of the request counter index assigned to the device itself; the firmware then simply has to indicate the target of a request or acknowledge, and the hardware can use the internal CTRBASE register to determine the actual counter to increment. This also facilitates broadcasts, as the firmware can simply supply a bitmask in the CTRMASK DMA Controller register corresponding to the chips that form the target of the broadcast, and the hardware will use the bitmask in conjunction with the CTRBASE register to increment multiple request counters across the system.

The foregoing has dealt with the use and configuration of the hardware communication means, i.e., the request and acknowledge counter banks. The situation for the actual data transfer is substantially simpler, as the ELAN 8x10 hardware and firmware place no restrictions on the PCI addresses from which data copies may be done. Once a particular request has been mapped to a specific source chip via the request counter index by any ELAN 8x10 device, the 32-bit PCI address from which the data transfer must be performed is defined by the 32-bit RemoteDD field of the expansion port descriptor assigned to track transfers from that chip. Thus the only hard limitation on the number of devices that may be supported by an ELAN 8x10 system is set by the number of request and acknowledge counters that are implemented in the ELAN 8x10 chips.

It should be noted that the assignments of counter indices to devices (as well as the association of devices with actual PCI addresses) is performed statically at initialization time by the boot image executed by every ELAN 8x10 device.

General Frame Transfer Sequence

The following diagram details the sequence of events when switching a unicast frame from a MAC port on the source ELAN 8x10 device to a MAC port on a remote ELAN 8x10 device via the expansion bus. (The same general sequence holds even if the destination device is not an ELAN 8x10 chip, as all devices that expect to communicate with the ELAN 8x10 must follow the same protocol.) Note that if a broadcast frame is sent to two or more ELAN 8x10 devices, then the frame reception and processing steps remain the same within the source device, but the frame transfer steps are repeated for each additional remote ELAN 8x10 device.


Source Receive Process
Source Switching Process
Source Acknowledge Handler
Destination Transfer Process
Destination Switching Process
Destination Transmit Process
Frame receive begins
DMA fills PBs & looks up addresses
DMA
Frame receive done
Read frame data from DMA
Check frame, decrement buffer count
Firmware
Queue frame on work queue
Work queue interrupt
Process work queue interrupt
Update SRC stats
Firmware
Attach frame to PCI transfer queue(s)
Increment request counter
Process req. ctr. interrupt
Allocate DD for DMA to use
Firmware
Set up PCI DMA transfer
DMA allocates local PBs
DMA
DMA copies data to PBs
Transfer done; DMA interrupts
Firmware
Decrement buffer limit
Increment acknowledge counter
Work queue interrupt
Attach frame to work queue
Firmware
Check frame & hash bucket
Free SRC PBs + DD
Initiate frame transmit
Attach frame to transmit queue
Firmware
DMA transmits frame
DMA
Increment SRC port buffer count
Frame transmit done, interrupt
Read DD & process frame
Firmware
Free DD & PBs
Update stats
Increment expansion port buffer count

1. One of the ELAN 8x10 devices in the system (the source device) begins receiving an Ethernet frame on one of its MAC channels; the frame is intended for a destination entity connected to a port on another ELAN 8x10 device (the destination device). The ELAN 8x10 MAC channel converts and places the incoming data into the MAC interface FIFO, as usual, until the end of the frame has been reached.

2. The DMA Controller allocates one or more packet buffers from the free packet buffer pool to hold the frame and fills them with data words copied from the MAC channel FIFO, computing the required 32-bit CRC over the frame at the same time. When the 14-byte frame header has been received, the DMA Controller uses the hash table lookup mechanism to scan the address table and obtain the pointers to the source and destination address table entries for the MAC addresses in the frame. Note that the destination address table entry in this case must be a forwarding tag, as the frame is destined to be transferred across the PCI bus to another device. The source address table entry (if one exists) must be a hash bucket.

3. The frame reception process continues until the frame has been completely received and transferred to packet buffers. When reception terminates, the DMA Controller flushes the MAC FIFO to the packet buffers and notifies the Switch Processor of the received frame by via an interrupt.

4. The Switch Processor executes the normal receive interrupt service routine in response to the interrupt; as usual, it first reads out the contents of the appropriate DMA Controller channel registers, and then re-enables the DMA channel. The receive interrupt service firmware checks the status code returned by the DMA to determine whether the frame is errored, and performs the necessary processing as indicated.

5. If a valid frame has been received, the firmware verifies that the port buffer allocation limit in the MaxBuffers field of the local port descriptor has not been exceeded (and begins backpressure if necessary, or frame discard if the limit has been exceeded, as already described).

6. The receive interrupt service firmware now allocates and initializes a data descriptor to point to the frame, as usual, and queues it on to tail of the work queue. The work counter is incremented, generating a work queue interrupt.

7. When the Switch Processor receives the work interrupt, it begins executing the work queue interrupt service routine, and will remove the frame from the work queue and process it. As before, the firmware begins by decoding the source hash bucket pointer returned by the DMA Controller and checking it for consistency, and will announce a topology change or learn a new address based on the results. It will also update the aging timestamp in the source hash bucket (if a valid one exists). The standard filter check is performed to insure that the source and destination ports are not the same. The destination address

table entry is also decoded and inspected; as this description relates to frames being forwarded across the PCI bus, the destination address table entry will be found to be a forwarding tag indicating a remote device as the target of the frame, or will indicate a multicast, or will be non-existent (in which case the frame will be flooded, and must be sent across the PCI bus to remote devices). If a forwarding tag was located during the address lookup for the destination MAC address, the remote hash bucket pointer is extracted from the SrcHashBkt field of the forwarding tag and placed in the HashBkt field of the data descriptor; otherwise, the source hash bucket pointer is placed in the HashBkt field.

8. The frame has been successfully processed and classified; the source MAC address statistics (i.e., the per-host statistics in the source hash bucket) and the per-port statistics are updated appropriately at this time.

9. The firmware now queues the frame for transfer to a remote device (in the case of broadcasts or floods, it queues the frame for transmission out local ports as well). The firmware then obtains the chip index or indices from the forwarding tag (or the broadcast restriction mask, in the case of broadcasts or floods), locates the expansion port descriptor(s), and obtains the transfer queue tail pointer from the expansion port descriptor(s). The firmware then enters the frame into the appropriate transfer queue(s), in the manner already described in a preceding section. Note that entry into multiple queues - i.e., a broadcast - entails the allocation of additional data descriptors and the generation of multiple queue entries, all set up to point to the same linked-list of packet buffers.

   When the frame has been placed on all of the necessary transfer queues, the firmware inspects the status of the DMA Controller counter increment channel . If the channel is not busy, then the firmware will load the channel registers with a bitmask that indicates the remote device(s) to which the frame is directed, and enable the DMA Controller to increment the request counter(s) in the remote device(s) across the PCI bus.

10. The DMA Controller on the source device performs a request counter increment to the specified remote device(s) as instructed by the firmware in step 9. This request counter increment causes, in turn, a request interrupt to be generated to the Switch Processor(s) in the remote device(s), notifying the Switch Processor of the presence of an outstanding frame transfer request. The Switch Processor(s) will then begin executing the interrupt service routine firmware that is intended to handle these frame transfer requests.

11. The first step in the handling of the request counter interrupt at the remote device is to inspect the buffer limit (maintained in the MaxBuffers field of the expansion port descriptor corresponding to the source device) to determine if the source device has exhausted its buffer allocation. If sufficient buffers remain, the firmware will then allocate a data descriptor from the free data

descriptor pool to hold a copy of the original data descriptor in the source device.

12. The firmware will then set up the DMA Controller to transfer the head of the appropriate transfer queue (both the data descriptor as well as the packet buffer chain that it points to) from the source device into local memory. It will pass the DMA Controller a pointer to the newly assigned data descriptor, into which the DMA Controller is expected to place the contents of the copied data descriptor; packet buffers must, however, be allocated dynamically by the DMA Controller. At this point, the request counter interrupt handler firmware has completed its processing; it decrements the request counter that triggered the interrupt (signifying that the request has been handled) and exits.

13. The DMA Controller will begin copying data across the PCI bus from the source device to the remote device's memory space. The first item to be copied is the data descriptor at the head of the transfer queue in the source device; this is written into the pre-allocated data descriptor passed to the DMA Controller in step 12. (Note that only the first 16 bytes of the data descriptor are read as the last four are only used internally.)The DMA Controller will then start transferring the actual frame data (i.e., the packet buffers) from the source device. It begins by allocating a packet buffer to hold the data. (When this packet buffer is filled, it will allocate the next buffer and chain it to the first one, thus building up a linked-list of buffers in local memory.)

14. The FirstPB field of the copied data descriptor provides a pointer to the packet buffer chain associated with it; the DMA Controller continues to copy data across the PCI bus from the packet buffers at the source device end into the locally allocated packet buffers, traversing the linked list of source packet buffers to build a virtually identical list of packet buffers in the remote device's memory. (The only difference between the two sets of packet buffers, once the DMA Controller has completed the copy, will be that the next pointers in the NextPB fields will be different, as the two linked lists lie in different memory spaces.) Note that the DMA Controller automatically writes a pointer to the first packet buffer in the local memory space into the FirstPB field of the copied version of the data descriptor, obliterating the original contents of the FirstPB field as originally copied.

15. When the transfer is done (the data descriptor and all of the packet buffers have been copied), the DMA Controller halts the transfer and interrupts the Switch Processor.

16. The Switch Processor responds to the interrupt by activating the PCI read completed interrupt service routine. The interrupt service routine firmware will then read the necessary information out of the DMA Controller channel control registers, and then decrement the MaxBuffers field of the proper expansion port descriptor by the number of copied packet buffers to signify that the source device has used up some of its local memory allocation at the remote device.

Once this is done, the firmware will set up the DMA Controller increment channel to increment the acknowledge counter assigned to the remote device within the source device across the PCI bus. (Thus, if device X is transferring the frame to device Y, device Y will increment its dedicated acknowledge counter within device X to indicate that the frame transfer has been completed successfully.) This in turn will cause an interrupt to be generated to the Switch Processor in the source device (exactly as in the case of the request counter) and notify it that the data at the head of its frame transfer queue may be released.

17. The firmware within the destination device will, finally, update the data descriptor and attach the data descriptor (and associated frame) to the tail of the work queue in the normal manner. It will also increment its work counter, causing an interrupt to be generated, as already described.

18. When a work queue interrupt routine is generated in the destination device, the work queue interrupt service routine starts executing. It obtains the data descriptor and associated packet buffers that were queued in step 17, and check the various fields within the data descriptor to determine the disposition of the frame. In particular, the firmware will inspect the Flags field of the data descriptor and find that it was received from a remote device across the PCI bus and must be queued on one or more local ports. It must then locate the port(s) on which the frame is to be queued.

    If the frame is a unicast, the firmware will then read the HashBkt field of the data descriptor to obtain a pointer to the local hash bucket corresponding to the destination MAC address. The firmware then accesses the hash bucket, and will read the pointer to the local port descriptor associated with the hash bucket (from the LPDPtr field) to obtain the data structure on which the frame should be queued.

    If the frame is a broadcast, the firmware will read the restriction mask data structure to identify the ports on which broadcasts are permitted, and queue the frame on each of the indicated ports. In addition, the firmware will check the Learn bit in the Flags field of the data descriptor, and will create a forwarding tag for the appropriate source MAC address if the bit is set; the SrcHashBkt field of the forwarding tag will be set up from the HashBkt field of the data descriptor.

19. After the target port(s) on which the frame is to be transmitted have been identified, the firmware will queue the frame to the transmit queue(s) maintained on the local port descriptor(s) corresponding to the target port(s). Additional data descriptors are allocated as necessary to handle broadcast frames (with the reference count in the packet buffer being updated as required). As the frame is attached to each transmit queue, the firmware will check the state of the DMA Controller channel associated with that queue, and initiate frame transmit in the

usual manner (from the head of the queue, which may not necessarily correspond to the frame just queued) if possible.

20. In the mean time, the acknowledge counter increment performed by the destination device to the source device in step 16 will cause an acknowledge counter interrupt to be performed to the Switch Processor in the source device. The Switch Processor then begins executing the acknowledge counter service routine. This routine will verify that the acknowledge counter update was valid (by consulting the expansion port descriptor corresponding to the incremented acknowledge counter), and then remove the entry at the head of the queue, updating the expansion port descriptor accordingly. The reference count maintained within the last packet buffer for the queue entry is decremented: if the count goes to zero, the packet buffer chain and the data descriptor are both freed to their respective free pools; otherwise, only the data descriptor is freed, and the updated reference count is simply written back to the packet buffer. Finally, if the packet buffer chain was freed, the MaxBuffers field in the original source local port descriptor is incremented by the count of buffers freed, to restore it to its original value prior to the reception of the frame.

21. The DMA Controller in the remote (destination) device has, during this time, been enabled to transmit the frame. It performs the transmit in the usual manner, traversing the linked-list of packet buffers and copying the data in them into the MAC channel FIFO. This process continues until the entire frame has been transmitted. (If a broadcast is being performed, the DMA Controller will perform the transmit operation for multiple channels concurrently.)

22. When the transmit is completed for a given channel, the DMA Controller will interrupt the Switch Processor in the destination device to signal that frame transmit has been accomplished and the channel should be serviced.

23. The standard transmit interrupt service routine will be invoked in response to the interrupt generated in step 22. The firmware reads the data descriptor from the head of the transmit queue, obtains a pointer to the last packet buffer from the LastPB field of the data descriptor, and reads and decrements the reference count.

24. If the decremented reference count is non-zero, only the data descriptor is freed to the free data descriptor pool, and the updated reference count is written back; otherwise, both the data descriptor and the packet buffer chain are freed to their respective pools.

    The firmware now reads and updates the per-port and per-MAC transmit statistics. The per-port statistics are stored in the local port descriptor; the per-MAC statistics are only updated in the case of unicast frames, in which case the firmware obtains a pointer to the required destination hash bucket from the HashBkt field of the data descriptor.

25. If the packet buffer chain was freed back to the free packet buffer pool in step 24, the firmware will locate the expansion port descriptor corresponding to the source device, and increment the MaxBuffers field in this descriptor by the number of buffers freed up. This has the effect of restoring the buffer allocation limit assigned to the corresponding source device, and allowing further frames to be transferred from the source device if required. This completes the processing of the frame.

Broadcast and multicast frames are handled in essentially the same manner as unicast frames when performing device-to-device transfers, with the primary difference being that steps 10 through 25 are repeated by each destination device to transfer multiple copies of the data descriptor and packet buffers belonging to the frame across the PCI bus. The source ELAN 8x10 consults its broadcast restriction mask, set up statically at configuration time or dynamically by the spanning tree algorithm, to determine the specific local ports and external devices to which the frame may be broadcast. It then queues the same frame for transmission on all of the required local and expansion port descriptor frame queues (using multiple data descriptors, without copying the packet buffers holding the actual frame data). Note that a frame is copied to a destination device only once, regardless of the number of ports on the destination on which the frame is to be ultimately transmitted; the destination is responsible for replicating the frame to the target transmit ports.

Address Learning and Aging

Under normal circumstances, the ELAN 8x10 device acts as a transparent learning bridge, automatically learning new MAC addresses seen in Ethernet frames and associating them with actual physical ports. Address learning is carried out in-line with frame switching whenever a new source MAC address is encountered. The presence of a new source MAC address is indicated by the failure of the hash lookup mechanism to return a hash bucket (or forwarding tag) for a source address lookup. At this point, the ELAN 8x10 Switch Processor will execute the address learning process. If multiple devices are present in the system, learned addresses will be propagated to all of them by means of a simple protocol. All address learning functions are implemented in firmware running on the Switch Processor. Note that only source MAC addresses are ever learned by the ELAN 8x10.

Address learning in a multi-device ELAN 8x10 system takes two steps: local address learning, by the device that received the frame containing the new source MAC address; and remote address learning, by all of the other devices in the system. In a single-device ELAN 8x10 system, only the local address learning process is performed, as there are no remote devices.

Local Address Learning

Address learning within the ELAN 8x10 device that first encountered the new source address takes the following steps:

1. The Switch Processor inspects the results of the source hash bucket search performed by the hash lookup mechanism and determines that it has failed, indicating that the source MAC address is not present in the system.

2. The Switch Processor then checks the global address-learning-enable flag (a variable in the Switch Processor operating environment) to verify that address learning is enabled. If this flag is clear, indicating that address learning is disabled, the Switch Processor does not attempt to learn the source address, but simply skips to step 7. Otherwise, the Switch Processor proceeds to step 3.

3. The address table limit counter (another variable in the Switch Processor operating environment) is now checked to determine whether the address table is full, i.e., the pre-configured maximum number of addresses have been learned. If the limit is zero, indicating that the address table is full and no more addresses can be learned, the Switch Processor cannot learn the new source address; it skips to step 7.

4. The Switch Processor has determined that address learning is enabled and possible. It now allocates an empty hash bucket from the free hash bucket pool and fills in the new source MAC address and the pointer to the source local port descriptor into the MACAddr and LPDPtr fields, respectively, of the blank hash bucket. The remainder of the statistics and flags fields will have been set up already when the blank hash bucket was placed in the free pool, either by the initialization process or by the aging process. The Switch Processor then updates the AgeControl field in the hash bucket.

5. The hash bucket must now be inserted into the hash table at the proper location. To do this, the Switch Processor computes the address of the appropriate entry within the hash pointer array. It then copies the contents of the 24 LSBs of this entry into the NextHB field of the newly created hash bucket, and writes the memory address of the hash bucket into the entry into the hash pointer array. This completes the insertion of the hash bucket into the linked-list of hash buckets pointed to by the hash pointer array entry.

6. Once the MAC address has been learned by the ELAN 8x10, all of the other devices in the system must be notified of the existence of the new address so that they can learn the new address as well (by means of the remote address learning mechanism). This is done quite simply by forcing a broadcast of the Ethernet frame, regardless of whether the frame was a unicast or broadcast, and setting the Learn bit in the data descriptor corresponding to the frame. The Switch Processor thus sets the Learn bit in the Flags field of the data descriptor, writes the memory address of the newly learned hash bucket into the HashBkt

field in the descriptor, and invokes the normal frame broadcast process to direct the frame to all of the remote devices in the system (and, as a side effect, transmits the frame out all the local ports as well). This completes the local address learning procedure.

7. If the MAC address cannot be learned by the ELAN 8x10, the Switch Processor locates the default hash bucket for the physical port on which the frame arrived, and uses this as the pointer to the source hash bucket instead. The remainder of the processing for the frame follows the normal switching requirements. As the MAC address has not been learned, subsequent frames received by the ELAN 8x10 that are directed towards this MAC address will always be flooded.

When the local learning process is complete, the source MAC address will have been represented by a new hash bucket in the address table (assuming that the learning was successful). The remainder of the frame processing is unchanged.

Remote Learning Process

In a multi-device system, it is necessary for all devices in the system to maintain a consistent view of the address table, in order to properly forward frames between devices in an efficient manner. Thus a learning process must be followed by all of the remote devices in the system when a local ELAN 8x10 signals that it has encountered (and learned) a new source MAC address. The remote learning process is triggered whenever a data descriptor corresponding to an Ethernet frame transferred over the PCI bus is found to have the Learn bit set in its Flags field, and results in a forwarding tag for the MAC address being placed into the address table.

The remote learning process takes the following steps:

1. The Switch Processor checks the Flags field of the data descriptor copied from the remote device (the source of the Ethernet frame transferred across the PCI bus) to see if the Learn bit is set. If the Learn bit is not set, the Switch Processor does not perform any address learning, but simply processes the frame in the normal manner.

2. The Switch Processor then checks the global address-learning-enable flag: If this flag is clear, indicating that address learning is disabled, the Switch Processor declares an error, as the remote device should not have attempted to learn the MAC address. Otherwise, it goes to step 3.

3. The address table limit counter is now checked to determine whether the maximum number of addresses have been learned. If the counter is zero, the address table is full: the Switch Processor skips the complete remote learning process and simply goes on to process the frame in the normal manner. Otherwise, it proceeds to step 4.

4. Remote address learning has been determined to be possible. The Switch Processor allocates an empty 16-byte memory block from the free data descriptor pool and formats it as a forwarding tag. It fills in the new source MAC address (read from the IEEE 802.3 source address field of the Ethernet frame), the pointer to the master hash bucket in the remote device (read from the HashBkt field of the data descriptor), and the chip number (obtained from the SrcChip field of the data descriptor) into the MACAddr, SrcHashBkt, and ChipNum fields, respectively, of the blank forwarding tag. In addition, it fills in the MulticastMask and FTFlags fields of the forwarding tag with their default values.

5. The forwarding tag must now be inserted into the hash table at the correct point. To do this, the Switch Processor hashes the MAC address and computes the index into the hash pointer array (in the same manner as implemented by the hash lookup mechanism). It then converts the index into the actual address of the appropriate entry within the hash pointer array, reads the 24 LSBs of this entry, and writes the 24 LSBs into the NextFT field of the newly created forwarding tag. Finally, it writes the memory address of the forwarding tag into the hash pointer array, which completes the insertion of the forwarding tag into the address table.

6. The creation and insertion of the forwarding tag, carrying with it the necessary information required to direct frames targeted at the given MAC address to the remote device, completes the remote learning process. The Switch Processor now continues with the rest of the frame switching process.

Note that the number of forwarding tags created within any of the ELAN 8x10 devices in a system must equal the sum of the numbers of hash buckets created in all of the other devices in the system (each new hash bucket created within any remote device results in a forwarding tag being created within this device). In addition, the sum of the forwarding tags and the hash buckets created by any one device cannot exceed the pre-set maximum number of addresses that can be learned by the system.

Default Hash Buckets

A special mechanism is implemented to deal with the case when a new source MAC address cannot be learned due to some reason (e.g., when the pool of available hash buckets is exhausted). This mechanism is necessary as the source hash bucket pointer that is used at various points during the switching process must always point to a valid hash bucket data structure, as this pointer is utilized by the switching firmware to update statistics and so on. In addition, the local port descriptor pointer within the hash bucket must also be valid, and pointing to the correct local port descriptor corresponding to the physical port upon which the frame arrived.

To deal with the above case, the initialization process creates eight dummy (default) hash buckets within the Switch Processor operating environment at system start-up.

These hash buckets are set up to correspond to the eight physical ports, respectively (i.e., their local port descriptor pointers are set up to point to the various ports). These hash buckets are **not** placed into the address table, and their MAC address fields are invalid. They will never be found by the hash lookup mechanism during the source and destination MAC address resolution process.

When a particular source MAC address is not found in the address table, and cannot be learned due to some constraint, the Switch Processor instead sets up the source hash bucket pointer placed into the data descriptor and used during the switching process to point to the appropriate default hash bucket corresponding to the physical port upon which the frame arrived. This source hash bucket pointer hence indicates a valid hash bucket (even though it is one with the wrong MAC address, and invalid statistics fields); thus the remainder of the switching and frame forwarding firmware will continue to operate as normal. Statistics updates may be done to the default hash bucket; as the statistics fields within the hash bucket are invalid anyway, these statistics updates will be irrelevant. In this manner, the Switch Processor can ensure that the source hash bucket pointer passed to the remainder of the system will always point to a valid hash bucket, even if learning is disabled or not possible.

Address Aging Process

To reclaim the address table resources that have been allocated to network entities that are currently inactive or non-existent, the ELAN 8x10 implements an automatic address table aging mechanism. In essence, this mechanism causes a given address table entry to be deleted if a frame has not been received from the corresponding host computer (or other originator of traffic) for a specified period of time. The mechanism also implements a means whereby a topology change in the network (i.e., the movement of one or more hosts between ELAN 8x10 ports) can result in the removal of the outdated address table entry or entries and the substitution of new ones to reflect the new state of the network.

In general, aging is carried out using the AgeControl field in the hash bucket data structures. This field is updated with the current value of the 32-bit real-time system clock (represented by the CLOCK register pair in the Switch Processor) whenever a valid frame has been received from the corresponding source entity. The AgeControl field thus indicates the last time at which the source was active. The aging process periodically scans the entire address table, comparing the AgeControl fields in the various hash buckets with the current value of the real-time clock; if the difference exceeds a pre-set threshold, then the hash bucket is removed and returned to the free hash bucket pool. In a system containing more than one ELAN 8x10 device, an additional process is followed to ensure that the forwarding tags on all of the other ELAN 8x10 devices corresponding to the deleted hash bucket are removed before the hash bucket is itself eliminated. This is necessary to prevent system corruption due to inconsistent data structures.

The normal aging time used by the ELAN 8x10 is set by the MaxAge variable, and may vary between 2 and 4000 seconds, with a default of 300 seconds. This aging time is used when there are no topology changes, and hash buckets are removed after their corresponding source entities have been silent for the time specified by MaxAge. If a topology change occurs, however, the aging time is set by the ForwardDelay variable, which may also range between 2 and 4000 seconds, but defaults to 2 seconds. The purpose of the ForwardDelay time is to quickly remove all inactive hash buckets without forcing the entire table to be purged, allowing the outdated address table entries resulting from the topology change to be replaced by the new ones. This mechanism has the benefit that MAC addresses from which heavy traffic is being received (but have not shifted physical ports) will remain untouched in the address table, while at the same time the altered state of the network will be rapidly re-learned. The net result is intended to minimize the impact of topology changes in a busy network. (Note that this process is also aimed at permitting simple implementation of the IEEE 802.1D standard.)

In some situations, it may be necessary to 'lock' certain address table entries permanently into the address table (i.e., to partially defeat aging without completely turning it off). Each hash bucket contains a flag (referred to as the NoAge flag) that may be set; if set, the aging process will be prevented from removing the hash bucket or its associated forwarding tags, if any. (Note that the AgeControl field within the hash bucket will still be updated whenever a frame is received with the corresponding source address.)

Three main components make up the aging process:

1. the work interrupt service routine that performs the primary frame dispatching tasks is responsible for updating the AgeControl field in the source hash bucket whenever a frame is received from the corresponding source. The work interrupt service routine also notices when a mismatch occurs between the physical port indicated by the source hash bucket and the actual physical port on which the frame arrived, and flags a topology change notification to cause the address table to be updated.

2. a background scanning task is implemented to periodically check the AgeControl fields in all the hash buckets and purge any that have been aged out (i.e., whose source entities have not been active for some time). The background scanning task is also responsible for notifying the remainder of the chips in a multi-device system when a hash bucket must be removed, to allow the other chips to delete the corresponding forwarding tags. The notification is performed via a special message (called an aging message) sent to the other chips.

3. a special routine, invoked whenever an aging message is received from another device in the system, that actually removes the forwarding tag referenced by the message.

Of the three components, the first has already been described in the sections dealing with frame switching and inter-device frame transfers (see above). The other two will be described in detail in this section.

Background Scanning Task

The background scanning task is invoked periodically (during idle periods, when frame switching is not occurring) to search the entire address table for hash buckets that need to be removed. The scan period is configurable, and defaults to 1 second.

The first step executed by the background scanning task is to determine the limit imposed on the age of hash buckets. If a topology change was not detected, this limit is simply the contents of the MaxAge variable. If, however, a topology change occurs, this limit is set to the contents of the ForwardDelay variable for a period equal to the ForwardDelay time, after which it reverts to the MaxAge time. Thus, for example, if the default values of 300 seconds and 2 seconds, respectively, are used for MaxAge and ForwardDelay, then for the 2 seconds following a topology change the age limit is set to 2 seconds, after which the age limit is set to 300 seconds. After the limit has been set, the CLOCK register pair is read and preserved as the current time against which the AgeControl fields in the hash buckets will be compared.

The next step is to scan the address table. To implement this, the hash pointer array is simply scanned linearly, starting at the first pointer and terminating at the last. If any pointer is found to be non-NULL, this indicates that a list of one or more hash buckets and/or forwarding tags is attached to that entry; the list is then sequentially scanned until no more elements remain in the list, after which the scanning returns to the next hash pointer array entry. After the last entry in the hash pointer array has been checked, the background scanning task returns.

For each hash bucket or forwarding tag found, the Flags field is checked to determine whether it is a hash bucket, and also whether it can be aged out. If so, the AgeControl field of the hash bucket is read and subtracted from the current time to give the age of the hash bucket, and the result is compared to the age limit. If the comparison fails (i.e., the age of the hash bucket is greater than the limit), then the hash bucket is removed from the linked list; otherwise, the hash bucket is left untouched and scanning proceeds.

Once a hash bucket that has aged out has been located and removed from the address table, it must be freed. This is not done immediately, however, as there may be other devices in the system with forwarding tags that reference this hash bucket; if the hash bucket is destroyed before deleting these forwarding tags, then incoming frames on other devices that are directed to the corresponding MAC address may cause inconsistencies and failures.

Instead, all of the hash buckets that have been removed from the address table are placed on a separate linked list (referred to as the pending free hash bucket list). In a multi-device system, a notification message is sent to all other devices in the system for each hash bucket placed on this list, requesting that any forwarding tags corresponding to the hash bucket be removed. To avoid the complexity of a request/acknowledge protocol, the pending free hash bucket list is simply maintained for a short time (1 second), which gives the other devices ample time to receive the notification message and remove the specified forwarding tags from their address tables. Following this time, the entire pending free hash bucket list is deleted and the hash buckets returned to the free hash bucket pool.

The format of a notification message is given in the following subsection.

Forwarding Tag Removal Task

The forwarding tags corresponding to a removed hash bucket must also be removed from the system, otherwise system errors can occur due to transferred frames referencing inconsistent address tables. The removal of forwarding tags is triggered by the notification messages sent out whenever hash buckets are aged out by any device in the system.

The format of the notification message follows that of a standard data descriptor, but with NULL FirstPB and LastPB pointers, and a special Flags field, as given below:

31 24	23 16	15 8	7 0	Byte Offset
Flags	NextDD			0
Unused	0x000000			4
Unused		HashIndex		8
Unused	HashBktAddr			12

NextDD:
24-bit pointer to next data descriptor in linked-list of data descriptors.

Flags:
8-bit data descriptor flags field, formatted as below.

HashIndex:
16-bit index of hash pointer array at which the hash bucket (and forwarding tags) may be found.

HashBktAddr:
24-bit pointer to hash bucket being removed.

The NextDD field is set up properly when the message descriptor is queued. The Flags field is formatted as shown below:

7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0

The encoding of the Flags field indicates that this is a message descriptor that is notifying the recipient of the need to delete the forwarding tag corresponding to an aged-out hash bucket.

The HashIndex and HashBktAddr fields are expected to be used by the recipient in locating the forwarding tag that must be removed. The HashIndex field is simply the index of the hash pointer array at which the hash bucket itself is located, which will be identical to the index at which the forwarding tag will be located in the remote device. The HashBktAddr value is the 24-bit address of the hash bucket being deleted, and will be compared to the corresponding field in the forwarding tag to ensure that the correct forwarding tag will be removed.

The notification messages described above are treated as standard data descriptors and transferred to the remote device by the owner of the hash bucket using the normal PCI expansion port transfer protocol (with the exception that no packet buffers are transferred, as the FirstPB pointers in the message descriptors are NULL). Upon receipt of a notification message, a remote device (ELAN 8x10 or other) must access its own address table, locate the proper forwarding tag using the data specified in the message, and remove and de-allocate the forwarding tag.

Buffer and Queue Management

The Switch Processor implements all of the buffer and queue management in an ELAN 8x10 device, using mechanisms implemented as firmware routines. Buffer management is carried out by means of a collection of per-port buffer allocation limits, plus the global broadcast frame limit that prevents excessive use of system resources by high levels of broadcasts. Queue management (on the transmit queues) is performed via a background task that periodically monitors the different transmit queues and handles housekeeping tasks as required. In addition, broadcast rate limiting is implemented to avoid system overload during broadcast storms or other malfunctions.

All of the parameters associated with the buffer and queue management functions implemented by the ELAN 8x10 device are configurable by means of system variables. These variables may be set statically to default values at boot time, or altered during normal system operation as required.

Buffer Management

As already mentioned, two forms of buffer management are performed by the ELAN 8x10 device. The first mechanism attempts to distribute the available frame buffer space equitably across the eight ports in the device, such that no one port will be permitted to starve the others of buffer space to hold incoming frames. The second mechanism places a limit on the number of broadcast or multicast frames that will be buffered by the system at any one time; this limit prevents unicast traffic from being unfairly affected due over-consumption of system resources by large amounts of broadcasts or multicasts.

Per-port Buffer Allocation Limits

Per-port buffer allocation is controlled using the 16-bit MaxBuffers fields in the local port descriptor structures. This field is set up at initialization time with the maximum number of (fixed-size) packet buffers that the given local port is allowed to consume. Whenever a valid frame is received on a MAC channel, the Switch Processor firmware subtracts the number of packet buffers required to hold the frame from the MaxBuffers field in the corresponding local port descriptor. If the result is less than or equal to zero, it indicates that the port does not have sufficient buffer capacity to hold the frame; the frame is then discarded. If the result is greater than zero, the frame is accepted, and the MaxBuffers field is updated to reflect this fact. The MaxBuffers field is subsequently incremented (again, by the number of packet buffers used to contain the frame data) when the frame is either transmitted, transferred to another device across the PCI bus, or discarded.

This method of limiting buffer allocation has the advantage that there is no need to physically reserve portions of memory for specific ports. All of the available free packet buffers are placed into a single, common, global free pool, and allocated by the various ports as required. The MaxBuffers fields in the local port descriptor structures serve as 'soft' limits, preventing any port or group of ports with high incoming traffic from starving the others. It is not necessary that all ports be assigned the same buffer limit; the MaxBuffers fields may, for example, be set higher for ports that are expected to carry high traffic, and lower for the others. It is required, however, that the sum of the MaxBuffers fields of all of the local ports be no greater than the total number of packet buffers in the system (minus approximately 200 buffers to compensate for system overheads and temporary overruns). The permissible range of the MaxBuffers fields is between 44 and 32767, depending on the available packet buffer memory.

Buffer allocation limits are also maintained separately for each external ELAN 8x10 (or other) device in the system that is connected to this device via the PCI bus. These buffer allocation limits are located in the MaxBuffers fields of the expansion port descriptors, and operate in substantially the same way as the corresponding fields in the local port descriptors, but control frame transfers from other devices to this ELAN 8x10 device. One significant difference is that frames are not discarded when the MaxBuffers field in an expansion port descriptor becomes less than or equal to zero;

instead, the Switch Processor firmware will simply cease to transfer frames from the corresponding remote device across the PCI bus. The effect is that external devices that attempt to consume too much of the ELAN 8x10 device's local memory will be forced to stop, and instead accumulate frames in their own local storage. This imposes fairness between devices.

If the BackPEn bit is set in the PortFlags field of a local port descriptor, and a high traffic condition causes its MaxBuffers field to become less than or equal to zero, then backpressure flow control will be started for that port. In this case, the Switch Processor firmware will enable the MAC channel to begin jamming the medium by continuously transmitting a bit pattern. The bit pattern is interrupted periodically to prevent downstream jabber timers from locking out the ELAN 8x10 MAC interface; a standard interframe gap is inserted at these times, after which the jamming will begin again. If a collision is encountered during a backpressure jam, the MAC channel logic on the ELAN 8x10 will perform the normal collision jam process, insert a normal interframe gap, and then restart the jam (i.e., no backoff will be performed). The backpressure jam will be maintained until the Switch Processor detects that the MaxBuffers field has risen above a programmable threshold, at which time the Switch Processor firmware will shut off the jam and resume normal operation on that channel. Note that if a normal transmit frame is queued for transmission out a jamming ELAN 8x10 port, then the jamming will temporarily cease, the frame will be transmitted (after waiting an interframe gap), and the jamming will restart (again, after a standard interframe gap).

Note that the MaxBuffers variables in the local and expansion port descriptors are maintained by the Switch Processor firmware in cached data space, and thus cannot be updated by direct access over the PCI bus interface. Instead, the messaging interface must be used to modify these variables during system operation, if desired. This function is TBD.

Broadcast Frame Limit

Broadcast and multicast frames consume a considerable amount of memory resources of an ELAN 8x10 system for a potentially long time. If, for example, a port is receiving a high rate of broadcasts, then the broadcast frames (which must be sent out over all the other ports in the system) may have to be buffered for a long time, and will also require a large number of data descriptors in order to occupy all of the local port transmit and expansion port transfer queues in the system. A programmable broadcast frame limit is therefore provided to permit the system implementer to exercise some control over the amount of system resources that may be consumed by broadcast and multicast traffic.

The broadcast/multicast frame limit is implemented by the Switch Processor firmware using a single global MaxBcstFrames variable in the cached data space. This variable is set up initially to the maximum number of broadcast frames permitted to be buffered by the ELAN 8x10 device. Whenever a newly received broadcast or multicast frame is detected by the switching code executed by the Switch Processor, the MaxBcstFrames

variable is decremented by 1; if the result is equal to or less than zero, the broadcast or multicast frame is discarded, as the broadcast frame limit has been exceeded. If the result is greater than zero, the broadcast/multicast frame is accepted for processing in the usual manner (i.e., queued to the target transmit ports), and the MaxBcstFrames variable is updated accordingly. Whenever a broadcast or multicast frame has been completely transmitted or transferred (i.e., the packet buffer chain holding the broadcast or multicast frame has been returned to the free packet buffer pool), the MaxBcstFrames variable is incremented, allowing additional broadcast or multicast frames to be accepted. Note that the broadcast/multicast frame limit is applicable to frames received over the PCI expansion port, as well as frames received over local MAC ports.

The MaxBcstFrames variable is maintained by the Switch Processor firmware in cached data space, and is hence not accessible directly over the PCI bus interface. The messaging interface must be used to update it. This operation is TBD.

Transmit Queue Management

The ELAN 8x10 implements two mechanisms to monitor and manage the transmit queues (i.e., the queues holding frames waiting to be transmitted out the local MAC ports) in the system.

Transmit Queue Purging

The first mechanism is intended to ensure that frames do not persist in the transmit queues for too long before being transmitted. This can happen if the medium associated with a MAC channel is continually busy due to a malfunctioning system or a jabbering upstream host or switch, or if some hardware fault is present that prevents frames from being transmitted. In this case, it is possible that the transmit queue belonging to the MAC channel will fill up with frames that will never be sent out, and thus their packet buffers will never be released back to the free pool. In extreme cases, the entire buffering in the system can ultimately be consumed by these blocked frames, bringing the system to a halt.

To detect such a condition, the ELAN 8x10 employs a mark-and-sweep algorithm on the entries in the various transmit queues, using the lower 4 bits (i.e., the TypeSpecific subfield) of the 8-bit Flags fields of the data descriptors placed on the transmit queues to hold a marker. This field is initially set to zero by the switching firmware when frames are queued to the transmit queues, and the data descriptors pointing to the frame data are linked into the linked-lists forming the queues. The ELAN 8x10 Switch Processor then executes a background task that runs periodically at a pre-set interval (normally 200 milliseconds). This background task sweeps over all of the transmit queues in the device, inspecting the data descriptors (if any) at the heads of the queues. For each data descriptor, the background task checks the Flags field of the descriptor, as follows:

- if the TypeSpecific subfield of the Flags field is zero, it is changed to a 1, but nothing more is done with that descriptor; the background task proceeds to the next transmit queue.

- if the TypeSpecific subfield of the Flags field is non-zero, the descriptor must have been marked in a previous sweep, and hence must have been present on the transmit queue for at least one sweep interval. Thus the transmit queue has been inactive for at least that long; the descriptor (and potentially the frame data, if no other queues are pointing to the same frame) is removed from the queue and returned to the free pool.

The mark-and-sweep algorithm thus detects when a frame has been present in a transmit queue for an excessive amount of time by marking it in one sweep, and detecting the marked frame in the next sweep. If the frame was transmitted in the interval between sweeps, it will be deleted from the transmit queue; the next entry in the transmit queue will be marked in its turn, preparatory to the next sweep. If a transmit queue is blocked, therefore, frames will be prevented from remaining in it indefinitely, and their memory resources will be reclaimed.

Transmit Queue Restart

In certain rare cases, it is possible for frames to remain in a transmit queue without any transmit attempt being made (even if the MAC channel is completely idle). To handle this situation, the Switch Processor executes a periodic background task (referred to as the transmit queue restart process) that is invoked at short intervals during idle periods, when the Switch Processor is not busy with frame switching. This background task scans all of the MAC channels; for each idle MAC channel, it checks to see if a corresponding non-empty transmit queue exists. If so, the background task simply reads the data descriptor at the head of the transmit queue and attempts to start transmission of that frame over the MAC channel; otherwise, the background task continues on to the next channel. When all eight channels in the ELAN 8x10 have been scanned, the background task terminates, and waits until it is invoked again.

The transmit queue restart process is performed at the same time as the mark-and-sweep algorithm that purges stalled entries from the transmit queues.

Broadcast Rate Limiting

It is desirable to provide some form of limit on the rate of throughput of broadcasts handled by an ELAN 8x10 system; this limit should be set to allow a relatively small, 'normal' level of broadcasts, but prevent the excessive rates of broadcasts that can occur during malfunctions such as broadcast storms. It should be noted that the rate at which broadcasts occur (not the total number of broadcasts being handled by the system) must be limited; the limit is therefore set in terms of broadcast frames accepted

and processed by the system per second. It should also be noted that multicasts are not subject to this limit.

The ELAN 8x10 implements broadcast rate limiting via firmware running on the Switch Processor, using a combination of operations performed by the frame switching tasks coupled with a periodic background process. The broadcast rate limit is actually implemented by the 16-bit BcstRem variable in the Switch Processor firmware data space. BcstRem is initially set to a programmable value, defined by the 16-bit BcstLimit variable (which in turn is initialized to a default during system boot time, and which may be altered during system operation to modify the permissible broadcast rate).

Whenever a broadcast frame (i.e., one with the destination MAC address set to all-ones) is being processed by the switching firmware, the Switch Processor decrements the BcstRem variable by 1 and checks the result. If the result is less than or equal to zero, the Switch Processor simply discards the frame, as the broadcast rate limit has been exceeded; otherwise, it updates the BcstRem variable with the decremented value, and accepts the broadcast frame for normal processing and eventual transmission. The BcstRem variable is never incremented by the switching firmware. Hence, when it goes to zero, no more broadcast frames will be accepted by the ELAN 8x10 device.

A background task is periodically executed every ten milliseconds by the ELAN 8x10 Switch Processor under timer control. The purpose of this background task is to reset the BcstRem variable to the contents of BcstLimit, thus permitting broadcasts to continue for the next ten millisecond period. The value of the BcstLimit variable is therefore directly equal to the number of broadcast frames that will be forwarded by the system every ten milliseconds. The BcstLimit variable may be set to any value between 0 and 32,767. Setting this variable to 100, for example, results in a maximum average broadcast rate of 10,000 frames per second; setting it to zero shuts off broadcasts completely, while setting it to its maximum value (32,767) effectively removes the broadcast rate limiting feature.

It should be noted that the broadcast rate limiting feature allows a short burst of broadcast every 10 milliseconds until the limit is reached, at which point all further broadcast frames are discarded.

Statistics Collection

The ELAN 8x10 device collects a number of SNMP and RMON MIB statistics during normal operation. These statistics are maintained in the local port descriptors, the port descriptor counter structures, and the hash buckets. Some internal statistics are also held in the global memory area.

The locations and sizes of the individual statistics counters are described in the sections above that deal with the various data structures. A summary of all of the

statistics maintained by the ELAN 8x10 device, as well as the data structures in which they are located, is given in the following tables. Note that all of the counters are 32 bits in size, and roll over (according to the SNMP standard) when they are incremented beyond their maximum count.

The counters maintained by the ELAN 8x10 device on a per-port basis are described in the following table:

Per-port Counters		
Name	**Location**	**Description**
RXUcstFrames	PortDesc	Valid unicast frames received on this port
RXUcstOctets	PortDesc	Valid unicast octets received on this port
RXBcstFrames	CtrBlk	Valid broadcast frames received on this port
RXBcstOctets	CtrBlk	Valid broadcast octets received on this port
RXMcstFrames	CtrBlk	Valid multicast frames received on this port
RXMcstOctets	CtrBlk	Valid multicast octets received on this port
RXFloodFrames	CtrBlk	Valid frames received and then flooded to all ports
RXFloodOctets	CtrBlk	Valid octets received and then flooded to all ports
AlignErrors	CtrBlk	Frames received on this port with alignment errors
CRCErrors	CtrBlk	Frames received on this port with CRC errors
DribbleErrors	CtrBlk	Frames received on this port with dribble bit errors
LongErrors	CtrBlk	Oversize frames received with good CRCs
JabberErrors	CtrBlk	Oversize frames received with bad CRCs
ShortErrors	CtrBlk	Undersize frames received with good CRCs
RuntErrors	CtrBlk	Undersize frames received with bad CRC
MACErrors	CtrBlk	Frames lost due to internal MAC/FIFO errors
DropErrors	CtrBlk	Frames lost due to resource limits (out of buffers)
RXErrorOctets	CtrBlk	Errored octets received
RXFrames64	PortDesc	64-byte frames received (valid or errored)
RXFrames65	PortDesc	65-127 byte frames received (valid or errored)
RXFrames128	PortDesc	128-255 byte frames received (valid or errored)
RXFrames256	PortDesc	256-511 byte frames received (valid or errored)
RXFrames512	PortDesc	512-1023 byte frames received (valid or errored)
RXFrames1024	PortDesc	1024-1518 byte frames received (valid or errored)
TXFrames	PortDesc	Valid unicast frames transmitted out this port
TXOctets	PortDesc	Valid unicast octets transmitted out this port
TXDefers	PortDesc	Frames subject to deference on first transmit
TXErrors	CtrBlk	Transmit frames lost due to internal errors
SingleCollide	CtrBlk	Frames transmitted after a single collision
MultCollide	CtrBlk	Frames transmitted after multiple collisions
LateCollide	CtrBlk	Frames discarded due to late collisions
CollideAbort	CtrBlk	Frames discarded due to excessive collisions

In the table above, note that "PortDesc" refers to the local port descriptor structure corresponding to the associated MAC channel, while "CtrBlk" denotes the port

descriptor counter structure for that MAC channel. The per-port statistics maintained in the port descriptor counter structures may be retrieved at any time by direct accesses to these structures via the PCI bus interface; however, the statistics in the port descriptors themselves are cached by the Switch Processor, and hence can only be retrieved via the message interface.

The ELAN 8x10 device also maintains counters on a per-MAC-address (or per-host) basis, with a separate set of counters being maintained for each MAC address that has been learned by the system. These counters are held in the hash bucket associated with the learned MAC address. (Note that forwarding tags do not contain any counters; hence only one set of counters will be maintained for each MAC address in a multi-device system, rendering retrieval of the per-host statistics relatively straightforward.) These counters are described in the following table:

Per-host Counters	
Name	**Description**
RXHUcstFrames	Valid unicast frames received from this host
RXHBcstFrames	Valid broadcast frames received from this host
RXHMcstFrames	Valid multicast frames received from this host
RXHFloodFrames	Valid frames received from this host and then flooded
RXHErrorFrames	Errored frames received from this host
RXHOctets	Octets received from this host, valid or invalid
TXHFrames	Valid unicast frames transmitted to this host
TXHOctets	Valid unicast octets transmitted to this host
LastRX (AgeControl)	Time at which frame was last received from this host
CreateTime	Creation time of hash bucket

The per-host statistics may be retrieved at any time via direct accesses to the hash buckets in the address table through the PCI bus interface, or using the message interface.

NOTES pertaining to statistics collection:

- the maximum number of octets that can be counted for any received frame (whether erred or received without error) is decimal 65,535 (which is 0xffff). This limits the number of bytes that will be counted if receiving a long or jabber frame and maintained as variable “RXErrorOctets” in the above table.

Messaging Protocol

This section is TBD.

Spanning Tree Operation

This section is TBD.

RTOS Operation

This section is TBD.

UDP/IP Protocol Stack Operation

This section is TBD.

SNMP Operation

This section is TBD.

Device Interface Via PCI

This section describes how external devices that are not ELAN 8x10s (or members of the ELAN family) may be interfaced to the ELAN 8x10 device via the PCI bus. In general, such devices must follow the defined ELAN 8x10 messaging and frame transfer protocol for smooth operation. If the protocol is followed, then the external devices appear to the ELAN 8x10 chip as members of the ELAN family; they may then participate in all aspects of normal operation, including frame transfers, address learning and aging, messaging, and statistics gathering.

Frame Transfer to Non-ELAN Devices

This section is TBD.

Participation in Address Learning

This section is TBD.

Participation in the Messaging Protocol

This section is TBD.

Accessing Variables and Statistics

This section is TBD.

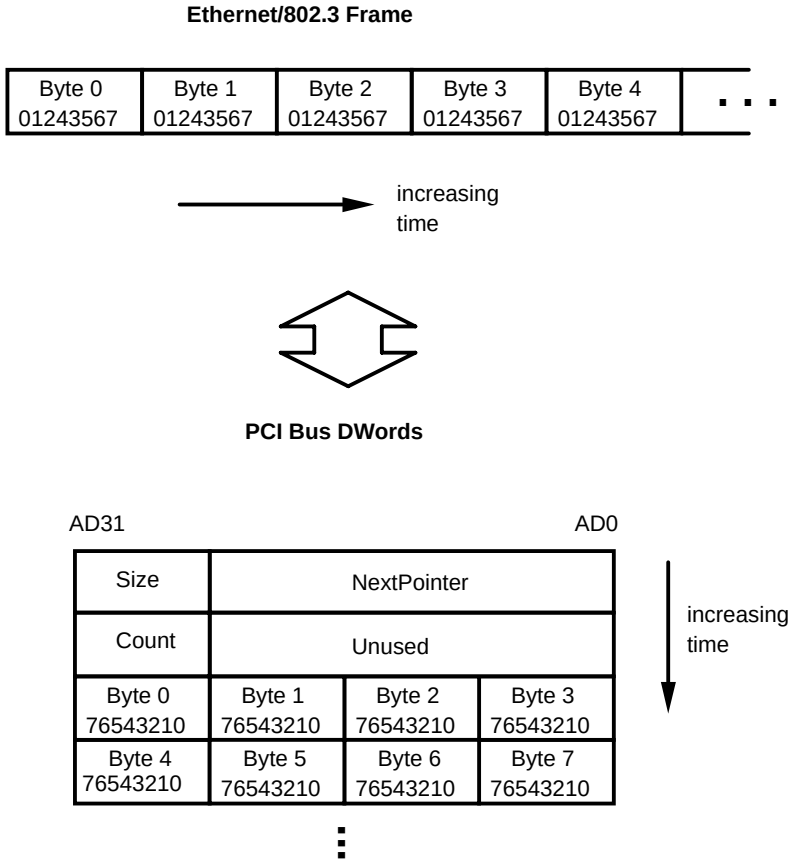
Byte Ordering

The ELAN 8x10 internally uses a **little-endian** byte ordering scheme to store frames and data structures in local memory, and to transfer frames and communication messages across the little-endian PCI bus. In this scheme, the least significant bit of any 32-bit data word appears on the least-significant bit of the memory and PCI data buses, and is also placed within the least-significant byte address in memory. If less than four bytes are being transferred over a data bus, they are right-aligned on the bus, and placed in the memory in increasing address order.

The Ethernet standard, however, specifies a **big-endian** byte ordering scheme, which is the reverse of the ordering used by the ELAN 8x10 device. To properly align the various fields within a received Ethernet frame for handling by the Switch Processor, therefore, the ELAN 8x10 performs a big-endian to little-endian conversion in the MAC interface block during receive, and a little-endian to big-endian conversion during transmit.

The following diagram illustrates the default mapping between an Ethernet frame and packet buffers transmitted across the PCI bus or stored in local memory:

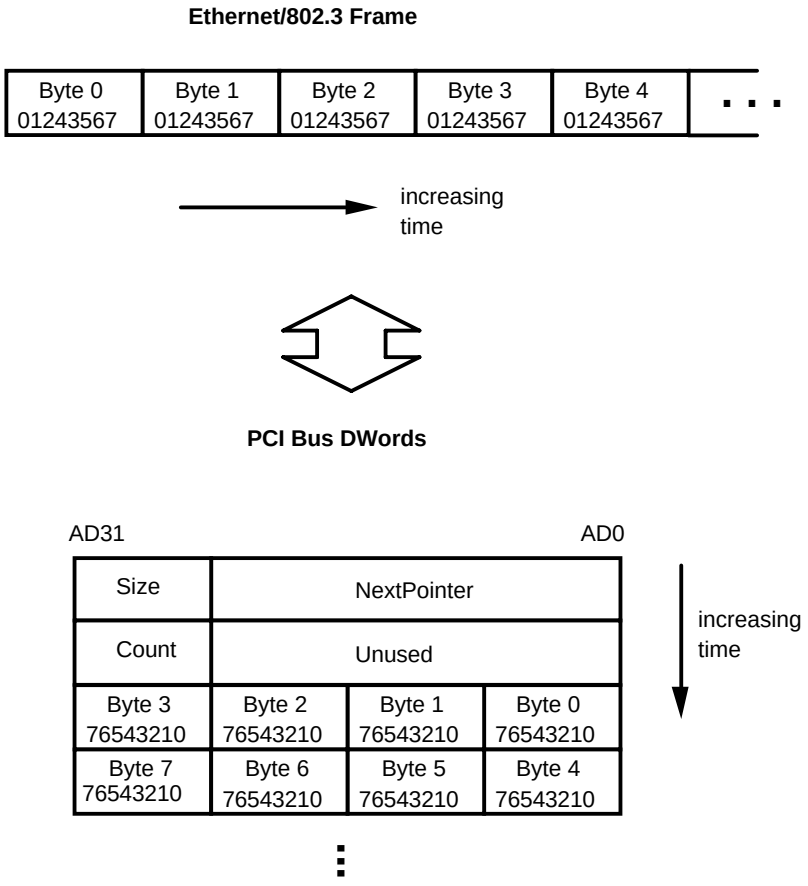
Default Byte Ordering



To permit interfacing of non-ELAN 8x10 devices via the PCI bus, this byte ordering scheme may be selectively modified using the MWBE, MRBE, SWBE, SRBE and EPBE byte swap enable bits in the host control register. The MWBE and MRBE bits control the PCI bus master, and cause all data (payload and header information for packet buffers, as well as data descriptor contents) to be statically byte-swapped prior to writing them to or reading them from a PCI bus slave (i.e., as a result of DMA Controller transfers). Similarly, the SWBE and SRBE bits force the PCI bus slave logic in the ELAN 8x10 to swap data words written to and read from internal registers and memory by an external master. Finally, the EPBE bit, if set, disables the big-endian to little-endian conversion performed on the Ethernet frame information by the MAC interfaces, without affecting the byte ordering of the rest of the data structures or the packet buffer headers.

When the EPBE bit is set, the alternate byte ordering scheme used is shown in the figure below (assuming that the MWBE, MRBE, SWBE and SRBE bits retain their default values):

Alternate Byte Ordering



ELAN 1x100 / ELAN 8x10 Expansion Bus Data Transfer Rates

This section describes what the expected PCI expansion bus utilization is as a function of the following system parameters:

- number of ELAN-1x100 chips in the system
- number of ELAN-8x10 chips in the system
- Ethernet traffic patterns for the 10 Mbit and 100 Mbit links

The PCI bus utilization metric is then applied to determine if the switch configuration and traffic pattern will result in a blocking or non-blocking switch implementation. The data provided allows you to create your own spreadsheet to determine if the switch configuration and traffic mix that is required in your implementation can be supported. It is assumed for the sake of clarity that the traffic mix is sustained (i.e. steady state); since both the PM3350 and PM3351 implement store-and-forward switching with

dynamically allocatable buffers you should get the same result if you take a statistical time average of the data traffic as opposed to an instantaneous frame rate such that the period of averaging is less than or equal to the time it would take to receive frames that would utilize 50% of the maximum allocatable number of packet buffers.

Since the PM3351 Elan 1x100 is a 1-port device the limiting factor in constructing a switch using the chip is the bandwidth that can be supported over the PCI expansion bus. The PCI bus interface that is used on both the ELAN-1x100 and ELAN-8x10 devices together with the ELAN frame transfer protocol that is used results in a maximum **sustainable throughput of 500 Mbit/sec on the PCI bus.**

A switch configuration and traffic mix that requires sustained throughput of l**ess than 500 Mbit/sec on the PCI expansion bus will be non-blocking;** a switch configuration and traffic mix corresponding to a sustained throughput of greater than 500 Mbit/sec will result in a blocking Ethernet switch configuration. The PM3351 and PM3350 chips, if used in a blocking configuration, will do one of two things:

1. if flow control is enabled, then flow control will be enabled until PacketBuffers are returned to the free queue
2. if flow control is not enabled, then frames that are being received from the Ethernet physical layer device will be dropped until there are a sufficient number of PacketBuffers in the free pool in which to format the frame.

The following derivation shows how the rate of traffic being received on the Ethernet physical layer corresponds to the transfer rate as seen on the PCI expansion bus.

GIVEN:

1. Let **Mlp** represent the percentage of frames that are received on the link media that require to be switched across the PCI expansion bus. Let **Mports** represent the number of ports of the device: for the PM3350 this is 8, for the PM3351 it is 1.

   Since the PM3351 is a 1-port device, each frame received on the 100 Mbit link media is switched over the PCI expansion bus so Mlp is 1. However, the PM3350 is an 8-port device so not all unicast traffic may need to be switched across the PCI bus; hence, Mlp may be less than 1 in that case.

2. Let **Mrfr** be the relative frame rate. This is the ratio of the actual frame rate to the maximum Ethernet frame rate. Restating, this is the relative percentage of time that the link media is non-idle that does NOT include the standard interframe gap of 96 bit times.

   If you are modelling a network running at maximum Ethernet frame rates, Mrfr is 1.

3. Let **Mdplx** represent a per-port multiplier that is 0.5 for half duplex operation and 1.0 for full duplex operation.
4. Let **Mpfsz** represent a multiplier that represents the Elan Frame Transfer Protocol overhead with respect to the time it took for the frame to be sent on the link media (assuming a standard interframe gap of 96 bit times and nominal preamble length of 64 bits).

Mpfsz is a function of:

- the link rate (Mpfsz is directly proportional to the link media rate).
- the Ethernet framesize
- the size of the PacketBuffer structure being used on the chip (default size is 80 bytes).

Calculation of **Mpfsz** for a 64 bytes frame for a **100Mbit** link:

link time = 96*10 (ifg) + 64*10 (preamble) + 64*8*10 (data) = 6720 ns

Bits required to have frame read across the PCI bus by one chip:

Data Descriptor = 16 bytes

1 Request counter increment + 1 Acknowledge counter increment = 8 bytes

1 PacketBuffer = 80 bytes

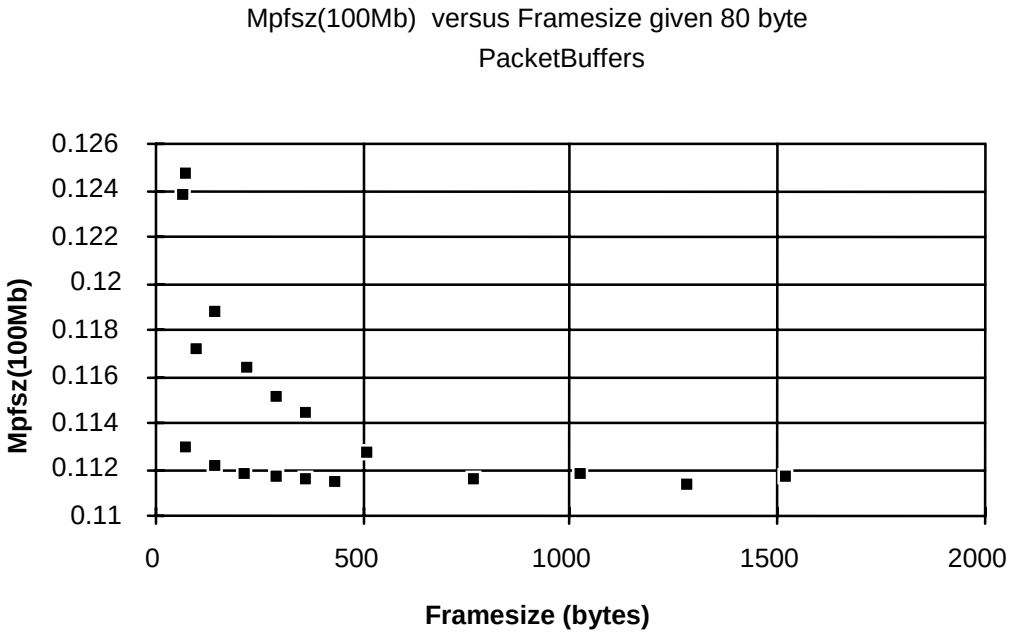
So Mpfsz is (104 bytes * 8 bits/bytes) / 6720 ns = 0.124 bits/ns.

Example 4.2:

Calculation for a 64 bytes frame for a **10Mbit** link:

This is simply 1/10 of the Mpfsz for the 100 Mbit example, or 0.0124 bits/ns

A graph of Mpfsz versus framesize is given is shown below. As can be seen from this figure, the worst case values of Mplfsz is approximately 0.125 bits/ns for 100 Mbit media (PM3351) and 0.0125 for 10 Mbit media (PM3350).



5. Let **Mdest** be the number of chips in the system that will need to read the frame across the PCI expansion bus utilizing the ELAN frame transfer protocol.

   For unicast frames, Mdest is simply 1. For broadcast frames this is given by N-1 where N is the total number of switch chips connected to the PCI expansion bus.

   Whether the frame is unicast or broadcast/mulitcast over the PCI expansion bus is a function of the destination address and whether the source and destination addresses have been learned.

EXAMPLE #1:

Lets examine the worst-case PCI expansion bus data rate for a 2 port -100 Mb, 32-port 10 Mb switch constructed using two PM3351 and four PM3350 chips assuming:

- all ports operating at maximum frame rate (Mfr =1)
- all frames are switched over the PCI bus (Mlp = 1)
- the 100 Mbit ports are full-duplex and the 10 Mbit ports are half-duplex
- all of the 10 Mbit traffic will be switched over the PCI expansion bus (i.e. not switched to local ports on the ELAN-8x10 devices).

The PCI data transfer rate is therefore:

{2*num_pm3351 * 1 (Mlp) * 1 (Mports) * 1 (Mrfr) * 1 (Mdplx) * 0.125 bits/ns (max Mpfsz) * 1 (Mdest for unicast)} + {3*num_pm3350 * 1 (Mlp) * 8 (Mports) * 1 (Mrfr) * 0.5 (Mdplx) * 0.0125 bits/ns (max Mpfsz) * 1 (Mdest for unicast)}

= (0.25 bits/ns + 0.20 bits/ns)

= 450 Mbit/sec

Since this number is less than 500 Mbit/sec the switch configuration will be non-blocking.

EXAMPLE #2:

Lets examine the maximum sustainable broadcast frame for a 2 port -100 Mb, 24-port 10 Mb switch constructed using two PM3351 and three PM3350 chips assuming:

- the 100 Mbit ports are full-duplex and the 10 Mbit ports are half-duplex

PCI bus max transfer rate is 500 Mbit/sec

< {2*num_pm3351 * 1 (Mlp) * 1 (Mports) * 1 (Mrfr_max_learn) * 1 (Mdplx) * 0.125 bits/ns (max Mpfsz) * 3 (Mdest since three other chips in the system)} + {2*num_pm3350 * 1 (Mlp) * 8 (Mports) * 1 (Mrfr_max_learn) * 0.5 (Mdplx) * 0.0125 bits/ns (max Mpfsz) * 3 (Mdest since three other chips in the system)}

< Mrfr_max_learn * {750 Mbit/sec (received on 100 Mbit links) + 300 Mbit/sec (received on 10 Mbit link)}

Solving for Mrfr_max_learn gives:

Mrfr_max_learn = 500 Mbit/s / (1050 Mbit/s) = 0.47

So the PCI expansion bus has a transfer rate adequate to support up to 47% of the frames received being “broadcast to all chips in the system” (this could include frames that are required to be learned).

EXAMPLE #3:

Determine an equation that gives the maximum number of 100 Mbit and 10 Mbit ports that can be used with the PM3351 and PM3350 devices and still allows for a non-blocking switch implementation. Given:

- all destination traffic is unicast

- all 100 Mbit links operate in full duplex
- all 10 Mbit links operate in half duplex

PCI bus max transfer rate is 500 Mbit/sec:

< (num_pm3351 * 125 Mb/s) + (num_pm3350 * 8 * 0.5 * 12.5 Mb/s)

Solving for num_pm3351 and num_pm3350 gives the results that are given in the table found in the “DESCRIPTION” section of the datasheet.

PCI REGISTER/MEMORY ACCESS

The ELAN 8x10 PCI configuration register space, the entire 16 MB local memory address space, plus a subset of the internal device registers, are visible via the PCI interface. The device registers and memory may be accessed to configure the device, monitor status and perform packet buffer transfers.

PCI Configuration Space

The 256 byte PCI configuration register space implemented by the ELAN 8x10 is organized as 64 32-bit registers. Only 19 of these are currently defined. Sixteen registers (at byte addresses 0x00 to 0x3F) are used to implement the standard PCI configuration space header; an additional 3 (at byte addresses 0x40 to 0x48) are used for bus error monitoring and recovery.

PCI configuration registers can only be accessed when the PCI Interface is a target and a configuration cycle is in progress as indicated using the IDSEL input. The format of the PCI configuration register space supported by the ELAN 8x10 is given below:

31 — 0	Byte Addr.
DeviceID VendorID	0x00
Status Command	0x04
ClassCode RevID	0x08
BIST HeaderType LatencyTimer CacheLineSiz	0x0C
LocalMemBase	0x10
Reserved	0x14
.	.
.	.
.	.
Reserved	0x2C
ExpansionROMBaseAddress	0x30
Reserved	0x34
Reserved	0x38
Max_Lat Min_Gnt InterruptPin InterruptLine	0x3C
0x001B80 Merror MRd MaxBurstLen	0x40
CurrMasterRdAddr	0x44
CurrMasterWrAddr	0x48

Notes on PCI Configuration Register Bits:

1. Writing values into unused register bits has no effect. However, to ensure software compatibility with future, feature-enhanced versions of the product, unused register bits must be written with logic zero. Reading back unused bits can produce either a logic one or a logic zero; hence unused register bits should be masked off by software when read.

2. Except where noted, all configuration bits that can be written into can also be read back. This allows the processor controlling the PM3350 to determine the programming state of the block.

3. Writable PCI configuration register bits are cleared to logic zero upon reset unless otherwise noted.

4. Writing into read-only PCI configuration register bit locations does not affect PM3350 operation unless otherwise noted.

5. Certain register bits are reserved. These bits are associated with megacell functions that are unused in this application. To ensure that the PM3350 operates as intended, reserved register bits must only be written with their default values. Similarly, writing to reserved registers should be avoided.

PCI Configuration Registers

PCI configuration registers can only be accessed by the PCI host. For each register description below, the hexadecimal register number indicates the PCI offset.

Register 0x00: Vendor Identification/Device Identification

Bit	Type	Function	Default
Bit 31 to 16	R	DeviceID[15:0]	0x3351
Bit 15 to 0	R	VendorID[15:0]	0x11F8

VendorID[15:0]:

The VendorID[15:0] bits identifies the manufacturer of the device, and contains the vendor ID assigned to PMC-Sierra by the PCI SIG.

DeviceID[15:0]:

The DeviceID[15:0] bits define the particular device, and is set to 0x3351 hexadecimal.

The DeviceID of both the PM3350 and the PM3351 are identical: 0x3351. To distinguish between the two devices, a PCI memory read can be done to a

specified local memory address. Contact PMC-Sierra applications support for additional information, if required.

Register 0x04: Command/Status

Bit	Type	Function	Default
Bit 31	R/W	DPE	0
Bit 30	R/W	SSE	0
Bit 29	R/W	RMA	0
Bit 28	R/W	RTA	0
Bit 27	R/W	STA	0
Bit 26	R	DVtim[1]	0
Bit 25	R	DVtim[0]	1
Bit 24	R/W	DPR	0
Bit 23	R	FBCap	1
Bit 22 t o 16	R	Reserved	0x00
Bit 15 to 10	R	Reserved	0x00
Bit 9	R/W	FBen	0
Bit 8	R/W	SEen	0
Bit 7	R	Wcyc	0
Bit 6	R/W	PEen	0
Bit 5	R	VPs	0
Bit 4	R	MWIen	0
Bit 3	R	SCen	0
Bit 2	R/W	Men	0
Bit 1	R/W	MSen	0
Bit 0	R	ISen	0

The lower 16 bits of this register make up the Command register which provides basic control over the PM3350's ability to respond to PCI accesses. When a 0 is written to all bits in the command register, the PM3350 is logically disconnected from the PCI bus for all accesses except configuration accesses.

The upper 16-bits are used to record status information for PCI bus related events. Reads to the status portion of this register behave normally. Writes are slightly different in that bits can be reset, but not set. A bit is reset whenever the register is written, and the data in the corresponding bit location is a 1.

ISen:

The I/O Space Enable (ISen) bit is hardwired to zero as the PM3350 does not implement I/O space.

MSen:

When the Memory Space Enable (MSen) bit is set to one, the PM3350 will respond to PCI bus memory accesses. Clearing MSen disables memory accesses.

Men:

When the Master Enable (Men) bit is set to one, the PM3350 can act as an master. Clearing Men disables the PM3350 from generating PCI accesses.

SCen:

The PM3350 does not decode PCI special cycles. The SCen bit is hardwired to 0.

MWIen:

The PM3350 does not generate memory-write-and-invalidate commands. The MWIen bit is hardwired to 0.

VPs:

The PM3350 is not a VGA device. The VPs bit is hardwired to 0.

PEen:

When the PEen bit is set to one, the PM3350 can report parity errors. Clearing the PEen bit causes the PM3350 to ignore parity errors.

Wcyc:

The PM3350 does not perform address and data stepping. The Wcyc bit is hardwired to 0.

SEen:

When the SEen bit is set high, address parity errors result in the assertion of the SERR* output. Clearing the SEen bit disables the SERR* output. SEen and PEen must be set to report an address parity error.

FBen:

If the fast back-to-back enable (FBen) bit is set, the PM3350 bus master will attempt to do fast back-to-back cycles across the targets.

FBCap:

The Fast Back-to-Back Capable (FBCap) bit is hardwired to one to indicate the PM3350 supports fast back-to-back transactions with other targets.

DPR:

The Data Parity Reported (DPR) bit is set high if the PM3350 has monitored or reported a data parity error to one of its transactions as a bus master and the PEen bit is set. The DPR bit is cleared by the PCI Host.

DVtim[1:0]:

The DEVSEL Timing (DVtim) bits specify the allowable timings for the assertion of DEVSEL* by the PM3350 as a target. These bits are hardwired to zeros as the PM3350 asserts DEVSEL using fast (1-cycle) response timing.

STA:

The Signaled Target Abort (STA) bit is hardwired to zero because the PM3350 will never generate a Target Abort during a slave access cycle.

RTA:

The Received Target Abort (RTA) bit is set high by the PM3350 when as an initiator, its transaction is terminated by a target abort. The RTA bit is cleared by the PCI Host.

RMA:

The Received Master Abort (RMA) bit is set high by the PM3350 when as an initiator, its transaction is terminated by a master abort. The RMA bit is cleared by the PCI Host.

SSE:

The Signaled System Error (SSE) bit is set high when ever the PM3350 asserts the SERR* output. The SSE bit is cleared by the PCI Host.

DPE:

The Data Parity Error (DPE) bit is set high when ever the PM3350 detects a parity error, even if parity error handling is disabled by clearing PEen in the Command register. The DPE bit is cleared by the PCI Host.

Register 0x08: Revision Identifier/Class Code

Bit	Type	Function	Default
Bit 31 to 24	R	CCODE[23:16]	0x02
Bit 23 to 16	R	CCODE[15:8]	0x00
Bit 15 to 8	R	CCODE[7:0]	0x00
Bit 7 to 0	R	REVID[7:0]	0x01

REVID[7:0]:

The Revision Identifier (REVID[7:0]) bits specify a device specific revision identifier and are chosen by PMC-Sierra.

CCODE[23:0]:

The class code (CCODE[23:0]) bits are divided into three groupings: CCODE[23:16] define the base class of the device, CCODE[15:8] define the

sub-class of the device and CCODE[7:0] specify a register specific programming interface.

Note:

Base Class Code:	0x02	Network Controller
Sub-Class Code:	0x00	Ethernet Network Controller
Register Class Code:	0x00	None defined.

Register 0x0C: Cache Line Size/Latency Timer/Header Type

Bit	Type	Function	Default
Bit 31 to 24	R	BIST[7:0]	0x00
Bit 23 to 16	R	HDTYPE[7:0]	0x00
Bit 15 to 8	R/W	LT[7:0]	0x00 or 0x20
Bit 7 to 0	R/W	CLSIZE[7:0]	0x00

CLSIZE[7:0]:

The Cache Line Size (CLSIZE[7:0]) bits specify the size of the system cacheline in units of dwords. The PM3350 is not capable of generating the Memory Write and Invalidate command so this register is hardwired to zeros.

LT[7:0]:

The Latency Timer (LT[7:0]) bits specify, in units of the PCI clock, the value of the Latency Timer for the PM3350. The value of the LT is application specific and should be set appropriately by the PCI BIOS at system initialization.

The default value depends on the PCIRUN bit value set by the state on the MDATA[31] pin when RST* is de-asserted. If the PCIRUN bit defaults to a logic 0, the reset the value of LT[7:0] is zero. If the PCIRUN bit defaults to a logic 1, the reset the value of LT[7:0] is decimal 32.

HDTYPE[6:0]:

The Header Type (HDTYPE[7:0]) bits specify the layout of the first 64 bytes of the configuration space. Only the 0x00 encoding is supported.

BIST[7:0]:

Built In Self Test (BIST) is not implemented so the register is hardwired to zero.

Register 0x10 : Local Memory Base Address Register

Bit	Type	Function	Default
Bit 31 to 24	R/W	MemBase[27:20]	MDATA[25:22] or 0x00
Bit 23 to 4	R	FixBase[19:0]	0x00000
Bit 3	R	PF	0
Bit 2	R	TYPE[1]	0
Bit 1	R	TYPE[0]	0
Bit 0	R	MSI	0

The PM3350 supports memory mapping only. At boot-up the internal registers space is mapped to memory space. The device driver can disable memory space through the PCI Configuration Command register.

MSI:

The Memory Space Indicator (MSI) bit is hardwired to zero to indicate that the PM3350 resources map into memory space.

TYPE[1:0]:

The TYPE field indicates where the internal registers can be mapped. The TYPE field is set to 00B to indicate that the internal registers can be mapped anywhere in the 32 bit address space.

PF:

The Prefetchable (PF) bit is set if there are no side effects on reads and data is returned on all the lanes regardless of the byte enables. Otherwise the bit is cleared. The PF bit is hardwired to one to indicate that it is safe to prefetch data from the PM3350 register/memory resources.

FixBase[19:0]:

The Fixed Base Address (FixBase[19:0]) bits define the size of the memory space required for the PM3350 resources.

This 20-bit subfield is hardcoded to zero indicating the PM3350 register/memory resources can only be mapped into the PCI address space on 16-megabyte boundaries.

MemBase[27:20]:

The Memory Base Address (MemBase[27:20]) bits define location of the memory space required for the PM3350 resources. The MemBase[27:20] bits correspond to the most significant 8 bits of the PCI address space.

After determining the memory requirements of the PM3350 resources, the PCI Host can map them to its desired location by modifying the MemBase[27:20] bits.

The default value depends on the PCIRUN bit value set by the state on the MDATA[31] pin when RST* is de-asserted. If the PCIRUN bit defaults to a logic 0, the reset the value of MemBase[27:20] is zero. If the PCIRUN bit defaults to a logic 1, the reset the value of MemBase[27:20] becomes the zero extended value of MDATA[25:22] (CHIPID[3:0).

Register 0x30: Expansion ROM Base Address

Bit	Type	Function	Default
Bit 31 to 0	R	XROMBase[31:0]	0x00000000

XROMBase[31:0]:

The PM3350 does not contain an expansion ROM; therefore, this register is hardwired to zeros.

Register 0x3C: Interrupt Line / Interrupt Pin / MIN_GNT / MAX_LAT

Bit	Type	Function	Default
Bit 31 to 24	R	MAX_LAT[7:0]	0x00
Bit 23 to 16	R	MIN_GNT[7:0]	0x03
Bit 15 to 8	R	INTPIN[7:0]	0x01
Bit 7 to 0	R/W	INTLNE[7:0]	0x00

INTLNE[7:0]:

The Interrupt Line (INTLNE[7:0]) field is used to indicate interrupt line routing information. The values in this register are system specific and set by the PCI Host.

INTPIN[7:0]:

The Interrupt Pin (INTPIN[7:0]) field is used to specify the interrupt pin the PM3350 uses. Because the PM3350 will use INTA* on the PCI bus, the value in this register is set to one.

MIN_GNT[7:0]:

The Minimum Grant (MIN_GNT[7:0]) field specifies how long of a burst period the bus master needs. The PM3350 has a minimum grant of 750 ns (25 cycles). This register always returns 0x03.

MAX_LAT[7:0]:

The Maximum Latency (MAX_LAT[7:0]) field specifies how often a bus

master needs access to the PCI bus. The PM3350 has no specific requirements in this regard; this register always returns zero.

Register 0x40: Merror / Mrd / Max. Burst Length

Bit	Type	Function	Default
Bit 31 to 25	R	Reserved	0x00
Bit 24	R/W	TestDiscardTimer	0
Bit 23 to 16	R/W	Tdiscon	0x6E
Bit 15 to 10	R	Reserved	0x00
Bit 9	R	Mrd	0
Bit 8	R	Merror	0
Bit 7 to 0	R/W	MaxBurstLen[7:0]	0x10

MaxBurstLen[7:0]:

This register should be initialized with the maximum single-transaction burst length in DWORDs to be used by the PCI bus master. It defaults to 16 DWORDS.

Merror:

A logic one in this bit position indicates that the PCI bus master has encountered a fatal error.

Mrd:

A logic one in this bit position indicates that the PCI bus master is currently performing a read transaction.

Tdiscon:

The slave timer disconnect register is used to limit TRDY and STOP assertion based on the value in this register. Tdiscon[3:0] sets the disconnect delay for the first data access, while Tdiscon[7:4] defines the disconnect delay for subsequent data phases of the same burst. Note that the actual number of cycles, as measured from the start of the data phase to the cycle in which the disconnect is returned by the PCI bus slave interface, is two greater than the numeric values programmed into the Tdiscon[3:0] and Tdiscon[7:4] fields. This register is for factory configuration purposes only. The value defaults to 0x6E (disconnect after 16 cycles on the first access and 8 cycles thereafter), and must remain at that value for correct operation.

TestDiscardTimer:

This register bit allows testing of the Discard Timer feature in the PCI bus slave logic. It defaults to zero. If set to logic 1, it causes the Discard Timer in the slave to be initialized to 0x2FF0 rather than 0x0000. This bit is for factory test purposes only. It must remain at logic 0 for correct operation.

Register 0x44: Current Master Read Address

This register is provided for diagnostic purposes.

Bit	Type	Function	Default
Bit 31 to 0	R	CurrMasterRdAddr	0x00000000

CurrMasterRdAddr[31:0]:

Current read target address used by PCI bus master.

Register 0x48: Current Master Write Address

This register is provided for diagnostic purposes.

Bit	Type	Function	Default
Bit 31 to 0	R	CurrMasterRdAddr	0x00000000

CurrMasterRdAddr[31:0]:

Current write target address used by PCI bus master.

PCI Memory Space

The mapping of the device resources to the PCI address space is shown below:

Mnemonic	PCI Offset	Description
	0x000000-0xFEFFFF	Local Memory Space (four banks)
HSTAT	0xFF0000	Host Interface Status register
HCTRL	0xFF0004	Host Interface Control register
MREG_ADDR	0xFF0040	Memory Interface Register Address
MREG_DATA	0xFF0044	Memory Interface Register Data
INSTCSR	0xFF0048	RAM/ROM Instruction Control register
INSTDATA	0xFF004C	RAM/ROM Instruction Data register
DIBCTRL	0xFF0080	Breakpoint/debug control/status register
DPC	0xFF0084	RISC program counter
DEPC	0xFF0088	RISC exception program counter
DVEC	0xFF008C	Vector force address
DIBADDR1	0xFF0090	Instruction breakpoint #1 address
DIBMASK1	0xFF0094	Instruction breakpoint #1 mask
DIBADDR2	0xFF0098	Instruction breakpoint #2 address
DIBMASK2	0xFF009C	Instruction breakpoint #2 mask
	0xFF009D-0xFFFEFF	Unused, reserved
RCOUNT0	0xFFFF00	Channel 0 Request Counter Register
RCOUNT1	0xFFFF04	Channel 1 Request Counter Register
RCOUNT2	0xFFFF08	Channel 2 Request Counter Register
RCOUNT3	0xFFFF0C	Channel 3 Request Counter Register
RCOUNT4	0xFFFF10	Channel 4 Request Counter Register
RCOUNT5	0xFFFF14	Channel 5 Request Counter Register
RCOUNT6	0xFFFF18	Channel 6 Request Counter Register
RCOUNT7	0xFFFF1C	Channel 7 Request Counter Register
ACOUNT0	0xFFFF20	Channel 0 Acknowledge Counter Register
ACOUNT1	0xFFFF24	Channel 1 Acknowledge Counter Register
ACOUNT2	0xFFFF28	Channel 2 Acknowledge Counter Register
ACOUNT3	0xFFFF2C	Channel 3 Acknowledge Counter Register
ACOUNT4	0xFFFF30	Channel 4 Acknowledge Counter Register
ACOUNT5	0xFFFF34	Channel 5 Acknowledge Counter Register
ACOUNT6	0xFFFF38	Channel 6 Acknowledge Counter Register

ACOUNT7	0xFFFF3C	Channel 7 Acknowledge Counter Register

Local Memory Access

The external local memory is completely visible to the PCI interface. Direct memory access is provided to allow reads and writes to frame data, switching data structures, and configuration information. Note that the uppermost 64 Kbytes of the local memory space is not visible, as it is used for access to internal device registers.

Device Control/Status Registers

Register 0x00FF0000: Host Interface Status

Bit	Type	Function	Default
Bit 31 to 12		Unused	X
Bit 11 to 7	R	HSTATE[4:0]	0
Bit 6	R	Unused	0
Bit 5	R	RINT	0
Bit 4	R	EXTI	0
Bit 3	R	RHALT	0
Bit 2	R	BKPT	0
Bit 1	R	MERR	0
Bit 0	R	MPERR	0

HSTATE:
Internal host interface state: for factory test purposes only

RINT:
Flags assertion of interrupt to host from Switch Processor

EXTI:
Signals assertion of external interrupt input (MINTR_ pin)

RHALT:
Switch Processor halt status

BKPT:
Switch Processor breakpoint status

MERR:
PCI bus master catastrophic error status

MPERR:
PCI bus master parity error detect

Register 0x00FF0004: Host Interface Control

Bit	Type	Function	Default
Bit 31 to 16		Unused	X
Bit 15	R/W	GRES	0
Bit 14	R/W	MWBE	0
Bit 13	R/W	MRBE	0
Bit 12	R/W	SWBE	0
Bit 11	R/W	SRBE	0
Bit 10	R/W	EPBE	0
Bit 9	R/W	RAMDIS	0
Bit 8	R/W	ROMDIS	0
Bit 7	R/W	RHALT	MDATA[30][1]
Bit 6	R/W	Reserved	0
Bit 5	R/W	RINTMSK	0
Bit 4	R/W	EXTMSK	0
Bit 3	R/W	RHALTMSK	0
Bit 2	R/W	BKPTMSK	0
Bit 1	R/W	MERRMSK	0
Bit 0	R/W	MPERRMSK	0

GRES:
General device reset

MWBE:
PCI Master write byteswap enable

MRBE:
PCI Master read byteswap enable

SWBE:
PCI Target (Slave) write byteswap enable

SRBE:
PCI Target (Slave) read byteswap enable

EPBE:
Ethernet payload byteswap enable

[1]The default value is the negation of the logical value on the specified pin when RST* is de-asserted.

RAMDIS:
Switch Processor internal instruction RAM disable

ROMDIS:
Switch Processor internal instruction ROM disable

RHALT:
Switch Processor halt enable control

RINTMSK:
Interrupt mask: enables interrupts from Switch Processor to host

EXTMSK:
Interrupt mask: enables interrupts to host via MINTR_ input pin

RHALTMSK:
Interrupt mask: enables interrupt to host when Switch Processor is halted

BKPTMSK:
Interrupt mask: enables interrupt to host when Switch Processor reaches pre-set breakpoint

MERRMSK:
Interrupt mask: enables interrupt to host on PCI bus master catastrophic errors

MPERRMSK:
Interrupt mask: enables interrupt to host on PCI bus master parity errors

The default value is the negation of the logical value on the specified pin when RST_ is de-asserted.

Device Configuration Registers

Register 0x00FF0040: Memory Register Bank Address (MREG_ADDR)

Register 0x00FF0044: Memory Register Bank Data (MREG_DATA)

The on-chip memory interface contains several registers that are accessible over the PCI bus via the two registers MREG_ADDR and MREG_DATA. The user writes the applicable register select value to the MREG_ADDR register and then performs a read (or write) to the MREG_DATA register to perform a PCI bus read (or write) operation to the target register. The following table gives the indices for the registers in the on-chip memory interface.

MREG_ADDR[3:0]	Register accessed	Description
0x0	MCONFIG	Memory configuration register
0x1	DCONFIG	Device configuration register
0x2	SMR	SDRAM mode register (reserved)
0x3	Reserved	
0x4	Reserved	
0x5	MEMCTRL	Memory control/status
0x6	MSTART_l	Memory statistics collection start
0x7	MSTART_h	
0x8	MSTOP_l	Memory statistics collection stop
0x9	MSTOP_h	
0xA	MCCNT_l	Memory cycle count
0xB	MCCNT_h	
0xC	MWCNT_l	Memory write count
0xD	MWCNT_h	
0xE	MRCNT_l	Memory read count
0xF	MRCNT_h	

MREG_ADDR[31:4] bits are unimplemented so must be written with zero and ignored on reads.

Locations 0x5 through 0xF are for diagnostic purposes only.

By way of example, the following the sequence of operations are required to first read the DCONFIG register and then write the MCONFIG register in the memory interface:

- PCI write MREG_ADDR to 0x1 — expansion address for DCONFIG
- PCI read MREG_DATA — read DCONFIG register
- PCI write MREG_ADDR to 0x0 — expansion address for MCONFIG
- PCI write MREG_DATA to val[15:0] — MCONFIG register written as val[15:0]

Note that the SMR, MEMCTRL, MSTART, MSTOP, MWCNT and MRCNT registers are all reserved for factory testing, and must not be accessed during normal operation.

MCONFIG (Memory Configuration register)

Bit	Type	Function	Default
Bit 31 to 16		Unused	X
Bit 15 to 14	R/W	MXSEL[1:0]	MDATA[15:14][1]
Bit 13	R/W	MSLO	MDATA[13][1]
Bit 12	R/W	MDCAS	MDATA[12][1]
Bit 11 to 9	R/W	MTYPE3[2:0]	MDATA[11:9][1]
Bit 8 to 6	R/W	MTYPE2[2:0]	MDATA[8:6][1]
Bit 5 to 3	R/W	MTYPE1[2:0]	MDATA[5:3][1]
Bit 2 to 0	R/W	MTYPE0[2:0]	MDATA[2:0][1]

MXSEL[1:0]:

These bits configure the DRAM row/column multiplexing to accommodate different DRAM organizations:

MXSEL	Column Address Bits
00	8
01	9
10	10
11	11

MSLO:

Determines the expected access time of the local memory: if a logic 1, 80ns DRAM is expected; otherwise, 60ns DRAM is expected. The PM3350 is intended to be used with 60ns EDO DRAM; hence, MSLO must be a logic 0 for correct operation.

MDCAS:

Used to indicate the organization of DRAM used in the system, if any: if set to a logic 1, the ELAN 8x10 will generate control signals for 2-CAS DRAMs; otherwise, the ELAN 8x10 will assume 1-CAS DRAM devices

MTYPE3[2:0]:

Indicates the type of memory device selected with MCS[3]*:

000	Reserved
001	Reserved
010	Reserved
011	Reserved
100	Reserved
101	200ns (E)EPROM

110 60ns EDO DRAM
111 Reserved

MTYPE2[2:0]:
Indicates the memory type associated with MCS[2]*. The encoding is the same as MTYPE3[2:0].

MTYPE1[2:0]:
Indicates the memory type associated with MCS[1]*. The encoding is the same as MTYPE3[2:0].

MTYPE0[2:0]:
Indicates the memory type associated with MCS[0]*. The encoding is the same as MTYPE3[2:0].

Note 1: The default value is the logical value on the specified pin when RST* is de-asserted.

DCONFIG (Device Configuration register)

Bit	Type	Function	Default
Bit 31 to 16		Unused	X
Bit 15	R/W	PCIRUN	MDATA[31][1]
Bit 14	R/W	RISCRUN	MDATA[30][1]
Bit 13	R/W	RSTTM	MDATA[29][1]
Bit 12	R/W	IMDIS	MDATA[28][1]
Bit 11	R/W	PCI3V	MDATA[27][1]
Bit 10	R/W	FIRM	MDATA[26][1]
Bit 9 to 6	R/W	CHIPID[3:0]	MDATA[25:22][1]
Bit 5 to 0	R/W	RTCDIV[5:0]	MDATA[21:16][1]

PCIRUN:
This bit selects the operating mode of the PCI interface.
If logic 1:
• the on-chip PCI interface will latch its slave base address from the CHIPID bits of the DCONFIG register.
• the PCI COMMAND register bits for "Bus Master" and "Memory Space" are set to logic 1, thereby allowing the device to respond to PCI memory space accesses and to be a bus master.
If logic 0:
• the on-chip PCI interface will has a slave base address of 0x0.

• the PCI COMMAND register bits for "Bus Master" and "Memory Space" are set to logic 0; the device is disabled from responding to PCI memory space accesses and will not be a bus master.

RISCRUN:

If set to zero, the Switch Processor enters a halt state immediately after system reset is de-asserted, placing the ELAN 8x10 into stand-by mode.

RSTTM:

For factory test purposes only: set to logic 0 for correct operation

IMDIS:

Internal Switch Processor ROM disable: if set to logic 1, the ELAN 8x10 Switch Processor begins execution from external memory after reset, otherwise it executes from internal ROM.

PCI3V:

If logic 1, configures the PCI interface for the 3.3V signaling environment; otherwise, configures the PCI interface for the 5V signaling environment. Must be set to logic 0 since all AC parametric test data taken solely with this bit set to “0”.

FIRM:

Reserved for use by firmware; should be set to zero.

CHIPID[3:0]:

ELAN 8x10 chip identifier: these bits are zero-extended to 8 bits and loaded into the most significant byte of the PCI memory base address register upon reset if the RUN bit is set.

RTCDIV[5:0]:

Prescaler divide ratio for internal real-time clock: must be set to the numeric value of the SYSCLK frequency in megahertz (e.g., 50 decimal when using a 50 MHz system clock).

Note 1: The default value loaded into the field is the logical value driven on to the specified pin when RST* is de-asserted.

Register 0x00FF0048: RAM/ROM instruction control

This register is reserved for factory test purposes, and should not be modified during normal operation.

Register 0x00FF004C: RAM/ROM instruction Data

This register is reserved for factory test purposes, and should not be modified during normal operation.

Device Debug Registers

Registers 0x00FF0080 - 0x00FF009C: Debug Control

These registers are used to perform debug of Switch Processor operating code via the PCI bus interface. They are reserved for factory test purposes, and should not be modified during normal operation.

Debug control/status register (DIBCTRL):

bit 31	= vector force recognized
bits 30:9	= unused, reserved
bit 8	= vector force enable
bit 7	= instruction cache flush
bit 6	= data cache flush
bit 5	= RISC exception disable
bits 4:3	= unused, reserved
bit 2	= RISC breakpoint halt enable
bit 1	= breakpoint #2 enable
bit 0	= breakpoint #1 enable

Request and Acknowledge Counter Registers

Registers 0x00FFFF00 - 0x00FFFF1C: Request Counters

These counters are used for requests from a source device to a destination device across the PCI bus. A write (of any arbitrary data) to one of these 32-bit locations increments the corresponding request counter by one.

Registers 0x00FFFF20 - 0x00FFFF3C: Acknowledge Counters

These counters are used for acknowledges from a destination device to a source device across the PCI bus. A write (of any arbitrary data) to one of these 32-bit locations increments the corresponding acknowledge counter by one.

PROGRAMMER'S MODEL

This section contains information pertinent to RISC firmware designers.

Local Memory Map

The ELAN 8x10 creates and uses data structures stored in an external local memory that is organized as a contiguous 16MB address space. Memory addresses are 24 bits in width, and reference 8-bit bytes in little-endian form, i.e. the least-significant byte is located at the lowest byte offset within a 32-bit word or 16-bit halfword.

Register Map

General-Purpose Registers

There are sixteen 16-bit general registers, *gr0* through *gr15*. They are used for all arithmetic operations, memory load and store operations, and branch condition tests, and are accessed via CPU instructions.

All general registers are identical; they are all readable and writable by software.

An alternate bank of registers is switched in to replace the current set of general registers when the REGSW instruction is executed or an exception is taken. The alternate registers are accessed with the same indices as the normal register set.

Special-Purpose Registers

The special-purpose registers implement memory pointers, hold subroutine return addresses, contain exception control and status information, support the microcode instructions, and implement the real-time clock and alarms.

The contents of any special-purpose register may be accessed or modified by the MOVE and LDI instructions.

The memory pointers (MP0-MP6) are replaced by an alternate bank of registers when the REGSW instruction is executed or an exception is taken. The alternate registers are accessed in the same way and by the same instructions as the main register set.

Mnemonic	Address	Register
MP0L	0x00	Memory Pointer #0 (Low)
MP0H	0x01	Memory Pointer #0 (High)
MP1L	0x02	Memory Pointer #1 (Low)
MP1H	0x03	Memory Pointer #1 (High)
MP2L	0x04	Memory Pointer #2 (Low)
MP2H	0x05	Memory Pointer #2 (High)
MP3L	0x06	Memory Pointer #3 (Low)
MP3H	0x07	Memory Pointer #3 (High)
MP4L	0x08	Memory Pointer #4 (Low)
MP4H	0x09	Memory Pointer #4 (High)
MP5L	0x0A	Memory Pointer #5 (Low)
MP5H	0x0B	Memory Pointer #5 (High)
MP6L	0x0C	Memory Pointer #6 (Low)
MP6H	0x0D	Memory Pointer #6 (High)
LNRL	0x0E	Link Register (Low)
LNRH	0x0F	Link Register (High)
ESCL	0x10	Exception Status/Control Register (Low)
ESCH	0x11	Exception Status/Control Register (High)
EPCL	0x12	Exception Program Counter (Low)
EPCH	0x13	Exception Program Counter (High)
EMASK	0x14	Exception Mask Register
OFFSET	0x15	Data Segment Offset Register
OUTCBL	0x16	Microcode Output Control Bits Register (Low)
OUTCBH	0x17	Microcode Output Control Bits Register (High)
CLOCKL	0x18	Real-Time Clock Counter Register (Low)
CLOCKH	0x19	Real-Time Clock Counter Register (High)
ALARM1L	0x1A	Clock Alarm #1 (Low)
ALARM1H	0x1B	Clock Alarm #1 (High)
ALARM2L	0x1C	Clock Alarm #2 (Low)
ALARM2H	0x1D	Clock Alarm #2 (High)
PCR	0x1E	Processor Control Register
INCB	0x1F	Microcode Input Control Bits Register

Control Registers

The 96 16-bit control registers, *cr0* through *cr95*, provide control and status of the functional blocks comprising the PM3350.

The control registers can be read by the RISC only by using MOVE instructions to transfer their contents to the general-purpose registers. The contents of any control register may be accessed or modified by the MOVE and LDI instructions. Some of the control registers may exhibit side effects when written to; these are noted in the register descriptions.

Mnemonic	Address	Register
DMCTRL	0x20	DMA control word (cr0)
DMSTAT	0x21	DMA status register (cr1)
INSTCSR	0x22	RAM/ROM instruction control register (cr2)
INSTDATA	0x23	RAM/ROM instruction data register (cr3)
DMFLISTL	0x24	DMA freelist pointer low (cr4)
DMFLISTH	0x25	DMA freelist pointer high (cr5)
MREG_ADDR	0x26	Memory Interface Register Address register (cr6)
MREG_DATA	0x27	Memory Interface Register Data register (cr7)
	0x28	
REQID	0x29	Request info (cr9)
ACKID	0x2A	Acknowledge info (cr10)
TASKCTR	0x2B	Task counter (cr11)
DMNFREEL	0x2C	DMA next free packet buffer low (cr12)
DMNFREEH	0x2D	DMA next free packet buffer high (cr13)
CTRSEL	0x2E	Request/ack counter select reg (cr14)
CTRDATA	0x2F	Request/ack counter data reg (cr15)
RPCICMD	0x30	RPCIM command register (cr16)
RPCIDATA	0x31	RPCIM Data register (cr17)
RPCIADDRL	0x32	RPCIM Address register high (cr18)
RPCIADDRH	0x33	RPCIM Address register low (cr19)
RPCISTAT	0x34	RPCIM status register (cr20)
WTIMER	0x35	RST module: Watchdog Timer (cr21)
FRST	0x36	RST module: Firmware RST register (cr22)
DBGCTRL	0x37	DBG module Control register (cr23)
RAMREGL	0x38	HASH ram register 15:0 (cr24)
RAMREGH	0x39	HASH ram register 31:16 (cr25)
HCADDR	0x3A	HASH control address register (cr26)

HCDATA	0x3B	HASH control data address (cr27)
	0x3C-0x3F	Reserved
IFS1_TIME	0x40	Interframe gap time (1st part) (cr32)
IFS2_TIME	0x41	Interframe gap time (2nd part) (cr33)
PRE_TIME	0x42	Preamble time (cr34)
JAM_TIME	0x43	Jam time (cr35)
MAX_TIME	0x44	Max frame size (cr36)
MIN_TIME	0x45	Min frame size (cr37)
BLIND_TIME	0x46	Blind time (cr38)
LBK_BYTE	0x47	Loopback byte (cr39)
PBSIZE	0x48	Packet buffer size (in bytes) (cr40)
INC_DATAL	0x49	Increment channel data word low (cr41)
INC_DATAH	0x4A	Increment channel data word high (cr42)
ETYPE	0x4B	Mac control frame type field (cr43)
LATE_TIME	0x4C	Late time (cr44)
	0x4D-0x4F	Reserved
SCC_STAT	0x50	Serial / Parallel status (cr48)
SCC_CTRL	0x51	Serial / Parallel control (cr49)
SCC_DATA	0x52	Serial / Parallel data (cr50)
	0x53-0x5E	
DMWIN	0x5F	DMA channel register window (cr63)
LWRXSIZE	0x60	Link windowed rx frame size (cr64)
	0x61-0x67	Reserved
LWTXSIZE	0x68	Link windowed tx frame size (cr72)
	0x69-0x6F	Reserved
LWCTRL	0x70	Link control word (cr80)
LWSTAT	0x71	Link Status word (cr81)
LWBACK	0x72	Backoff timer (cr82)
LWNBUFS	0x73	Number of Packet Buffers (cr83)
LWFIRSTL	0x74	First PB address low (cr84)
LWFIRSTH	0x75	First PB address high (cr85)
LWLASTL	0x76	Last PB address low (cr86)
LWLASTH	0x77	Last PB address high (cr87)
LWPADRL	0x78	Remote PB address high (cr89)
LWPADRH	0x79	Remote PB address high (cr88)
DMSTAT2	0x7A	DMA stats part 2 (cr90)
LWSIZE	0x7B	Local Link Last Frame Size (cr91)
HSRCRESULTL	0x7C	HASH source result register low word (cr92)

HSRCRESULTH	0x7D	HASH source result register high word (cr93)
HDSTRESULTL	0x7E	HASH destination result register low word (cr94)
HDSTRESULTH	0x7F	HASH destination result register high word (cr95)

Register Descriptions

CPU Special Registers

For further information about the following CPU registers please refer to the ECPU programmers manual.

Register 0x00-0x0D: Memory Pointers #0 - #6

Register 0x0E, 0x0F: Link Register

Register 0x10, 0x11: Exception Status Register

Register 0x12, 0x13: Exception Program Counter

Register 0x14: Exception Mask Register

Register 0x15: Data Segment Offset Register

Register 0x16, 0x17: Microcode Output Control Bits Register

Register 0x18, 0x19: Real-Time Clock Counter Register

Register 0x1A, 0x1B: Clock Alarm #1

Register 0x1C, 0x1D: Clock Alarm #2

Register 0x1E: Processor Control Register

Register 0x1F: Microcode Input Control Bits Register

Device Control Registers

Register 0x20: DMA control word (cr0)

Bit	Type	Function	Default
Bit 15	R/W	DMCR_enhpcirdmode	0
Bit 14	R/W	DMCR_pci_rd	0
Bit 13	R/W	DMCR_lbk	0
Bit 12	R/W	DMCR_drst	0
Bit 11		Unused	X
Bit 10	R/W	DMCR_pincen	0
Bit 9	R/W	DMCR_pwren	0
Bit 8	R/W	DMCR_prden	0
Bit 7	R/W	DMCR_ch7en	0
Bit 6	R/W	DMCR_ch6en	0
Bit 5	R/W	DMCR_ch5en	0
Bit 4	R/W	DMCR_ch4en	0
Bit 3	R/W	DMCR_ch3en	0
Bit 2	R/W	DMCR_ch2en	0
Bit 1	R/W	DMCR_ch1en	0
Bit 0	R/W	DMCR_ch0en	0

DMCR_ch0en:
Active high enable for local MAC channel 0

DMCR_ch1en:
Active high enable for local MAC channel 1

DMCR_ch2en:
Active high enable for local MAC channel 2

DMCR_ch3en:
Active high enable for local MAC channel 3

DMCR_ch4en:
Active high enable for local MAC channel 4

DMCR_ch5en:
Active high enable for local MAC channel 5

DMCR_ch6en:
Active high enable for local MAC channel 6

DMCR_ch7en:
Active high enable for local MAC channel 7

DMCR_prden:
Active high enable for PCI read channel. Automatically set when a set_transmit flip is performed on channel 8.

DMCR_pwren:
Active high enable for PCI write channel. Automatically set when a set_transmit flip is performed on channel 9.

DMCR_pincen:
Active high enable for PCI increment channel. Automatically set when a set_transmit flip is performed on channel 10.

DMCR_drst:
Active high reset for DMA logic. If this bit is set to a logic 1, the DMA functional block is held in its reset state.

DMCR_lbk:
If this bit is a logic 1 the LBK output pin is asserted high to force the PHY device into loopback mode.

DMCR_pci_rd:
If this bit is a logic 1 the out of packet buffers notification for the PCI read channel will be suspended. If this bit is logic 0 during PCI reads if the free list goes to zero(out of packet buffers) the RISC will be notified during the PCI read channel interrupt.

DMCR_enhpcirdmode:
If this bit is a logic 0 the PCI read channel will read the entire packet buffers worth of data over the PCI. If this bit is logic 1 the PCI read channel will run in enhanced mode by reading across the PCI only the required data in the packet buffer. This is done in the following manner. If the number of packet buffers in the frame is greater than one (1) then it is expected that the first packet buffer header of the frame to be formatted in the following manner:

	Bits [31:24]	Bits [23:0]
Word 0	Packet Buffer Bytes	Packet Buffer Next Pointer
Word 1	Last Packet Buffer Bytes	Don't Care

Word 0 contains the normal information for the PC read channel. Word 1 contains the number of bytes in the last packet buffer in this frame.

Register 0x21: DMA Stats (cr1)

Bit	Type	Function	Default
Bit 15	R	DMSR_gntpend	X
Bit 14	R	DMSR_PMWF	X
Bit 13	R	DMSR_PMCF	X
Bit 12	R	DMSR_PMRE	X
Bit 11 to 8	R	DMSR_chsel[3:0]	X
Bit 7	R	DMSR_flerr	X
Bit 6	R	DMSR_merr	X
Bit 5 to 0	R	DMSR_state[5:0]	X

DMSR_state[5:0]:
current DMA state machine state

DMSR_merr:
PCI master error detected

DMSR_flerr:
empty free list error detected

DMSR_chsel[3:0]:
current internal DMA channel select

DMSR_PMRE:
PCI master read data FIFO empty flag

DMSR_PMCF:
PCI master read control FIFO full flag

DMSR_PMWF:
PCI master write FIFO full flag

DMSR_gntpend:
This bit becomes a logic 1 when a DMA memory grant is pending.

Register 0x22: RAM/ROM Instruction Control register (cr2)

Bit	Type	Function	Default
Bit 15	R/W	RAMPR	0
Bit 14	R/W	ROMPR	0
Bit 14		Unused	0
Bit 12	R/W	RW	0
Bit 11	R/W	HL	0
Bit 10 to 0	R/W	ADDR[10:0]	0

RAMPR:

Program (1) or execute (0) contents of RAM

ROMPR:

Program (1) or execute (0) contents of ROM

RW:

Select between write (1) and read (0) of the INSTDATA register

HL:

Select high (1) bank or low (0) bank of INSTDATA for control data.

ADDR[10:0]:

RAM/ROM instruction address. Only 7 bits for ROM.

Register 0x23: INSTDATA, RAM/ROM Instruction Data register (cr3)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IDATA[15:0]	0

IDATA[15:0]:

The register operates in conjunction with the RAM/ROM Instruction Control register (cr2) to read and write the internal instruction RAM and ROM.

Register 0x24: DMA freelist pointer low (cr4)

Bit	Type	Function	Default
Bit 15 to 0	R/W	FLIST[15:0]	0

Register 0x25: DMA freelist pointer high (cr5)

Bit	Type	Function	Default
Bit 15 to 8		Unused	X
Bit 7 to 0	R/W	FLIST[23:16]	0

FLIST[23:0]:

The freelist pointer locates the head of the linked list of free packet buffers within the local memory. These registers are modified by either writes by the RISC or by the DMA upon transfer of the packet buffer.

Register 0x26: Memory Register Bank Address (MREG_ADDR, cr6)

Register 0x27: Memory Register Bank Data (MREG_DATA, cr7)

The on-chip memory interface contains several registers that are accessible over the PCI bus via the two registers MREG_ADDR and MREG_DATA. The user writes the applicable register select value to the MREG_ADDR register and then performs a read (or write) to the MREG_DATA register to perform a PCI bus read (or write) operation to the target register.

These registers are accessible by the PCI expansion bus. Refer to the PCI Accessible Registers section for details on the use of these registers.

Register 0x29 (cr9): Request info (REQID)

Bit	Type	Function	Default
Bit 15	R	REQID_VALID	0
Bit 14	R	REQ_INT	0
Bit 13 to 7		Unused	0
Bit 6 to 4	R	REQID_NUM[2:0]	0x7
Bit 3 to 0		Unused	0

REQID_VALID:

Valid bit - there is a Request pending. Qualifies REQID_NUM.

REQ_INT:

Request Interrupt status (for observability; not used by firmware during normal switching).

REQID_NUM[2:0]:

Requesting chip number.

This is used by the Switch Processor firmware during service of a Request Interrupt.

This number is selected by dedicated hardware (counter bank support logic) on the PM3351 that examines all eight of the request counters (these are RCOUNT0 through RCOUNT7). It takes a snapshot of which of these counters is non-zero (request snapshot register) and services them in order from RCOUNT0 to RCOUNT7. The Switch Processor signals to the logic that the current REQID_NUM has been completed by doing a flip (REQ_DEC). This clears the bit in the request snapshot register for the currently selected counter, which allows the REQID_NUM to be updated with the next request counter number (which is only valid if the REQID_VALID bit is also set). A new snapshot of the request counters is taken whenever the following two conditions are met:

- the request snapshot register is zero.
- at least one of the RCOUNT0 through RCOUNT7 registers is non-zero.

Register 0x2A (cr10): Acknowledge info (ACKID)

Bit	Type	Function	Default
Bit 15	R	ACKID_VALID	0
Bit 14	R	ACK_INT	0
BIT 13 to 7		Unused	0
BIT 6 to 4	R	ACKID_NUM[2:0]	0x7
BIT 3 to 0		Unused	0

ACKID_VALID]:

Valid bit - there is a Acknowledge pending. Qualifies ACKID_NUM.

ACK_INT:

Acknowledge Interrupt status (for observability; not used by firmware during normal switching).

ACKID_NUM:

Acknowledge chip number.

This is used by the Switch Processor firmware during service of an Acknowledge Interrupt. Implementation of ACKID_NUM is similar to REQID_NUM with the following differences:

- there is an acknowledge snapshot register
- the Switch Processor does an "ACK_DEC" flip to signal completion of the service routine for the selected ACKID_NUM.

Register 0x2B (cr11): Task counter (TASKCTR)

Bit	Type	Function	Default
Bit 15 to 0	R/W	TCOUNT[15:0]	0

TCOUNT[15:0]:

This register implements a 16-bit up-down counter. The counter is incremented and decremented under control of the Switch Processor:

- the counter is incremented if the task counter increment flip is asserted (task_inc).
- the counter is decremented if the task counter decrement flip is asserted (task_dec) –AND- task_inc is not simultaneously asserted.
- If the TCOUNT[15:0] register is non-zero, the TaskCounter interrupt is asserted to the Switch Processor.

Note: the present firmware implementation of the PM3351 does not use this counter. It may be used in future firmware releases.

Register 0x2C: DMA next free packet buffer low (cr12)

Bit	**Type**	**Function**	**Default**
Bit 15 to 0	R/W	NFREE[15:0]	0

Register 0x2D: DMA next free packet buffer high (cr13)

Bit	**Type**	**Function**	**Default**
Bit 15 to 8		Unused	X
Bit 7 to 0	R/W	NFREE[23:16]	0

NFREE[23:0]:

The next free pointer is the address in local memory of the next available

packet buffer. These registers are modified by either writes by the RISC or by the DMA upon transfer of the packet buffer.

Register 0x2E: Request/ack counter select reg (cr14)

Bit	Type	Function	Default
Bit 15	R/W	CTRSELTM	0
Bit 14	R/W	EN_WR_COUNTER	0
Bit 13	R/W	DISABLE_REQ_INT	0
Bit 12	R/W	DISABLE_ACK_INT	0
Bit 11 to 4		Unused	0
Bit 3 to 0	R/W	CTRSEL[3:0]	0

During normal switching applications the CTRLSEL register will be left in the default state and is unused. It is implemented solely for factory test.

CTRSELTM:

The testmode bit for counter bank testing. This bit must be a logic 0.

EN_WR_COUNTER:

When set to a logic 1, this bit allows req/ack counters to be written by the Switch Processor.

DISABLE_REQ_INT:

When set to a logic 1, this bit disables the generation of the REQ_INT interrupt line.

DISABLE_ACK_INT:

When set to a logic 1, this bit disables the generation of the ACK_INT interrupt line.

CTRSEL[3:0]:

Selects the counter to view .

CTRSEL[3:0]	CTRDATA[11:0]
0000	rcountq0
0001	rcountq1
0010	rcountq2
0011	rcountq3
0100	rcountq4
0101	rcountq5
0110	rcountq6
0111	rcountq7
1000	acountq0
1001	acountq1
1010	acountq2
1011	acountq3
1100	acountq4
1101	acountq5
1110	acountq6
1111	acountq7

Register 0x2F: Request/ack counter data reg (cr15)

Bit	Type	Function	Default
Bit 15 to 0	R/W	CTRDATA[15:0]	0

During normal switching applications the CTRDATA register will be left in the default state and is unused. It is implemented solely for factory test.

CTRDATA[15:0]:

The data from the selected counter bank. This allows the Switch Processor to read or write one of the eight RCOUNT or eight ACOUNT registers that are also mapped into the PCI host register space.

The Switch Processor can READ a selected Request or Acknowledge counter as follows:

- the CTRSEL register is written with the CTRSEL.en_wr_counter bit clear (0) and the CTRSEL[3:0] field set to access the desired counter.
- there is a minimum of one processor delay slot

- the value of the selected counter can be read from the CTRDATA register using the MOVE assembly instruction.

The Switch Processor can WRITE a selected Request or Acknowledge counter as follows:

- the CTRSEL register is written with the CTRSEL.en_wr_counter bit set (1) and the CTRSEL[3:0] field set to access the desired counter.
- the Switch Processor uses the MOVE or LDI assembly instructions to write the desired value to the CTRDATA register.
- on the cycle after CTRDATA is written by the Switch Processor the write data updates the selected Request/Acknowledge counter.

RPCIM Functional Block (PCI access DMA channel)

The purpose of the RPCIM logic block on the PM3351 is to allow the Switch Processor to directly initiate PCI bus master transactions:

- memory read transactions
- memory write transactions
- configuration read transactions
- configuration write transactions

In normal Ethernet switching applications firmware will only utilize the RPCIM logic during system initialization; it is the various dedicated DMA channels on the PM3351 that are used to initiate all of the PCI master transactions on the PCI expansion bus that are required by the ELAN switch protocol. During system initialization all of the PCI bus transactions that the Master chip in a system does to initialize Slave chips is done utilizing the RPCIM logic block.

Outline of RPCIM memory/configuration WRITE transactions:

For write transfers, the data to be written to the PCI bus must be broken up into a sequence of transactions that are from 1 to 63 bytes in size. The data to be written to the PCI bus is held in an internal RPCIM data FIFO which is accessible via writes to the RPCIDATA register. The data FIFO write pointer is auto-incremented on each write to the RPCIDATA register. The transfer is initiated by writing to a series of Switch Processor accessible control registers: RPCIADDR (address) and RPCICMD (command related). The write to the RPCICMD register signals the RPCIM internal state machine to start the PCI master write transaction.

The RPCIM must arbitrate with the other internal DMA channels that can initiate PCI write transactions for access to a shared hardware resource: the PCI Master Write FIFO (MWF FIFO) that is part of the internal PCI Bus Interace Unit (PCI_BIU). The arbiter for access to the MWF FIFO is internal to the RPCIM logic. The grant for the write interface, `rwr_gt', is set on the cycle before the RPCIM writes the command word in the MWF FIFO.

Upon writing the last word of the command/address/data into the MWF FIFO for a given transaction, the following occurs concurrently:

- the rpcim_done line is asserted (this maps to one of the interrupt lines on the PM3351 Switch Processor).
- rwr_gt is deasserted; this allows another DMA channel to perform PCI write transactions.

On the PCI bus the write transaction will appear as a single burst transaction of "BYTECOUNT" bits.

The Switch Processor takes the rpcim_done interrupt:

- Switch Processor does a clr_rpcim_done flip at beginning of the interrupt routine, which deasserts the rpcim_done interrupt and clears state of the RPCIM module (returns the internal state machine to IDLE and resets data FIFO write and read pointers).
- if (the Switch Processor wants to continue the DMA transfer) the Switch Processor firmware computes the next PCI address and writes the new ByteWriteAddress into RPCIADDR:
    - ByteWriteAddress = ByteWriteAddress + sizeof_last_BYTECOUNT (which is probably 64);
- Switch Processor firmware writes data for the next transfer to the RPCIDATA register
- Switch Processor writes the RPCIMD register with BYTECOUNT of new transaction.

Outline of RPCIM memory/configuration READ transactions:

Although there is only one hardware mode of operation for performing read transactions the Switch Processor firmware can determine how transactions of larger than 64 bytes are handled:

- whether the entire block transfer will occur on the PCI bus as a single PCI bus transaction (which may result in greater latency of other internal PCI master read consumers to complete a PCI read transaction). The maximum number of bytes that can be read in a single transaction by the PM3351 is 255.

- or as a series of 1-64 bytes transactions.

The PCI read transaction will occur on the PCI bus as one burst read transaction.

The PCI read transfer is initiated by writing the RPCIADDR and RPCICMD registers. The write to the RPCICMD register signals the RPCIM internal state machine to start the PCI master read or configuration read transaction. The RPCIM must arbitrate with the other internal DMA channels that can initiate PCI read transactions for access to a shared hardware resource: the Master Read Command (MCF) and Master Read Data (MRF) FIFOs that are part of the internal PCI bus interface logic (PCI_BIU). The arbiter for access to the MCF/MRF FIFOs is internal to the RPCIM logic. The grant for the read interface, `rrd_gt', is set on the cycle before the RPCIM logic writes the MCF CMD word to the MCF FIFO.

When the desired read transaction is occuring on the PCI bus the read data is written into the internal RPCIM data FIFO until the first of these occur:

- 64 bytes of data have been read from the MRF FIFO

- or the internal BYTECOUNT size field has decremented to 0 indicating that all data has been read.

The rpcim_done line is asserted indicating that the Switch Processor can burst read the RPCIMDATA register to access the read data. Coincident with assertion of rpcim_done, if the next_BYTECOUNT field has decremented to `0' then the internal arbiter bit "rrd_gt" is deasserted (this will always be true if RPCIM.BYTECOUNT is never written to greater than 64). next_BYTECOUNT is an internal counter that is preloaded with BYTECOUNT and decrements as data is read.

The Switch Processor takes the rpcim_done interrupt:

- Switch Processor reads the read data from the RPCIMDATA control register. Every read returns either one or two valid data bytes and increments the internal RPCIM data register address and decrements next_BYTECOUNT.

- after the Switch Processor has read all of the data that it wants to read out of the RPCIM data FIFO it pulses the `clr_rpcim_done' line.

Upon assertion of the `clr_rpcim_done' line the rpcim_done line is deasserted and next_BYTECOUNT is assigned to RPCIM.BYTECOUNT. If RPCIM.BYTECOUNT is `0'

then the present transaction is done and the `rpcim_idle' output is asserted. However, if the RPCIM.BYTECOUNT field is greater than `0' (which would have occured only if the initial BYTECOUNT was greater than 64) the Master Read FIFO continues to be read and the data written to the RPCIM data FIFO.

As previously stated, the internal `rrd_gt' signal is always deasserted when it has been determined that the last byte of the PCI read transaction has been read by the RPCIM module from the MRF FIFO into the RPCIM internal data FIFO.

RPCIM Data FIFO: Data Alignment

The RPCIM module does NO alignment of data with respect to data written to the MWF FIFO and data read from the MRF FIFO. The internal PCI Bus interface logic is expecting that data is left aligned with respect to these FIFOs:

- if a single byte of data is to be written on the PCI bus, the MWF FIFO is expecting the data to be on bytelane 3.

- if a single byte of data is to be read from the PCI bus, the MRF FIFO will return the data byte on bytelane 3.

This data alignment is preserved in accessing the RPCIM data FIFO using the RPCIDATA register. Data to be written to the MWF FIFO is written into the RPCIDATA register from "left-to-right" and "first-to-last" order. This is regardless to the actual byte address offset (i.e. AD [1:0] of the transaction as it appears on the PCI bus during the address phase of the transaction). So if a 5-byte write transaction is desired, with D0 representing the data to be written starting at the 32-bit ByteWriteAddress and D4 the data to be written to "ByteWriteAddress+4" the Switch Processor firmware must do the following:

- write RPCIMADDRh/l register to "ByteWriteAddress"

- write RPCIDATA with first two data bytes: {D0,D1}

- write RPCIDATA with next two data bytes: {D2,D3}

- write RPCIDATA with last data byte: {D4,8'h0}

- write the RPCICMD register:
  BYTECOUNT=8'h5, cmd[1:0]=memory_write_cmd

Data alignment on read transactions is similar. Data to be read from the MRF FIFO is read from the RPCIDATA register from "left-to-right" and "first-to-last" order. Once again, this is regardless to the actual byte address offset as seen on the PCI bus during the address phase of the read transaction. So if a 5-byte read transaction is desired,

with D0 representing the data read from ByteReadAddress and D4 the data be read from "ByteReadAddress+4" the Switch Processor firmware must do the following:

- write RPCIMADDRh/l register to "ByteReadAddress"
- write the RPCICMD register:
  BYTECOUNT=8'h5, cmd[1:0]=memory_read_cmd
- the read transaction will then occur on the PCI bus. After the transaction has completed and all 5 data bytes are in the RPCIM data FIFO, the rpcim_done interrupt is asserted.
    - during the firmware service routine the Switch Processor will read the RPCIDATA register:
    - 1st read from RPCIDATA returns first two data bytes: {D0,D1}
    - 2nd read from RPCIDATA returns next two data bytes: {D2,D3}
    - 3rd read from RPCIDATA returns the last data byte: {D4,8'hX}

RPCIM: zero-length transactions

If the Switch Processor firmware tries to initiate an RPCIM transaction having a BYTECOUNT of size `0' then the RPCIM internal state machine will not leave state IDLE.

RPCIM: writing to RPCICMD.BYTECOUNT field if the RPCIM is not IDLE

If the `rpcim_idle' coprocessor test condition line returns 0 (indicating that the RPCIM module has a transaction in progress) then the BYTECOUNT field of the RPCICMD register must NOT be written. Doing so will lead to unexpected and possibly fatal results (i.e. PCI bus master error).

RPCIM: internal arbitration to PCI_BIU shared hardware resources

Since the RPCIM must share the PCI_BIU master write and read hardware with other PCI (i.e. DMA) write and read channels on the PM3351 there must be some way to determine which consumer is getting access to the interface. This is accomplished using a simple request-grant protocol.

On the PM3351 the request/grant protocol betweent the RPCIM logic and the various other DMA channels is done completely in hardware and is transparent to the programmer.

Register 0x30 (cr16): RPCIM command register (RPCIMCMD)

Bit	Type	Function	Default
15		Unused	0
14	R	MERR	0
13	R/W	MERGE	0
12	R/W	BSWP	0
11:10		Unused	0
9:8	R/W	CMD	0
7:0	R/W	BYTECOUNT	0

MERR:

PCI Bus Interface logic Master Error status.

MERGE:

this is the value of the `merge' bit in the command word written to the PCI_BIU MWF/MCF FIFO. It must be written to a logic 0 for correct device operation (a PCI bus transaction master abort will occur if the bit is incorrectly written to a logic 1).

BSWP:

this is the value of the `byteswap' bit in the command word written to the MWF/MCF FIFO. Byte-swap swaps the way that bytes are presented on the PCI bus (refer to description on Byte Ordering for additional details).

CMD[1:0]:

Defines type of transfer to be performed.

- 00 - memory read command
- 01 - memory write command
- 10 - configuration read command
- 11 - configuration write commands

BYTECOUNT[7:0]:

Byte count of PCI transaction. When non-zero, a PCI transaction will be initiated.

Register 0x31 (cr17): RPCIM Data register (RPCIDATA)

Bit	Type	Function	Default
15:0	R/W	RPCIDATA[15:0]	0

RPCIDATA[15:0]:

RPCIM data FIFO access register.

- for memory write or configuration write transactions, data that is to be written to the PCI bus is first **written** to this register.
- for memory read or configuration read transactions, the data that was read during the PCI transaction is available by **reading** this register.

Register 0x32 (cr18): RPCIM Address register low (RPCIMADDRL)

Bit	Type	Function	Default
15:0	R/W	RPCIAD[31:16]	0

Register 0x33 (cr19): RPCIM Address register high (RPCIMADDRH)

Bit	Type	Function	Default
15:0	R/W	RPCIAD[15:0]	0

RPCIAD[31:0]:

the contents of this register are used as the address for the PCI transaction that is initiated by the RPCIM logic.

Register 0x34 (cr20): RPCIM status register (RPCISTAT)

Bit	Type	Function	Default
Bit 15 to 12	R	S[3:0]	0
Bit 11 to 8		Unused	0
Bit 7	R	RWR_GT	0
Bit 6	R	DMAPWR_GT	0
Bit 5	R	RWR_RQ	0
Bit 4	R	DMAPWR_RQ	0
Bit 3	R	RRD_GT	0
Bit 2	R	DMAPRD_GT	0
Bit 1	R	RRD_RQ	0
Bit 0	R	DMAPRD_RQ	0

S[3:0]:
Current state of RPCIM state machine

RWR_GT:
Write grant to RPCIM for access to internal PCI_BIU master write interface.

DMAPWR_GT:
Write grant to other DMA channels for access to internal PCI_BIU master write interface.

RWR_RQ:
RPCIM Request for access to internal PCI_BIU master write interface.

DMAPWR_RQ:
other DMA channels Request access to internal PCI_BIU master write interface.

RRD_GT:
Read grant to RPCIM for access to internal PCI_BIU master read interface.

DMAPRD_GT:
Read grant to other DMA channels for access to internal PCI_BIU master read interface.

RRD_RQ:
RPCIM Request for access to internal PCI_BIU master read interface.

DMAPRD_RQ:
DMA Request for access to internal PCI_BIU master read interface.

Register 0x35 (cr21): RST module: Watchdog Timer (WTIMER)

Bit	Type	Function	Default
Bit 15 to 0	R/W	WTIMER[15:0]	0xFFFF

The watchdog timer provides the ability for the device to generate a reset if program execution experiences a fatal delay. By default, this feature is disabled.

The hardware associated with the watchdog reset is:

- a prescaler for the 50 MHz SYSCLK that divides it down to get a pulse every 1 millisecond
- a 16-bit counter driven from the 1 millisecond pulse
- a comparator that checks when the counter is either 0 or 0xffff (decimal 65535)
- disable logic that stops the counter if it is 0xffff

If firmware running on the Switch Processor want to maintain a watchdog timer then it simply loads the WTIMER register with a non-zero value that is less than 0xffff; the WTIMER register will count down in 1 msec intervals and ultimately cause a chip reset when it reaches zero unless the Switch Processor reloads the WTIMER register with a new non-zero value. When the WTIMER register decrements to 0x0000 (this event is called a watchdog timeout) the WTIMER register will roll-over to 0xffff after 1 millisecond.

In the event of a watchdog timeout, the following occurs:

- reset is applied for 1 millisecond internal to the core logic on the PM3351 device. This includes resetting all internal logic (including the Switch Processor itself) **except** for the PCI bus interface logic.
- the ERST_ output will be asserted for 10 milliseconds. As previously described, the ERST_ output is open drain and can be tied directly to the RST_ input of all of the ELAN 1x100 and ELAN 8x10 chips in a system. If ERST_ is directly connected to RST_, then a watchdog timeout occuring on any of these devices will cause a hardware reset to be applied for 10 milliseconds; this hardware reset will reset the core logic and the PCI bus interface logic of each of the ELAN 1x100 and ELAN 8x10 devices.

WTIMER[15:0]:

The binary value of the register is decremented every millisecond provided it is not 0xFFFF. When the count reaches 0x0000, the device will be reset for 1ms.

If the RISC wants to turn off the watchdog, it loads the counter with 0xFFFF. This is also the default state upon assertion of the RST* input.

If the RISC wants an immediate reset via software, it loads the counter with 0x0000.

Register 0x36: RST module: Firmware RST reg (cr22)

Bit	Type	Function	Default
Bit 15	R/W	ERST	0
BIT 14	R/W	Unused	0
BIT 13	R/W	MRST	0
BIT 12 to 5	R/W	LINK_RST[7:0]	0
BIT 4	R/W	LBLOGIC_RST	0
BIT 3	R/W	DMA_RST	0
BIT 2	R/W	RPICM_RST	0
BIT 1	R/W	HASH_RST	0
BIT 0	R/W	CTR_BANK_RST	0

CTR_BANK_RST:

If this bit is a logic 1, the Counter Bank functional block is held in its reset state.

HASH_RST:

If this bit is a logic 1, the Hash functional block is held in its reset state.

RPICM_RST:

If this bit is a logic 1, the RPCIM functional block is held in its reset state.

DMA_RST:

If this bit is a logic 1, the DMA functional block is held in its reset state.

LINK_RST[7:0]:

If a LINK_RST bit is a logic 1, the associated Link block is held in its reset state.

MRST:

ERST:

If ERST transitions from logic 0 to logic 1, the ERST_ output pin is asserted low for 10 ms (assuming 50MHz SYSCLK).

Register 0x37: DBG module Control Register (cr23)

Bit	Type	Function	Default
Bit 15		Unused	X
Bit 14 to 8	R/W	SSEL[6:0]	0
Bit 7 to 4		Unused	X
Bit 3	R/W	DBGMODE[2]	0
Bit 2	R/W	DBGMODE[1]	0
Bit 1	R/W	DBGMODE[0]	0
Bit 0	R/W	OE_DBG	0

This register is also accessible via the PCI interface at address 0x00FF00A0.

OE_DBG:

If this bit is a logic 1, the DBG[2:0] outputs drive valid logic levels; otherwise DBG[2:0] are high impedance.

DBGMODE[2:0]:

These bits determine the content and format of the data presented on the DBG[2:0] outputs.

DBGMODE[2:0]

000 The DBG[2:0] outputs present which memory consumer currently has been granted access. The eight consumers are encoded as follows:

DBG[2:0]
- 000 lmgt RISC
- 001 imgt RISC
- 010 DMA
- 011 HOSTIF
- 100 HASH
- 101 Unused
- 110 Unused
- 111 Unused

001 This mode allows the current program counter (PC) to be output simultaneously with a selected internal signal. The 22 most significant bits of the PC are serialized as 2-bit

data on DBG[2:1]. The PC is transmitted starting with the least significant bits (PC[3:2]) and is shifted every SYSCLK period. The DBG[0] output presents the internal signal selected by the SSEL[6:0] register bit.

010 This mode allows the current program counter (PC) to be output with a alignment marker. The PC is output as described above, but an active high pulse aligned to PC[3:2] is generated on the DBG[0] output.

SSEL[6:0]:

The SSEL bits select one of 128 internal signals for presentation on the DBG[0] output when OE_DBG is a logic 1 and DBGMODE[2:0] is either 3'b001 or 3'b011.

SSEL	Signal	SSEL	Signal	SSEL	Signal
127	c_win_sel	84	collision[5]	41	DMAprd_gt
126	hmrdy	83	collision[4]	40	DMApwr_gt
125	smrdy	82	collision[3]	39	rpcim_idle
124	dmrdy	81	collision[2]	38	rpcim_done
123	imrdy	80	collision[1]	37	mwf_ld
122	lmrdy	79	collision[0]	36	mcf_ld
121	sra_newaddr	78	mrf_valid	35	mrf_unld
120	sra_active	77	flist_lock	34	hashbusy[7]
119	sra_done	76	set_ch[2]	33	hashbusy[6]
118	sra_lastword	75	set_ch[1]	32	hashbusy[5]
117	swf_valid	74	set_ch[0]	31	hashbusy[4]
116	srfwen	73	lbwen	30	hashbusy[3]
115	swfren	72	lbren	29	hashbusy[2]
114	hwr	71	lbsel[2]	28	hashbusy[1]
113	hrd	70	lbsel[1]	27	hashbusy[0]
112	bkpt	69	lbsel[0]	26	dma_we1
111	merr	68	d_intr[9]	25	dma_we2
110	tframe[7]	67	d_intr[8]	24	dma_we3
109	tframe[6]	66	d_intr[7]	23	dstfail
108	tframe[5]	65	d_intr[6]	22	srcfail
107	tframe[4]	64	d_intr[5]	21	cop_cond[7]
106	tframe[3]	63	d_intr[4]	20	cop_cond[6]
105	tframe[2]	62	d_intr[3]	19	cop_cond[5]
104	tframe[1]	61	d_intr[2]	18	cop_cond[4]
103	tframe[0]	60	d_intr[1]	17	cop_cond[3]
102	rframe[7]	59	d_intr[0]	16	cop_cond[2]
101	rframe[6]	58	exp_idle[2]	15	cop_cond[1]
100	rframe[5]	57	exp_idle[1]	14	cop_cond[0]
99	rframe[4]	56	exp_idle[0]	13	intr[12]
98	rframe[3]	55	dbusy[7]	12	intr[11]
97	rframe[2]	54	dbusy[6]	11	intr[10]
96	rframe[1]	53	dbusy[5]	10	intr[9]

95	rframe[0]	52	dbusy[4]	9	intr[8]
94	empty[7]	51	dbusy[3]	8	intr[7]
93	empty[6]	50	dbusy[2]	7	intr[6]
92	empty[5]	49	dbusy[1]	6	intr[5]
91	empty[4]	48	dbusy[0]	5	intr[4]
90	empty[3]	47	req_lock	4	intr[3]
89	empty[2]	46	req_int	3	intr[2]
88	empty[1]	45	ack_int	2	intr[1]
87	empty[0]	44	req_present	1	intr[0]
86	collision[7]	43	DMAprd_rq	0	stall_
85	collision[6]	42	DMApwr_rq		

Register 0x38: HASH ram register 15:0 (cr24)

Bit	Type	Function	Default
Bit 15 to 0	R/W	Ramreg[15:0]	0

Register 0x39: HASH ram register 31:16 (cr25)

Bit	Type	Function	Default
Bit 31 to 16	R/W	Ramreg[31:0]	0

This is the register that the CPU uses to interface to the MAC_RAM and to the HRESULT_RAM . This register is intended to only be used for factory debug and testing purposes.

Register 0x3A: HASH control address register (cr26)

Bit	Type	Function	Default
Bit 15 to 7	R/W	Reserved	0
Bit 5 to 0	R/W	HCADDR[5:0]	0

HCADDR[5:0]: Decodes which control register the HCDATA register contents are intended for.

Value	Resulting Register
1	HREG_ADDR
2	HRESULTL_ACC
3	HRESULTH_ACC
4	MACL_ACC
5	MACM_ACC
6	MACH_ACC

Register 0x3B: HASH control data address (cr27)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HCDATA[15:0]	0

HCDATA[15:0]: Data to be read or written to the register selected by the Hash address control register (cr26)

Register 0x40: Interframe gap time (1st part) (cr32)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IFS1TIME[7:0]	d30

IFS1TIME[7:0]:

The number of bit times waited after a transmit, regardless of noise on the line.

Register 0x41: Interframe gap time (2nd part) (cr33)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7to 0	R/W	IFS2TIME[7:0]	d60

IFS2TIME[7:0]:

The number of bit times waited after IFS1TIME at the end of which another transmission is permitted so long as collision jamming is not active.

Register 0x42: Preamble time (cr34)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	PRESIZE[7:0]	d63

PRESIZE[7:0]:

Size of the preamble before the SFD in bit times. The value must be odd.

Register 0x43: Jam time (cr35)

Bit	Type	Function	Default
Bit 15 to 12		Unused	0
Bit 11 to 0	R/W	JAMSIZE[11:0]	0x004

JAMSIZE[11:0]:

The length of a jam sent after a collision in byte times.

Register 0x44: Maximum frame size (cr36)

Bit	Type	Function	Default
Bit 15 to 12		Unused	0
Bit 11 to 0	R/W	MAXSIZE[11:0]	d1518

MAXSIZE[11:0]:

Number of bytes at which an outgoing frame is truncated, and the number of bytes beyond which an incoming frame is considered to be long.

Register 0x45: Minimum frame size (cr37)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	MINSIZE[7:0]	d64

MINSIZE[7:0]:

Number of bytes below which a frame is considered to be short.

Register 0x46: Blind time (cr38)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	BLIND[7:0]	d40

BLIND[7:0]:

Blind time in bit periods. Currently does not work

Register 0x47: Loopback byte (cr39)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	LBKBYTE[7:0]	0x69

LBKBYTE[7:0]:

Used by all links as the transmission data when the loopback is a enabled.

Register 0x48: Packet buffer size (in bytes) (cr40)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	PBSIZE[7:0]	d80

PBSIZE[15:0]:

bytes

Register 0x49: Increment channel data word low (cr41)

Bit	Type	Function	Default
Bit 15 to 0	R/W	INCDATA[15:0]	0x0000

Register 0x4A: Increment channel data word high (cr42)

Bit	Type	Function	Default
Bit 15 to 0	R/W	INCDATA[31:16]	0x0000

INCDATA[31:0]:

Register 0x4B: Mac control frame type field (cr43)

Bit	Type	Function	Default
Bit 15 to 0	R/W	ETYPE[15:0]	0x8808

ETYPE[15:0]:

The ETYPE field is compared to the Ethernet type field in an incoming frame to see if it is a control frame. If so, a GPI bit is set.

Register 0x4C: Late time (cr44)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	LATETIME[7:0]	d56

LATETIME[7:0]:

The number of bit periods after the start of a frame after which a collision is considered late.

Serial Communications Controller logic (SCC module)

The SCC module is instantiated within the DBG module of the PM3350 chip. This module is intended to allow the DBG[2:0] pads on the PM3350 to provide additional functions beyond the diagnostic output provided by the DBG[2:0] pins if used solely with the DBG module itself. Two uses of the SCC are:

- driving/interfacing to external hardware, such as an 8 pin non-volatile ram or the like
- providing an RS232 serial port to allow remote diagnostics,configuration, and possibly management access. For use with RS232 an external circuit is required for voltage conversion.

The SCC module uses three firmware accessible registers: SCC_CTRL, SCC_STAT, and SCC_DATA. These three registers are used for control, status, and serial data in/out, respectively. The external interrupt line going to the RISC processor is the logical OR of the PM3350 external interrupt pin, MINTR*, and the SCC rx interrupt and and SCC tx interrupt lines, where the SCC interrupts are maskable. The purpose of the status register is to provide the following features:

- allows firmware to decode the source of the external interrupt line
- provide status on whether the SCC rx and tx ports are active
- read the current value of the DBG[2:0] pins.

The SCC control register is used to configure the operation of the SCC module. Among other things, it selects the baud rate to be used for reading/writing of the serial data, selects whether each of the DBG[2], DBG[1], and DBG[0] lines will be configured to drive or receive data, and enables the SCC rx and tx interrupts. The SCC_DATA register is one byte: if written the value is placed in the TX holding register; if read, the next byte of the SCC RX fifo is read out. The baud rate select value chooses one of eight possible prescaler values that can be used to divide down the PM3350 internal RealTimeClock (which will usually be selected to be at 1 MHz).

The TX module has two bytes of storage: a holding register and a shift register. Writing to the SCC_DATA register validates the data in the holding register. Whenever the shift register is empty, the holding register, if valid, is loaded into the shift register, thereby starting the transmit cycle. If the SCC TX interrupt is enabled, the interrupt will occur whenever the hold register is empty. **The serial data out line is always driven on the DBG[0] pin, if enabled.**

The RX module is basically a one-byte deserializer and a 16 byte FIFO. Whenever a complete byte of data is received, it is written to the next non-valid element of the FIFO. The SCC RX interrupt will be asserted if there is at least one valid byte in the FIFO and

the RX interrupt is enabled. **The serial data in line is mapped to the DBG[1] pin input buffer.**

Register 0x50: SCC_STAT (cr48)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7	R	MINT_STAT	MINTR* pin value
Bit 6	R	SCCRXintra	0
Bit 5		Unused	0
Bit 4	R	TXsending	0
Bit 3	R	SCCTXintra	0
Bit 2 to 0	R	DBG_I[2:0]	DBG[2:0] pin value

MINT_STAT:

This bit is the logical state of the MINTR* pin of the device after it has been synchronized to SYSCLK. It can be used by firmware to decode whether interrupt_11 on the PM3350 is due to the MINTR* pin being asserted or, through decoding other bits in the SCC_STAT register, that the interrupt is because of the SCC RX interrupt or SCC TX interrupt lines.

SCCRXintra:

SCC RX interrupt active. Asserted when the SCC RX FIFO has at least one bye of valid data. The SCCRXintra bit is logically ANDed with a mask bit before it can set the SCC_intr line.

TXsending:

This status bit will be asserted when there is no valid data to be transmit on the serial port. It indicates that both the one-byte holding and one-byte serializer registers are empty in the SCC TX port.

SCCTXintra:

SCC TX interrupt active. This status bit will be asserted when the one-byte SCC TX holding register is empty, indicating that the next byte of data to be transmit can be written into the SCC_DATA register. The SCCTXintra bit is logically ANDed with a mask bit before it can set the SCC_intr line.

DBG_i[2:0]:

Value of the DBG[2:0] pin input buffers.

Register 0x51: SCC_CTRL (cr49)

Bit	Type	Function	Default
Bit 15 to 11	R/W	Unused	0
Bit 10 to 8	R/W	BAUD_SEL{2:0]	0
Bit 7	R/W	SCC_DBE_OE[2]	0
Bit 6	R/W	SCCRXintr_en	0
Bit 5	R/W	SCC_DBG_OE[1]	0
Bit 4	R/W	SCCTX_en	0
Bit 3	R/W	SCCTXintr_en	0
Bit 2	R/W	DBG2_dout	0
Bit 1	R/W	DBG1_dout	00
Bit 0	R/W	DBG0_dout	

BAUD_SEL[2:0]:

Selects baud rate to be used by SCC module. This selects the value to be used to divide down the internal 1MHz RealTimeClock (refer to DCONFIG RTCDIV setting for additional information on the RealTimeClock). The baud rate is given by the following formula:

frequency_RealTimeClock / (26 * prescale_divider_value).

Baud_sel[2:0]	Prescale divider value	Baud rate (bit/sec)
0	128	300
1	64	600
2	32	1200
3	16	2400
4	8	4800
5	4	9600
6	2	19200
7	1	38400

SCC_DBG_OE[2]:

Output enable for the DBG[2] pin. The DBG[2] pin will be driven by the PM3350 if either the SCC_DBG_OE[2] bit is set –OR- if the DBGCTRL register output enable bit is set.

SCCRXintr_en:

Mask bit for SCCRXintra bit (see SCC_STAT register).

SCC_DBG_OE[1]:

Output enable for the DBG[1] pin. The DBG[1] pin will be driven by the PM3350 if either the SCC_DBG_OE[1] bit is set –OR- if the DBGCTRL register output enable bit is set.

SCCTX_en:

Enable SCC serial transmit. If a logical 1, then output enable for the DBG[0] pin is asserted.

SCCTXintr_en:

This bit has a dual purpose depending on the value of the SCCTX_en bit. If SCCTX_en is a logical 1: then this bit is used as the mask bit for the SCCTXintra bit (see SCC_STAT register). If SCCTX_en is a logical 0: then thi bit is used as the output enable for the DBG[0] pin.

DBG2_out:

Value to be driven on DBG[2] pin if SCC_DBG_OE[2] is set.

DBG1_out:

Value to be driven on DBG[1] pin if SCC_DBG_OE[1] is set.

DBG0_out:

Value to be driven on DBG[0] pin if (SCCTX_en is a logic 0) –AND- (SCCTXintr_en is a logic 1).

Register 0x52: SCC_DATA (cr50)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	SCCdata[7:0]	0

SCCdata[7:0]:

If the control register is *written*, then SCCdata[7:0] will be written to the one byte TX holding register and the register valid bit will be set; if the register was previously valid, the contents are overwritten with the new value. If the control register is*read,* then SCCdata[7:0] is the value of the element in the SCC RX FIFO that is pointed to by the FIFO readpointer. The byte of data

may or may not be “valid”; as stated above, the SCCRXintra bit of the SCC_STAT register will be a logic 1 if the FIFO read data is valid.

Register 0x5F: DMA channel register window (cr63)

Bit	Type	Function	Default
Bit 15 to 4		Unused	X
Bit 3 to 0	R/W	WIN[3:0]	0

WIN[3:0]: Selects bank of windowed link registers.

Register 0x70: Link control word (cr80)

Bit	Type	Function	Default
Bit 15 to 12		Unused	X
Bit 10	R/W	LWCTRL[11]	0
Bit 10	R/W	LWCTRL[10]	0
Bit 9	R/W	LWCTRL[9]	0
Bit 8	R/W	LWCTRL[8]	0
Bit 7	R/W	LWCTRL[7]	0
Bit 6	R/W	LWCTRL[6]	0
Bit 5	R/W	LWCTRL[5]	0
Bit 4	R/W	LWCTRL[4]	0
Bit 3	R/W	LWCTRL[3]	0
Bit 2	R/W	LWCTRL[2]	0
Bit 1	R/W	LWCTRL[1]	0
Bit 0	R/W	LWCTRL[0]	0

LWCTRL[11]:

Enable little endian byte ordering

LWCTRL[10]:
Invert ten input

LWCTRL[9]:
Invert col input

LWCTRL[8]:
Invert cd input

LWCTRL[7]:
Invert rclk input

LWCTRL[6]:
Invert tclk input

LWCTRL[5]:
Force jam on line

LWCTRL[4]:
Reset the tx_if

LWCTRL[3]:
Reset the rx_if

LWCTRL[2]:
Flush the fifo

LWCTRL[1]:
Append crc to frames

LWCTRL[0]:
Transmit enable (Also set by set_transmit flip on the channel)

Register 0x71: Link Status word (cr81)

Bit	Type	Function	Default
Bit 15	R	LWSTAT[15]	X
Bit 14	R	LWSTAT[14]	X
Bit 13	R	LWSTAT[13]	X
Bit 12	R	LWSTAT[12]	X
Bit 11	R	LWSTAT[11]	X
Bit 10	R	LWSTAT[10]	X
Bit 9	R	LWSTAT[9]	X
Bit 8	R	LWSTAT[8]	X
Bit 7	R	LWSTAT[7]	X
Bit 6	R	LWSTAT[6]	X
Bit 5	R	LWSTAT[5]	X
Bit 4	R	LWSTAT[4]	X
Bit 3	R	LWSTAT[3]	X
Bit 2	R	LWSTAT[2]	X
Bit 1	R	LWSTAT[1]	X
Bit 0	R	LWSTAT[0]	X

Writing to this register thaws the link to receive new data.

LWSTAT[15:13]:
Interrupt code (provided by DMA)

LWSTAT[12]:
First word in transfer (provided by DMA)

LWSTAT[11]:
CRC error (provided by DMA)

LWSTAT[10]:
Received frame too long

LWSTAT[9]:
Received frame runt

LWSTAT[8]:
Alignment error

LWSTAT[7]:
Fifo overrun

LWSTAT[6]:
Collision status

LWSTAT[5]:
Transmitted runt

LWSTAT[4]:
Transmitted long frame

LWSTAT[3]:
Fifo status - full

LWSTAT[2]:
Fifo status - half_full

LWSTAT[1]:
Fifo status - empty

LWSTAT[0]:
transmitting preamble

Register 0x72: Backoff timer (cr82)

Bit	Type	Function	Default
[15:10]	R/W	Reserved	X
[9:0]	R/W	LWBACK	0

LWBACK[10:0]: Link backoff counter.

Register 0x73: Number of Packet Buffers (cr83)

Bit	Type	Function	Default
[15:8]	R/W	Reserved	X
[7:0]	R/W	LWNBUFFS	0

LWNBUFFS[8:0]: When Link Channel 0 through 7 are selected LWNBUFFS is the local link number of packet buffers counter. When the widowed register selects the PCI Increment Channel the LWNBUFFS registers hold the PCI chip select flag.

Register 0x74: First PB address low (cr84)

Bit	Type	Function	Default
[15:0]	R/W	LWFIRSTl	0

Register 0x75: First PB address high (cr85)

Bit	Type	Function	Default
[15:8]	R/W	Reserved	X
[7:0]	R/W	LWFIRSTh	0

LWFIRST[23:0]: When Link Channel 0 through 7 are selected LWFIRST equals the local link first packet buffer pointer. When the windowed register selects the PCI Increment Channel the LWFIRST registers contain the request offset address value for this chip.

Register 0x76: Last PB address low (cr86)

Bit	Type	Function	Default
[15:0]	R/W	LWLASTl	0

Register 0x77: Last PB address high (cr87)

Bit	Type	Function	Default
[15:8]	R/W	Reserved	X
[7:0]	R/W	LWLASTh	0

LWLAST[23:0]: When Link Channel 0 through 7 are selected LWLAST equals the local link last packet buffer pointer address. When the windowed register selects the PCI Increment Channel the LWLAST registers contain the base PCI address value for the other chips in the system. The following table describes the bit correspondence.

LWLAST[7:0]	PCI Chip Base Address [7:0] chip 0
LWLAST[15:8]	PCI Chip Base Address [7:0] chip 1
LWLAST[23:16]	PCI Chip Base Address [7:0] chip 2
LWLAST[31:24]	Reserved

When the windowed register selects the PCI Write Channel the LWLAST registers contain the base local source memory pointer address for the Write Channel transaction. The following table describes the bit correspondence.

LWLAST[15:0]	PCI write local memory pointer[15:0]
LWLAST[23:16]	PCI write local memory pointer[23:0]
LWLAST[31:24]	Reserved

Register 0x78: Remote PB address high (cr88)

Bit	Type	Function	Default
[15:0]	R/W	LWPADRl	0

Register 0x79: Remote PB address high (cr89)

Bit	Type	Function	Default
[15:8]	R/W	Reserved	X
[7:0]	R/W	LWPADRh	0

LWPADR[23:0]: When Link Channel 0 through 7 are selected LWPADR equals the local link remote packet buffer pointer address. When the windowed register selects the PCI Increment Channel the LWPADR registers contain the base PCI address value for the other chips in the system. The following table describes the bit correspondence.

LWPADR[7:0]	PCI Chip Base Address [7:0] chip 4
LWPADR[15:8]	PCI Chip Base Address [7:0] chip 5
LWPADR[23:16]	PCI Chip Base Address [7:0] chip 6
LWPADR[31:24]	PCI Chip Base Address [7:0] chip 7

When the windowed register selects the PCI Write Channel the LWPADR registers contain the base remote destination memory pointer address for the Write Channel transaction. The following table describes the bit correspondence.

LWPADR[15:0]	PCI write remote memory pointer[15:0]
LWPADR[23:16]	PCI write remote memory pointer[23:0]
LWPADR[31:24]	Reserved

Register 0x7A: DMA stats part 2 (cr90)

BIT	Type	Function	Default
Bit 15 to 9		Unused	X
Bit 8	R	DMSTAT2[8]	X
Bit 7	R	DMSTAT2[7]	X
Bit 6	R	DMSTAT2[6]	X
Bit 5	R	DMSTAT2[5]	X
Bit 4	R	DMSTAT2[4]	X
Bit 3	R	DMSTAT2[3]	X
Bit2	R	DMSTAT2[2]	X
Bit 1	R	DMSTAT2[1]	X
Bit 0	R	DMSTAT2[0]	X

DMSTAT2[8]:

Packet buffer allocated to frame. (window decoded)

DMSTAT2[7]:

First word has not been transferred. (window decoded)

DMSTAT2[6]:

End of frame (window decoded)

DMSTAT2[5]:

End of transmit (window decoded)

DMSTAT2[4]:
Bad transmission (window decoded)

DMSTAT2[3]:
Free list empty (rxstop) (window decoded)

DMSTAT2[2]:
Collision (window decoded)

DMSTAT2[1]:
Overflow (window decoded)

DMSTAT2[0]:
Crc error (window decoded)

Register 0x7B: Local Link Last Frame Size (cr91)

Bit	Type	Function	Default
[15:8]	R/W	Reserved	X
[7:0]	R/W	LWSIZE[7:0]	0

LWSIZE[7:0]: When Link Channel 0 through 7 are selected LWSIZE equals the local link remote packet buffer pointer address. When the windowed register selects the PCI Increment Channel the LWSIZE registers contain the base PCI address value for the other chips in the system. The following table describes the bit correspondence.

LWSIZE[7:0]	PCI Chip Base Address [7:0] chip 3

When the windowed register selects the PCI Write Channel the LWSIZE registers contain the size in bytes of data to transfer via the Write Channel transaction. The following table describes the bit correspondence.

LWSIZE[7:0]	PCI write number of bytes[7:0]

Register 0x7C: HASH source result register low word (cr92)

Bit	Type	Function	Default
[15:0]	R/W	Source_result	0

Register 0x7D: HASH source result register high word (cr93)

Bit	Type	Function	Default
[31:0]	R/W	Reserved	X
[23:16]	R/W	Source_result	0

Source_result[23:0]: Resulting Hash bucket address for given source MAC address.

Register 0x7E: HASH destination result register low word (cr94)

Bit	Type	Function	Default
[15:0]	R/W	Destination_result	0

Register 0x7F: HASH destination result register high word (cr95)

Bit	Type	Function	Default
[31:24]	R/W	Reserved	X
[23:16]	R/W	Destination_result	0

Destination_result[23:0]: Destination hash bucket address for given MAC destination MAC address.

Interrupts

Int0: Link 0 interrupt
Int1: Link 1 interrupt
Int2: Link 2 interrupt
Int3: Link 3 interrupt
Int4: Link 4 interrupt
Int5: Link 5 interrupt
Int6: Link 6 interrupt
Int7: Link 7 interrupt
Int8: PCI read interrupt
Int9: PCI write interrupt (Or RPCIM if enabled)
Int10: Request/Acknowledge interrupt
Int11: External interrupt
Int12: Work queue interrupt (also called task counter interrupt.)

Interrupt Codes (only for link interrupts)

Code0: No interrupt present

Code1: --unused--
Code2: End of frame received
Code3: End of Transmit
Code4: Bad Transmit (fifo under-run)
Code5: End of frame received with errors
Code6: Free list emptied while receiving frame
Code7: Collision

The state of the DMA may fall into more than one interrupt type. They have an implicit priority of: 7,6,5,2,4,3,0

Coprocessor Condition Tests

Cop0: Channel idle (window decoded)
Cop1: Rxstop (DMA out of Frame Buffers. Link and PCI traffic halted)
Cop2: --unused--
Cop3: --unused--
Cop4: --unused--
Cop5: RPCIM idle
Cop6: RPCIM done
Cop7: --unused--
Cop8: --unused--
Cop9: --unused--
Cop10: --unused--
Cop11: --unused--
Cop12: General Purpose loop back line. (Set by RISC GPO line 25)
Cop13: General Purpose loop back line. (Set by RISC GPO line 26)
Cop14: General Purpose loop back line. (Set by RISC GPO line 27)
Cop15: General Purpose loop back line. (Set by RISC GPO line 28)

General-Purpose Inputs

GPI0: Alignment error (window decoded)
GPI1: Crc error (window decoded)
GPI2: Rx runt (window decoded)
GPI3: Rx Frame too long (window decoded)
GPI4: Fifo overrun (rx) (window decoded)
GPI5: MAC control frame (window decoded)
GPI6: REQ present
GPI7: Hash source fail (window decoded)
GPI8: Hash destination fail (window decoded)
GPI9: Hash channel busy (window decoded)
GPI10:--unused--
GPI11:--unused--
GPI12:--unused--

General-Purpose Outputs

Flip0: Free list lock off (pulse) - Unlock freelist
Flip1: Free list lock on (pulse) - Lock freelist. Note: When locking the free list a one cycle delay must be inserted between the time of the flip and the time the DMA free frame buffer control register may be read.
Flip2: RISC access to hash RAM (level) (1) RISC (0) other
Flip3: RISC Request lock. (level) (1) lock (0) other
Flip4: Req decrement (pulse)
Flip5: Ack decrement (pulse)
Flip6: Worklist increment (pulse)
Flip7: Worklist decrement (pulse)
Flip8: "Set transmit" bit. This flip starts a transfer on a given channel. (pulse)
Flip9: Hold off request interrupt for 512 clocks. This flip requires a one cycle delay before the request interrupt will be masked off.
Flip10: --unused--
Flip11: --unused--
Flip12: --unused--
Flip13: --unused--
Flip14: --unused--
Flip15: --unused--
Flip16: --unused--
Flip17: --unused--
Flip18: --unused--
Flip19: --unused--
Flip20: Enable RPCIM module for RISC access to PCI master
Flip21: --unused--
Flip22: --unused--
Flip23: When used in conjunction with flip8 on increment channel, does an acknowledge increment.
Flip24: --unused--
Flip25: General Purpose loop back line. (Monitored by coprocessor condition line 12)
Flip26: General Purpose loop back line. (Monitored by coprocessor condition line 13)
Flip27: General Purpose loop back line. (Monitored by coprocessor condition line 14)
Flip28: General Purpose loop back line. (Monitored by coprocessor condition line 15)
Flip29: RISC settable interrupt line
Flip30: Activate Alarm1 Interrupt
Flip31: Activate Alarm2 Interrupt

Programming Notes

General Switching Firmware Structure

Local Port Interrupt Handlers

Local Port Valid Receive Handler

Local Port Errored Receive Handler

Local Port Transmit Handler

Local Port Collision Handler

Local Port Transmit Under-run Handler

Local Port Out-of-Buffers Handler

Work Queue Interrupt Handler

Context Save/Restore

Unicast Frame Processing

Broadcast/Multicast Frame Processing

Address Learning

Special Frame Processing

Expansion Port Counter Interrupt Handler

Request Counter Handler

Acknowledge Counter Handler

Expansion Port Data Transfer Interrupt Handler

Alarm Interrupt Handlers

RTOS Alarm Service

Address Aging Alarm

Frame Lifetime Control Alarm

Background Task Dispatcher

Aging Process

Transmit Restart Process

SYSTEM IMPLEMENTATION CONSIDERATIONS

Power Supply Sequencing

Due to ESD protection structures in the pads it is necessary to exercise caution when powering a device up or down. ESD protection devices behave as diodes between power supply pins and from I/O pins to power supply pins. Under extreme conditions it is possible to blow these ESD protection devices or trigger latch up. The recommended power supply sequencing follows:

1) VDD5 (5V Bias) power must be supplied either before VDD or simultaneously with VDD.

2) To prevent damage to the ESD protection on the device inputs the maximum DC input current specification must be respected. This is accomplished by either ensuring that the VDD power is applied before input pins are driven or by increasing the source impedance of the driver so that the maximum driver short circuit current is less than the maximum DC input current specification (10 mA).

3) Power down the device in the reverse sequence.

System Reset

Device Configuration

PCI Interfacing

PCI Bus Arbitration

The ELAN 8x10 is not able to arbitrate its own PCI transactions, so an external PCI bus arbiter is needed. The arbiter receives requests via the REQ* lines, and gives grants via the GNT* lines. A simple round robin arbitration scheme works best.

Memory Bus Loading

ORDERING AND THERMAL INFORMATION

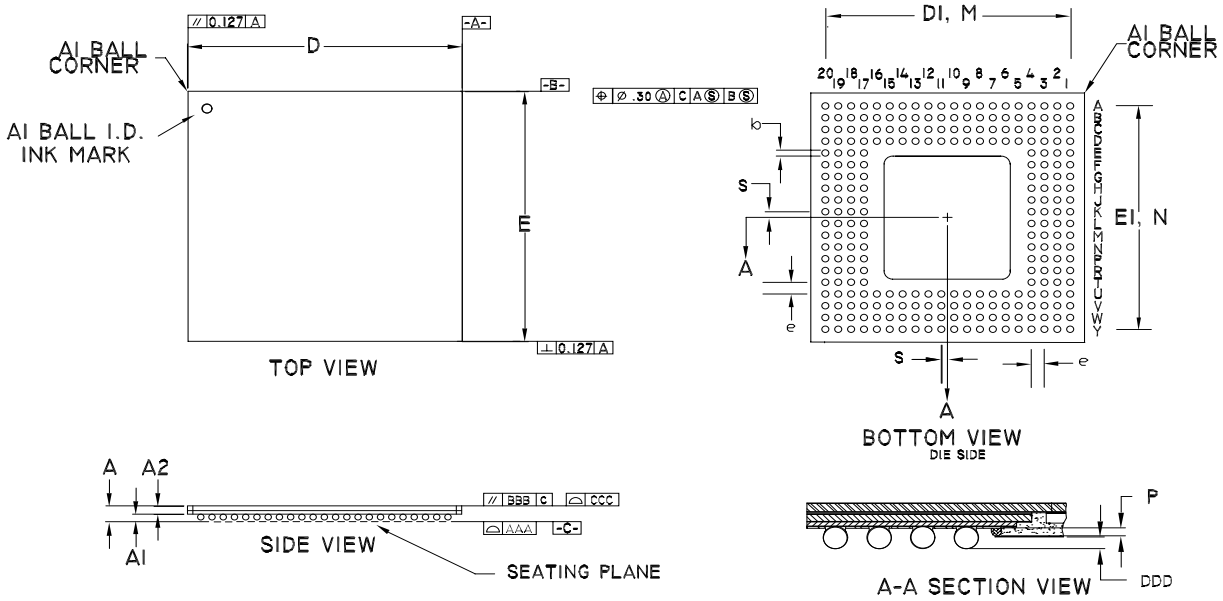
PART NO.	DESCRIPTION
PM3350-RC	256 Ball Grid Array (SBGA)
PM3350-SW	Unmanaged Switching Code Firmware

PART NO.	CASE TEMPERATURE	Theta Ja
PM3350-RC	-0°C to 70°C	21 °C/W

MECHANICAL INFORMATION

THERMALLY ENHANCED BALL GRID ARRAY(SBGA)

256 PIN SBGA -27x27 MM BODY - (B SUFFIX)



NOTES: 1) ALL DIMENSIONS IN MILLIMETER.
2) DIMENSION AAA DENOTES COPLANARITY
3) DIMENSION BBB DENOTES PARALLEL
4) DIMENSION CCC DENOTES FLATNESS

PACKAGE TYPE: 256 PIN THERMAL BALL GRID ARRAY																
BODY SIZE: 27 X 27 X 1.45 MM																
DIM.	A	A1	A2	D	D1	E	E1	M,N	e	b	AAA	BBB	CCC	DDD	P	S
MIN.	1.41	0.56	0.85	26.90	24.03	26.90	24.03			0.60				0.15	0.20	
NOM.	1.54	0.63	0.91	27.00	24.13	27.00	24.13	20X20	1.27	0.75				0.33	0.30	
MAX.	1.67	0.70	0.97	27.10	24.23	27.10	24.23			0.90	0.15	0.15	0.20	0.50	0.35	0.635

NOTES

CONTACTING PMC-SIERRA, INC.

PMC-Sierra, Inc.
105-8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: (604) 415-6000

Fax: (604) 415-6200

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Application Information: apps@pmc-sierra.com
Web Site: http://www.pmc-sierra.com

PMC-970109 (R3) ref PMC-961051 (R3) Issue date: April 1998