PM3351

ELAN 1X100

SINGLE PORT FAST ETHERNET SWITCH

DATA SHEET

ISSUE 3: FEBRUARY 1998

CONTENTS

FEATURES

- Single-chip, 1-port, full duplex or half duplex, 10/100BaseT switching 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., frame forwarding and filtering decisions).
- Performs frame switching at a rate of 200 Mbit/s (full duplex), 100 Mbit/s (half duplex).
- Fully compatible with the PM3350, 8-port 10 Mbit/s switch device; may be used to create a compact and inexpensive mixed 10/100 Mbit/s switching hub.
- Store-and-forward operation with full error checking and filtering.
- Filtering and switching at wire rates (up to 148,800 packets per second), supporting a mix of Ethernet and IEEE 802.3 protocols.
- Expansion port supports a peak system bandwidth of 1 Gbit/s, and is compatible with industry-standard PCI bus (version 2.1).
- Performs all address learning, address table management and aging functions for up to 32,768 MAC addresses (limited by external memory). Address learning rate of up to 100,000 addresses per second.
- Maximum broadcast/multicast at wire rates 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 frame lifetime control.

- Flow control supported for both full duplex and half duplex operation: supports IEEE 802.3x PAUSE frame flow control in full-duplex mode, and supports user-enabled backpressure flow control in half-duplex mode with configurable buffer thresholds and limits.

- 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.

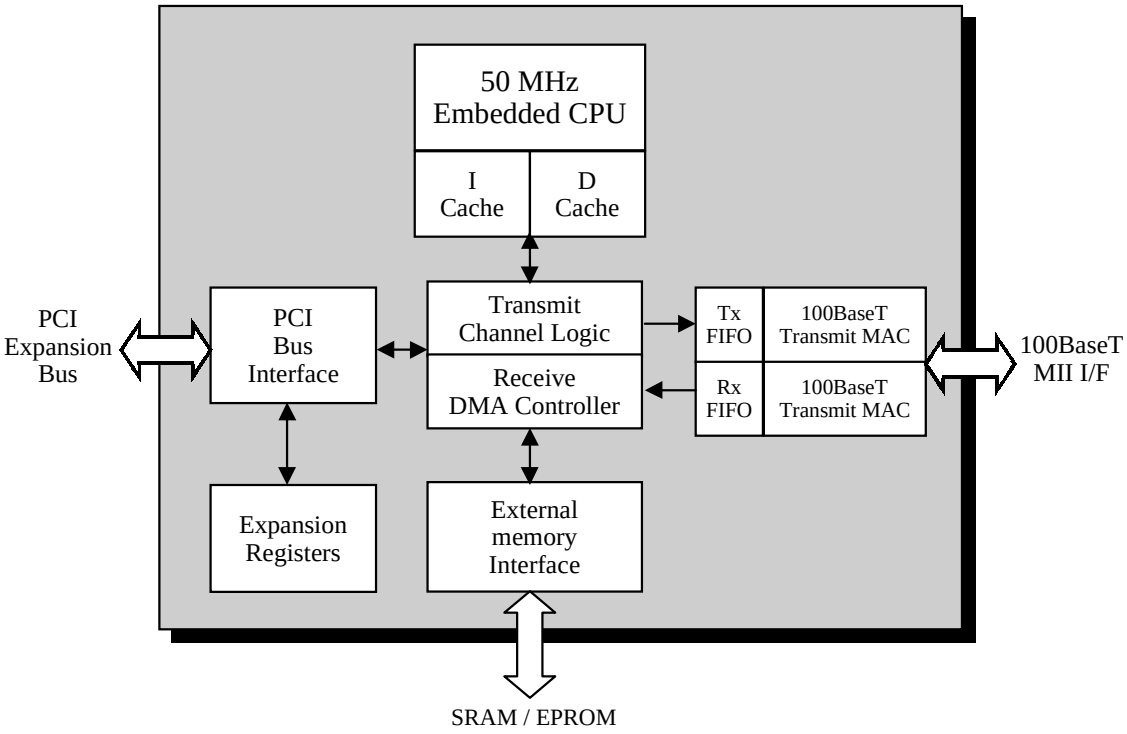
- Interfaces directly to industry-standard 100 Mbit/s transceivers with no glue logic via the built-in Medium Independent Interface (MII) port with full support for the autonegotiation function implemented by the PHY devices.

- Fully static CMOS operation at 50 MHz clock rates.

- 3.3 Volt core, 5 Volt compatible I/O

- 208 pin PQFP package.

BLOCK DIAGRAM



DESCRIPTION

The PM3351 is a low-cost, highly integrated stand-alone single-chip switching device for 10/100 Mbit/s Ethernet (IEEE 802.3u, IEEE 802.12) switching and bridging applications. The device supports all processing required for switching Ethernet packets between the on-chip Medium Independent Interface (MII) port and the built-in 1 Gbit/s expansion port, to which other PM3351 (ELAN 1x100) or PM3350 (ELAN 8x10) devices may be attached.

The PM3351 is directly compatible with the PM3350, 8-port 10Mbit/s Ethernet switch chip. The PM3351 can be used with the PM3350 to create non-blocking switches of the configurations shown in the table below, with each 100 Mbit/s port configured for full-duplex and each 10 Mbit/s port configured for half-duplex

Switch Configuration	# PM3350 Chips	# PM3351 Chips
4 x 100 Mbit/s	0	4
3 x 100 Mbit/s + 16 x 10Mbit/s	2	3
2 x 100 Mbit/s + 40 x 10 Mbit/s	5	2
1 x 100 Mbit/s + 56 x 10 Mbit/s	7	1
0 x 100 Mbit/s + 64 x 10 Mbit/s	8	0

A switch built using the PM3351 can be expanded to use up to 7 additional devices on the PCI expansion bus; the limitation to 8 devices is a result of using dedicated internal counters for implementing the ELAN switch protocol over the expansion bus. For details on the expected expansion bus data traffic requirements for different combinations of PM3350 and PM3351 chips please refer to section **Expansion Bus Data Transfer Rates**.

All of the initialization, switching, interfacing, management and statistics gathering functions are performed by the PM3351, minimizing the size and cost of a switching hub with one or more 100 Mbit/s ports.

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 1x100 also collects per-port and per-host

RMON statistics at wire rates on all ports. The PM3351 chip contains all the required elements of a high-performance Ethernet switch: an MII interface for connection to physical-layer transceivers, MAC-layer processing logic, buffer FIFOs, a high-speed DMA engine for fast frame transfers, a local memory interface for up to 4 Mbytes 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 for each 100 Mbit/s switch port are an MII compliant transceiver (supports 100BaseTX/FX, 100BaseT4,100BaseT2, and any future 802.3-compliant 100Mbit MII PHYs), passive line interface devices, a bank of external memory and a system clock. The amount of external memory may range up to 4 Mbytes, depending on the amount of frame buffering required and the number of MAC addresses to be supported, and may be implemented using standard asynchronous SRAM devices with 15 nsec access times. Switch configuration information is provided to the PM3351 using a single EPROM or EEPROM; only one EPROM is required in a multiple PM3351/PM3350 system.

The ELAN 1x100 device is implemented in a high-density, low-voltage CMOS technology for low cost and high performance. It is available in a 208-pin Quad Flat Pack, and is ideally suited for compact, low-cost desktop, workgroup and departmental Ethernet switching applications.

DEVICE DATA

Introduction

The ELAN 1x100 (PM3351) offers a complete system-level solution, integrating all required elements (except frame/address memory and transceiver logic) in a single high-density VLSI chip. It is a true single-chip switch; all the required functions, including management, RMON-level statistics collection, spanning tree support and self-configuration, are performed by the ELAN 1x100 without need for external CPUs or logic. In addition, the functions required for expandability are also integrated into the device.

The ELAN 1x100 is built around a RISC CPU based switching processor core which is closely 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 frame routing functions, and can switch packets arriving simultaneously from the single100BaseT port 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. It implements the full IEEE 802.1d spanning-tree transparent bridging protocol, which allows the ELAN 1x100 to act as an expandable learning bridge, performing learning, filtering and redirection at full speed. RMON statistics collection, plus a messaging system to support master/slave communications in a multi-device switch, are also implemented by the Switch Processor. When additional switch devices are connected to the ELAN 1x100 expansion port, the Switch Processors in all PM3350 and PM3351 ELAN chips intercommunicate to transfer frames to each other and also transparently support a distributed SNMP/RMON MIB.

Multi-channel DMA Processor

The on-chip DMA Controller contains four independent and concurrently operating channels: one for receiving frames over the 100BaseT port; another for transmitting frames over the same port; and two that are dedicated to the expansion port. The DMA Controller operates under the control of the Switch Processor to transfer packets and data at high speed between the 100BaseT port, the local memory and the expansion port. It also computes 32-bit IEEE Frame Check Sequence (FCS) CRC remainders over the transferred packets, allowing the Switch Processor to filter packets with errors and generate CRCs for transmitted packets as required. Control logic is provided to support full and half-duplex operation, as well as the handling of management traffic.

Ethernet/IEEE 802.3 MAC Interface

One Ethernet/IEEE 802.3u MAC-layer interface based on the Media Independent Interface (MII) is built into the ELAN 1x100 chip. It connects directly to the external 100BaseT compatible PHY devices via the industry-standard MII interface, and performs all of the MAC-layer processing tasks required for CSMA/CD networks. Both full-duplex and half-duplex modes of operation are supported at 10 and 100 Mbit/s data rates. The MAC interface includes independent receive and transmit FIFO buffers to support sustained full wire rate operation.

Configuration and initialization of the attached 100Mbit/s PHY devices can be performed using the MII management interface. This is accomplished by using the MII pins MDIO and MDCLK to read, write and poll the management registers built into an MII-compliant PHY device. This allows the PM3351 to support auto-negotiation, link status, and other management operations on the attached PHY device(s), including the Next Page functions. Serial management functions are implemented by the Switch Processor for maximum flexibility and "standards-proofing".

Expansion Port

A 32-bit parity-checked PCI-based expansion port is provided to allow the ELAN 1x100 to communicate transparently with other PCI bus devices to implement switches that have multiple 100Mbit/s and 10Mbit/s ports. The expansion port supports a maximum throughput of over 1 Gbit/s, and requires only a single PAL or similar device serving as a bus arbiter. A common protocol is used for inter-chip communication between the ELAN 1x100 (PM3351) and ELAN 8x10 (PM3350). Packets received on an ELAN 1x100 MAC port that are destined for an external ELAN switch chip 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 frame is first transferred to all of the external ELAN switch chips, after which it is transmitted out the required destination ports without any further use of the expansion bus. (In the case of an ELAN 1x100, only one port is present in each destination device, and hence only one level of replication takes place.) In addition, ELAN switch chips interconnected via the PCI bus exchange information to maintain the distributed MIB.

The expansion port is compatible with the industry-standard Peripheral Component Interconnect (PCI) specification version 2.1, which allows the ELAN 1x100 chip to be directly interfaced to any host computer supporting the PCI bus. The host CPU can then communicate with the ELAN 1x100 and control its functions, greatly expanding the range of potential applications. The maximum PCI clock of 40 MHz is supported.

Local Memory Controller

The 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 1x100 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.[1] Independent, software programmable access times may be set for each bank, allowing a glueless interface to a mix of SRAM, , EEPROM, and EPROM 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 MByte-per-second memory port bandwidth among them.

The ELAN 1x100 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

[1] When accessing SRAM and EEPROM devices, however, the actual limit is 1 Mbyte per bank, as the upper bits of the internal address bus are not brought out to physical pins. The entire 4 Mbytes per bank is only accessible when using DRAM memory types.

address, configuration options, etc.) may be placed in this EPROM or EEPROM, and will be loaded automatically by the ELAN 1x100.[1]

Clocking and Test

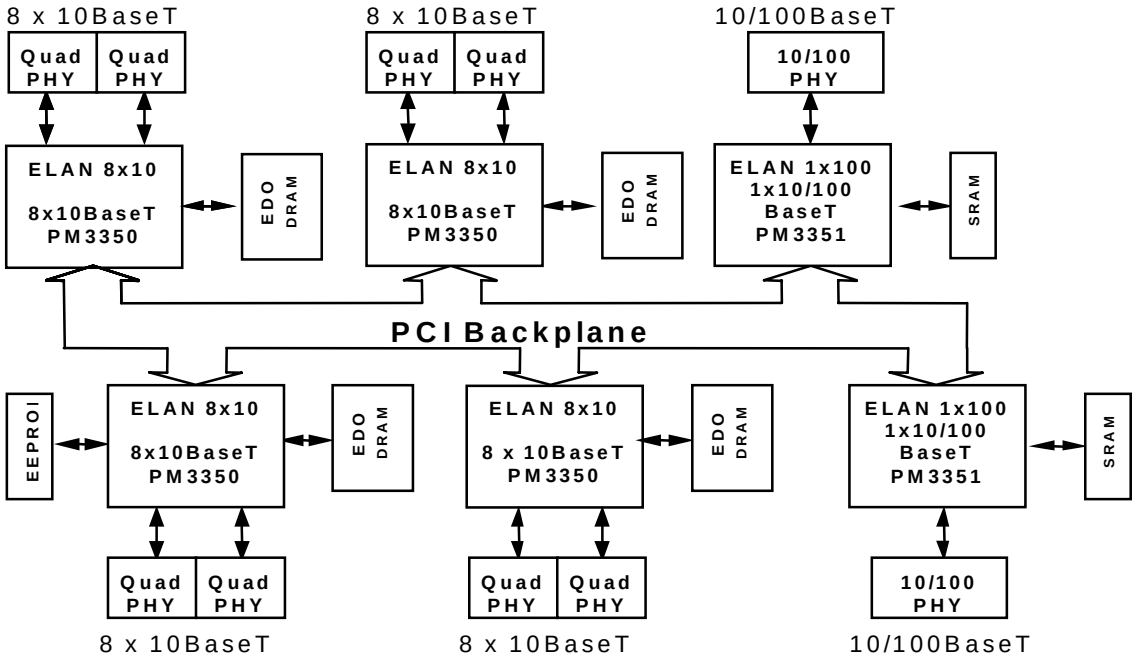
The PM3351 is implemented in fully-static CMOS technology, and can operate at any device clock frequency between 45 and 50 MHz (25 to 40 MHz for the expansion port bus). The Switch Processor performs a comprehensive power on self test (POST), and can report failure conditions and device status in a variety of ways (an 8-bit LED interface register connected to local memory, writing to a host over the PCI bus, writing to a serial port interface connected to local memory, etc).

[1] The ELAN 1x100 follows the same configuration process as the ELAN 8x10 chip.

TYPICAL SYSTEM APPLICATION

Low-cost 10/100 Mbit/s Switching Hub

The PM3351 chip can act as a high-speed server or backbone port in low-cost, compact Ethernet switching hub applications. Such a hub can be created from one or more 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 DRAM for each PM3350, 15 ns SRAM for each PM3351) for holding frame buffers and switching tables, a single 32k x 8-bit EEPROM device for configuration information, two LXT944 10BaseT interface adapters per PM3350 chip, one LXT 970 100 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 stacking connectors allow multiple switch assemblies to be seamlessly stacked.



Expandable 10/100 Mbit/s Ethernet Switch

PRIMARY FEATURES AND BENEFITS

As an Ethernet switching/bridging solution, the PM3351 offers a number of benefits in a low-cost switching hub:

Wire-speed frame switching.

Each PM3351 chip performs all of the functions required to implement a 10/100 Mbit/s full/half duplex switch port at wire rates (ranging from 148,808 frames/sec for a frame size of 64 bytes to 8130 frames/sec for a frame size of 1518 bytes). No additional logic, microprocessors, etc. are necessary. In addition, the low cost and compact size permitted by the single-chip PM3351 solution permits high-speed server or backbone ports to be added to even entry-level switching hubs very simply.

Combined Input- and Output-buffered switch

The ELAN 1x100 implements a hybrid input-buffered/output-queued switching algorithm which minimizes the possibility of frame loss, allows buffers to be allocated on a demand basis, and permits limits to be established to prevent any one memory consumer from acquiring all system buffer memory. Frame buffer storage is allocated within the external memory by the ELAN 1x100 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 PM3351 chips interconnect with no external glue logic (beyond a simple PCI arbiter device), allowing a family of scalable switches to be built without redesign. When used in conjunction with the PM3350 single-chip 8-port 10 Mbit/s Ethernet switch, a 10/100 Mbit/s switching hub can be realized at low cost. The 1 Gbit/s expansion port bandwidth ensures that network capacity grows linearly as more chips from the ELAN switch family are added.

Advanced switching features

The ELAN 1x100 implements per-frame 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. Two different methods of flow control are supported: backpressure for a half-duplex link and 802.3x Pause MAC Control frames for a full-duplex link. Flow control is a user-selectable feature, with the thresholds and limits capable of being user-configured in order to minimize frame loss in heavily loaded networks. Backpressure is

performed by jamming (colliding with) incoming packets on the 100Mb port when the port has no free receive buffers.

Spanning tree bridging capabilities

The ELAN 1x100 is capable of supporting 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, as the actual agent may be supported on the Switch Processor on one of the PM3350 chips.

Management and monitoring support

The ELAN 1x100 (PM3351) maintains and collects RMON port and host statistics for 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). When multiple ELAN 1x100 and ELAN 8x10 chips are present in a system, they may be configured to intercommunicate and create a distributed MIB in a transparent manner.[1]

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 1x100 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.

An optional on-chip watchdog timer is provided by the ERST* output of the ELAN 1x100. The ERST* output of the device can be tied directly to the RST* input to provide an optional system-wide watchdog reset. This facility permits the ELAN 1x100 to force an automatic system restart whenever a fatal error is encountered during operation.

Autoconfiguration via Local PROM/EEPROM

The ELAN 1x100 automatically self-configures 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 1x100 will automatically locate the EPROM or EEPROM and load the configuration parameters from it. The ELAN 1x100 also contains hardware that enables it to write to standard 3.3-volt EEPROM devices, thus permitting configuration information to be changed dynamically.

[1] The ELAN 8x10, if present in the system, can directly implement an optional on-chip SNMP agent on top of an integral TCP/UDP/IP protocol stack, supporting SNMP and the RFC1493 bridge MIB, and supplying SNMP access to the statistics gathered by the ELAN 1x100 (as well as the other devices in the system). Alternatively, the system vendor may elect to disable the SNMP agent and access all chip statistics and configuration variables directly from the expansion port.

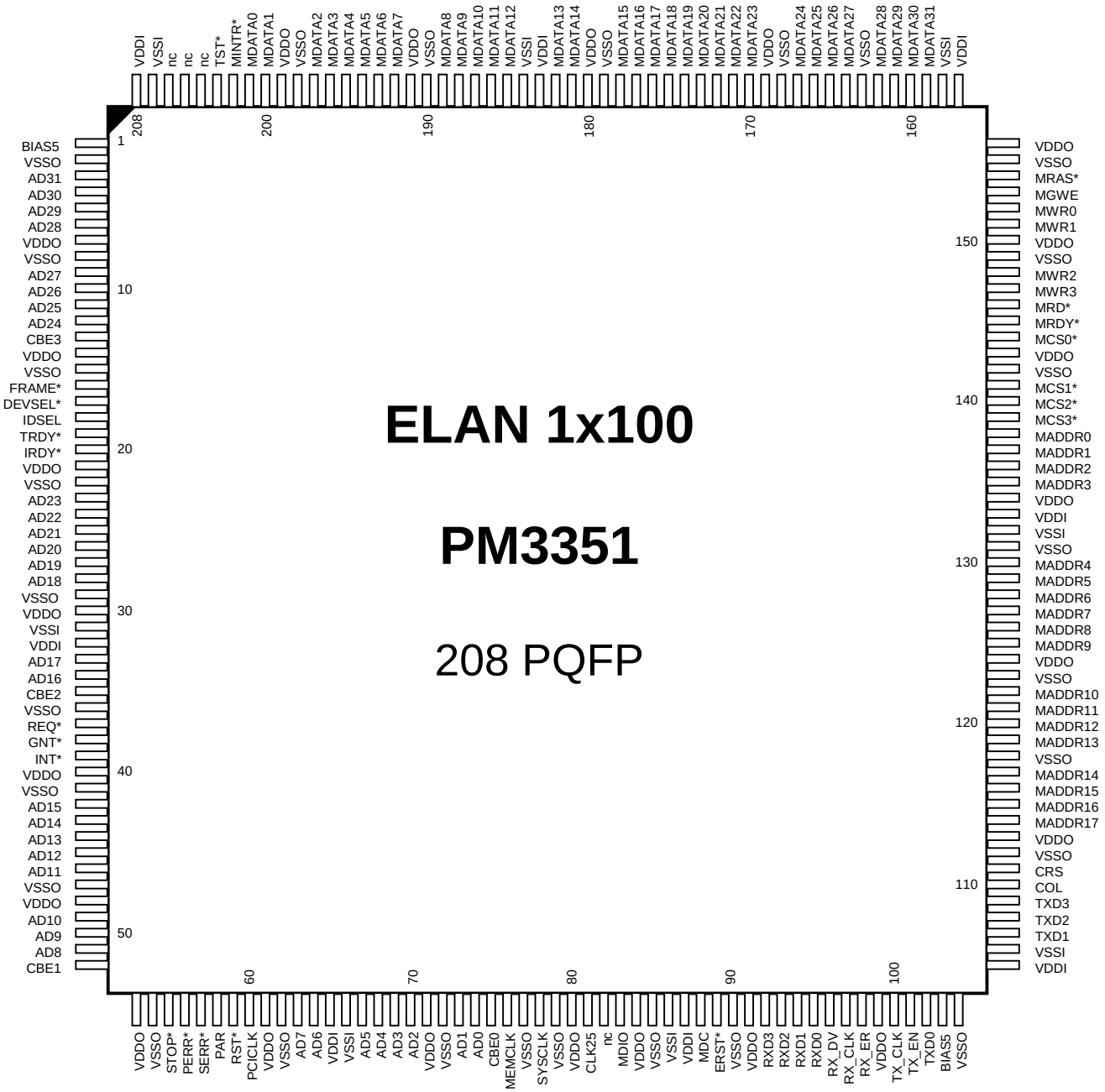
In a system with multiple ELAN devices, the master device (whether ELAN 1x100 or ELAN 8x10) can load its configuration information and then configure the other ELAN devices over the expansion bus, allowing configuration of the entire system with one EPROM.

-

PIN DIAGRAM

The PM3351 is packaged in a 208-pin PQFP, with 135 signal pins, 36 VSS (gnd) pins, 31 VDD (3.3V) pins, and two 5V supply pins. The VSS and VDD pins are split between internal (core) & i/o buffer pins (8 VSSI, 8 VDDI, 28 VSSO, & 23 VDDO pins). All pins must be connected properly. Attention should be given to the 4 no-connect pins.

208-Pin Quad Flat Pack Pin Diagram



PIN DESCRIPTION

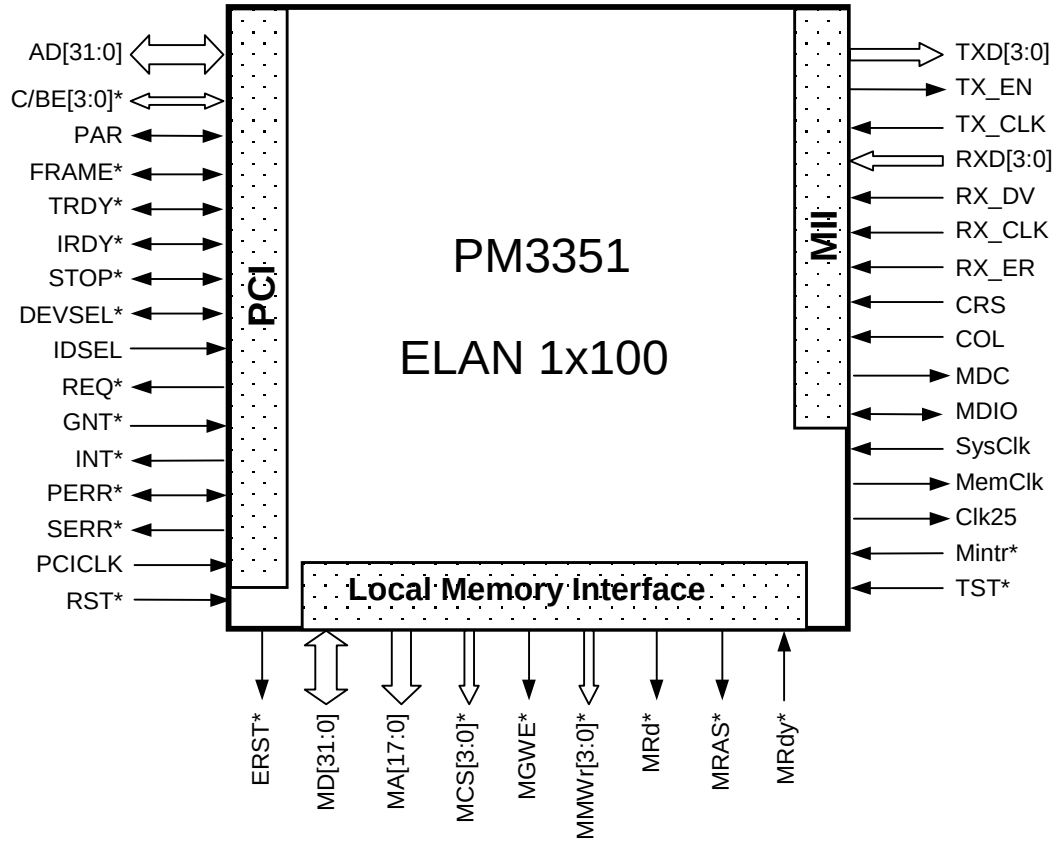
Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	BIAS5V	53	VDDO	105	VDDI	157	VDDI
2	VSSO	54	VSSO	106	VSSI	158	VSSI
3	AD31	55	STOP*	107	TXD1	159	MDATA31
4	AD30	56	PERR*	108	TXD2	160	MDATA30
5	AD29	57	SERR*	109	TXD3	161	MDATA29
6	AD28	58	PAR	110	COL	162	MDATA28
7	VDDO	59	RST*	111	CRS	163	VSSO
8	VSSO	60	PCICLK	112	VSSO	164	MDATA27
9	AD27	61	VDDO	113	VDDO	165	MDATA26
10	AD26	62	VSSO	114	MADDR17	166	MDATA25
11	AD25	63	AD7	115	MADDR16	167	MDATA24
12	AD24	64	AD6	116	MADDR15	168	VSSO
13	CBE3	65	VDDI	117	MADDR14	169	VDDO
14	VDDO	66	VSSI	118	VSSO	170	MDATA23
15	VSSO	67	AD5	119	MADDR13	171	MDATA22
16	FRAME*	68	AD4	120	MADDR12	172	MDATA21
17	DEVSEL*	69	AD3	121	MADDR11	173	MDATA20
18	IDSEL	70	AD2	122	MADDR10	174	MDATA19
19	TRDY*	71	VDDO	123	VSSO	175	MDATA18
20	IRDY*	72	VSSO	124	VDDO	176	MDATA17
21	VDDO	73	AD1	125	MADDR9	177	MDATA16
22	VSSO	74	AD0	126	MADDR8	178	MDATA15
23	AD23	75	CBE0	127	MADDR7	179	VSSO
24	AD22	76	MEMCLK	128	MADDR6	180	VDDO
25	AD21	77	VSSO	129	MADDR5	181	MDATA14
26	AD20	78	SYSCLK	130	MADDR4	182	MDATA13
27	AD19	79	VSSO	131	VSSO	183	VDDI
28	AD18	80	VDDO	132	VSSI	184	VSSI

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
29	VSSO	81	CLK25	133	VDDI	185	MDATA12
30	VDDO	82	nc	134	VDDO	186	MDATA11
31	VSSI	83	MDIO	135	MADDR3	187	MDATA10
32	VDDI	84	VDDO	136	MADDR2	188	MDATA9
33	AD17	85	VSSO	137	MADDR1	189	MDATA8
34	AD16	86	VSSI	138	MADDR0	190	VSSO
35	CBE2	87	VDDI	139	MCS3*	191	VDDO
36	VSSO	88	MDC	140	MCS2*	192	MDATA7
37	REQ*	89	ERST*	141	MCS1*	193	MDATA6
38	GNT*	90	VSSO	142	VSSO	194	MDATA5
39	INT*	91	VDDO	143	VDDO	195	MDATA4
40	VDDO	92	RXD3	144	MCS0*	196	MDATA3
41	VSSO	93	RXD2	145	MRDY*	197	MDATA2
42	AD15	94	RXD1	146	MRD*	198	VSSO
43	AD14	95	RXD0	147	MWR3*	199	VDDO
44	AD13	96	RX_DV	148	MWR2*	200	MDATA1
45	AD12	97	RX_CLK	149	VSSO	201	MDATA0
46	AD11	98	RX_ER	150	VDDO	202	MINTR*
47	VSSO	99	VDDO	151	MWR1*	203	TST*
48	VDDO	100	TX_CLK	152	MWR0*	204	nc
49	AD10	101	TX_EN	153	MGWE*	205	nc
50	AD9	102	TXD0	154	MRAS*	206	nc
51	AD8	103	BIAS5V	155	VSSO	207	VSSI
52	CBE1	104	VSSO	156	VDDO	208	VDDI

- All VDD and VSS lines **must** be connected to the 3.3V supply and ground respectively.
- Standard decoupling practices should be followed for proper device operation.
- The 4 pins marked as "nc" should be left as no-connects. They are intended solely for factory use. Do not connect pins marked "nc" directly to either VDD or VSS. If need be, the pins can be connected to either VDD or VSS through a terminating resistor of value 1 k-ohm or greater.

Functional Grouping

The following diagram shows the functional grouping of the PM3351 signal pins.



Pin description nomenclature:

- **I** input only
- **O** output only
- **I/O** bidirectional pin
- **OD** output only, open drain (requires external pull-up resistor to VDD).

The recommended pull-up resistance value for the open drain outputs is 2.7 k-ohm (this is the recommended pull-up value suggested for PCI bus signals in the 5V signalling environment).

PCI Expansion Bus Interface

Signal Name	Size	Type	Description
AD[31:0]	32	I/O	Multiplexed PCI address/data bus, used by the PCI host or the PM3351 to transfer addresses or data.
CBE[3:0]	3	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 PM3351 is acting as a target) or driven (when the PM3351 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 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 deasserted; otherwise, it indicates (in conjunction with STOP* and TRDY*) a target-abort.
IDSEL	1	I	Device identification (slot) select. Assertion of IDSEL signals the PM3351 that it is being selected for a configuration space access.
REQ*	1	O	Bus request (to bus arbiter), asserted by the PM3351 to request control of the PCI bus.
GNT*	1	I	Bus grant (from bus arbiter); this indicates to the PM3351 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 deasserted simultaneously).
INT*	1	OD	Interrupt request. This pin signals an interrupt request to the PCI host. 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 PM3351 as a bus slave, or sampled by the PM3351 as a bus master, to indicate a parity error on the AD[31:0] and CBE[3:0] lines.
SERR*	1	OD	System error, used by the PM3351 to indicate to the PCI central resource that there was a parity error on the AD[31:0] and CBE[3:0] lines during an address phase.
PCICLK	1	I	PCI bus clock; supplies the PCI bus clock signal to the PM3351.

RST*	1	I	PCI bus reset (system reset). Performs a hardware reset of the ELAN 1x100 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.

MII Interface Pins

Signal Name	Size	Type	Description
TXD[3:0]	4	O	MII transmit data. TXD[3:0] is the nibble-wide MII transmit data bus. TXD[3:0] transitions synchronously to the rising edge of TX_CLK. If TX_EN is asserted then the TXD[3:0] bus has valid data which is to be accepted for transmission by the PHY device.
TX_EN	1	O	MII transmit enable. Asserted by the PM3351 to indicate to the PHY device that the TXD[3:0] bus has valid data. TX_EN is asserted with the first nibble of preamble and remains continuously asserted throughout the frame. TX_EN is negated prior to the first TX_CLK following the final nibble of the frame. TX_EN is synchronous with respect to TX_CLK.
TX_CLK	1	I	MII transmit clock. TX_CLK is a continuous clock that provides the timing reference for the TX_EN and TXD[3:0] signals output from the PM3351. TX_CLK is a nibble rate clock; an MII transceiver operating at 100Mbit/s must provide a TX_CLK frequency of 25 Mhz. For improved noise immunity this input buffer uses a Schmitt trigger.
RX_DV	1	I	MII receive data valid. RX_DV is an input that indicates that the PHY is presenting recovered and decoded nibbles on the RXD[3:0] pins. In order for a received frame to be accepted by the PM3351, RX_DV must be asserted prior to or coincident to the first nibble of the Start of Frame Delimiter being driven on RXD[3:0]; it must remain continuously asserted until after the rising edge of RX_CLK when the last nibble fo the CRC is driven on RXD[3:0]. RX_DV is synchronous with respect to RX_CLK.
RXD[3:0]	4	I	MII receive data bus. RXD[3:0] is the nibble-wide MII receive data bus. RXD[3:0] transitions synchronously to the rising edge of RX_CLK. The PM3351 samples RXD[3:0] on every rising edge of RX_CLK that RX_DV is asserted. RXD[3:0] is ignored if RX_DV is deasserted.
RX_CLK	1	I	MII receive clock. RX_CLK is a continuous clock that provides the timing reference for the RX_DV, RX_ER, and RXD[3:0] signals input to the PM3351. RX_CLK is a nibble rate clock; an MII transceiver operating at 100Mbit/s must provide an RX_CLK frequency of 25 MHz during frame reception. For improved noise immunity this input buffer uses a Schmitt trigger.
RX_ER	1	I	MII receive error. RX_ER is driven by the PHY for one or more RX_CLK periods to indicate to the PM3351 that an error has occurred in the frame being received. The PM3351 samples RX_ER on the rising edge of RX_CLK only if RX_DV is asserted; all special code groups generated on RXD while RX_DV is deasserted are ignored. RX_ER is synchronous with respect to RX_CLK.
CRS	1	I	MII carrier sense. CRS is an input that indicates that the physical media is non-idle, either because of transmit or receive activity. In full-duplex mode CRS is ignored. The PHY is not required to have CRS transition synchronously to either TX_CLK or RX_CLK. The PM3351 samples CRS only on the rising edge of **TX_CLK**. It is used as the carrierSense variable in the MAC Deference process (see 802.3 spec clauses 4.2.8 and 22.1.3.2).

COL	8	I	MII collision detect. COL is an input that indicates that a collision has occurred on the physical media. In full-duplex mode COL is ignored. The PHY is not required to have COL transition synchronously to either TX_CLK or RX_CLK. The PM3351 only samples COL on the rising edge of **TX_CLK.**
MDIO	1	I/O	MII management data input/output.MDIO is the bidirectional MII management port data pin. MDIO is driven synchronously to MDCLK and is sampled on the rising edge of MDCLK. When a management frame is not being transferred, MDIO is not driven. In order to implement the PHY detection feature via the MII mechanical interface the MDIO pin should be externally pulled down to VSS through a 2.0-kohm +/-5% resistor.
MDC	1	O	MII management data clock. MDC is an aperiodic signal that is used as a timing reference for the MDIO pin. MDC is continuously driven by the PM3351; it is asserted only during management frame activity. MDC uses the SYSCLK input as a timing reference.

Local Memory Interface

Although the on-chip local memory interface is designed to operate with various memory types the memories intended to be used with the PM3351 are -15ns SRAM and -150ns PROM/EPROM/EEPROM. EDO DRAM is supported for management and custom applications, but is not intended to be used for standard switching.

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 PM3351 during local memory writes, and the data sent back to the PM3351 by the memory devices during local memory reads. In addition, configuration information is latched from the MDATA[31:0] lines during ELAN 1x100 reset and loaded into an internal configuration register; either pullup-pulldown resistors or tristate 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[17:0]	18	O	Memory address bus; supplies a word-aligned address to the external memory devices (i.e., address bits 19 through 2 of the 24-bit byte address generated by the internal PM3351 logic), and thus select a single 32-bit word to be read or written. Up to 1 MB of memory may be directly addressed in each bank using these address lines.
MCS[3:0]*	4	O	Memory bank chip selects. The MCS[3:0]* outputs select one of four memory banks (each bank has maximum depth of 4 megabyte). 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 PM3351 logic, and are synchronous to MEMCLK.
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. NOTE- can be left as a no-connect output if not using DRAM.

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.
MWR[3:0]*	4	O	Memory write enables, used by the PM3351 to enable the data presented on individual byte lanes of MDATA[31:0] to be individually written to memory. MWR[0]* corresponds to MDATA[7:0], MWR[1]* corresponds to MDATA[15:8], and so on. The MWR[3:0]* outputs should be connected to the appropriate byte write enables.
MGWE*	1	O	Gobal 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. NOTE- can be left as a no-connect output if not using DRAM
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 PM3351 to insert wait states into memory accesses. If the MRDY* line is deasserted, the PM3351 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 PM3351 when performing an SRAM-type access; it is ignored for all other memory types. This feature is not tested as part of the fuctional test program of the device. Therefore, MRDY* must be tied low (to logic 0) to ensure correct operation.
MINTR*	1	I	Local interrupt input. The MINTR* may be used to provide an interrupt input to the ELAN 1x100 in special applications. If the MINTR* input is not used it should be tied HIGH. For improved noise immunity this input buffer uses a Schmitt trigger.

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. For improved noise immunity this input buffer uses a Schmitt trigger.
MEMCLK	1	O	50 MHz clock derived from SYSCLK; supplies the re-timed 50 MHz clock (input on the SYSCLK pin) to external devices.
CLK25	1	O	25 MHz clock output. The 50 MHz input clock is internally divided by two and output as a symmetrical 25 MHz clock on the CLK25 output; this clock may be used as a clock reference input to an external PHY device.

Miscellanous Inputs and Outputs

Signal Name	Size	Type	Description
BIAS5V	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 BIAS5V 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 BIAS5V 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	External reset output. The ERST* pin is driven low by the ELAN 1x100 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 1x100 senses a condition requiring a system hardware reset. It is an open-drain output, and should be pulled up using a 2.7k-ohm 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 1x100 must not be allowed to reset the host CPU.)

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*, TST*, GNT*, and RST* have internal pull-up resistors.
4. The VSSI and VSSO ground pins are not internally connected together. Failure to connect these pins externally may cause malfunction or damage the part.
5. The VDDI and VDDO power pins are not internally connected together. Failure to connect these pins externally may cause malfunction or damage the part.

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 +7.0	Vdc
BIAS5V pin voltage with respect to Vss	Vbias5V	Minimum: VDD – 0.3V Maximum: 5.5V	Vdc
Input Voltage	Vin	V_{bias5v} +0.3	Vdc
Input Current	Iin	+/-10	mAdc
Static Discharge Voltage		±1000	V
Latch-Up Current		±80	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
BIAS5V Voltage	Vbias5v	+4.75	+5.0	+5.25	Vdc
Operating Ambient Temp.	Ta	0		+70	°C

NOTE: the PM3351 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 208 pin PQFP package 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 commercial temperature range; however, if operating over the industrial temperature range (which has a maximum ambient temperature of +85 °C) a minimum airflow of 100 linear feet per minute is required.

D.C. Characteristics

DC characteristics are specified over recommended operating conditions (T_A = 0°C to +70°C, V_{DD} = 3.3 V ±5%, V_{BIAS}= 5.0V ±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 (Vdd = min, I_{OH} = -2 mA, Note 2)	2.4	Vdd	2.4	Vdd	Vdc
Vol	Output Low Voltage (Vdd = 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, Note 4)	+100	+20			µA
Iihpu	Input High Current (Pull ups, V_{IH} = Vdd, Note 4)	-10	+10			µA
Iddop	Supply Current, Vdd = 3.47 MHz, Outputs Unloaded, SYSCLK = 50MHz		450			mA

Notes on D.C. Characteristics:

1. Negative currents flow into the device (sinking), positive currents flow out of the device (sourcing).
2. Output pin or bidirectional pin. Voh not measured on Open Drain outputs.
3. Input pin or bidirectional pin without internal pull-up resistor.
4. Input pin or bidirectional pin with internal pull-up resistor.

AC CHARACTERISTICS

AC characteristics are specified over recommended operating conditions (T_A = 0°C to +70°C, V_{DD} = 3.3 V ±5%, V_{BIAS}= 5.0V ±5%).

The PM3351 only supports a 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:

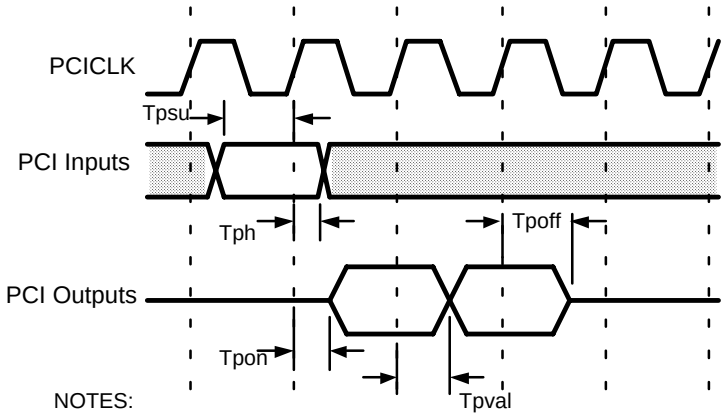
1. Output propagation delay time is the time in nanoseconds from the 1.4 Volt point of the reference signal to the 1.4 Volt point of the output.
2. Maximum and minimum output propagation delays are specified with a 50 pF load on the outputs, unless otherwise noted.
3. Output tristate delay is the time 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 shown as **Typical** in the following tables are tested for functionality under typical conditions. No guarantees are implied for maximum or minumum 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	1			nsec
Tpon	Minimum Float to active delay of all outputs from PCICLK rising		2		nsec
Tpoff	Maximum 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

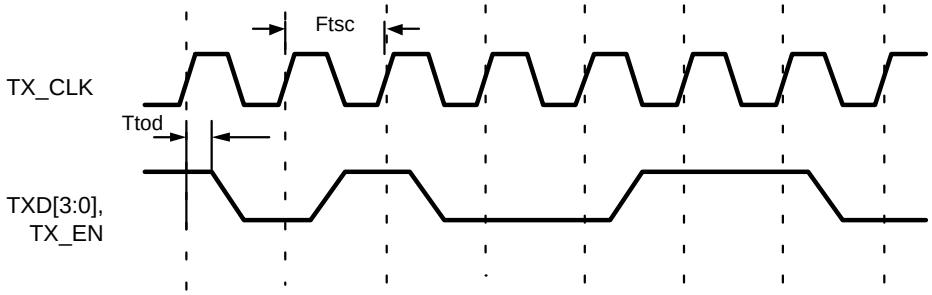


NOTES:
(1) PCI inputs are considered to be those signals that are driven by an external PCI device, while PCI outputs are signals that are driven by the PM3351. Note that many of the PCI signals are treated as outputs in some cycles and inputs in others.

PCI Bus Interface Timing

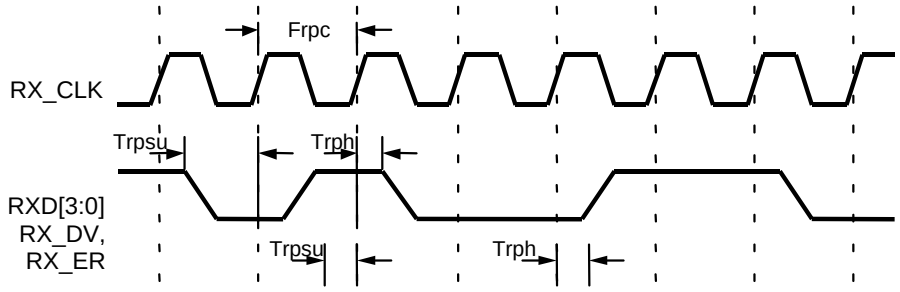
MII Interface

Parameter	Description	MIN	MAX	Units
Ftsc	TX_CLK frequency	0	25	MHz
Ttch/Ttcl	TX_CLK duty cycle (high/low)	35	65	Percent
Ttod	Output delay from TX_CLK rising to TXD[3:0], TX_EN	0	20	nsec



MII Interface Transmit Signals

Parameter	Description	MIN	MAX	Units
Frpc	RX_CLK frequency	0	25	MHz
Trch/Trcl	RX_CLK duty cycle (high/low)	35	65	Percent
Trpsu	RXD[3:0], RX_DV, RX_ER setup to RX_CLK	8		nsec
Trph	RXD[3:0], RX_DV, RX_ER hold to RX_CLK	8		nsec

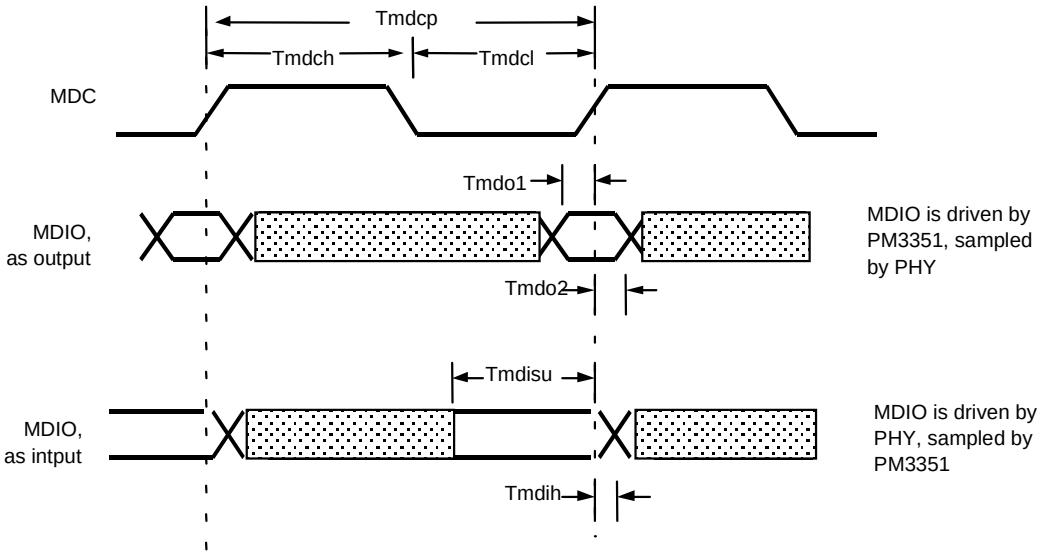


MII Interface Receive Signals

Parameter	Description	MIN	Typ	MAX	Units
Tmdch	Min MDC high pulse width (Cload = 390 pF) [1]		160		nsec
Tmdcl	Min MDC low pulse width (Cload = 390 pF) [1]		160		nsec
Tmdcp	Min MDC period (Cload = 390 pF) [1]		400		nsec
Tmdo1	Min MDIO (output delay) to posedge MDC[1] Cload = 470 pF. Measured from Vil,max (0.8V) or Vih,min(2.0V)		20		nsec
Tmdo2	Min MDIO (output delay) to posedge MDC[1] Cload = 470 pF. Measured from Vil,max (0.8V) or Vih,min(2.0V)		20		nsec
Tmdisu	Min MDIO (input) to MDC setup time		100		nsec
Tmdih	Min MDIO (input) to MDC hold time		0		nsec

Notes:

1. Tested at 50 pF in production test and derated.



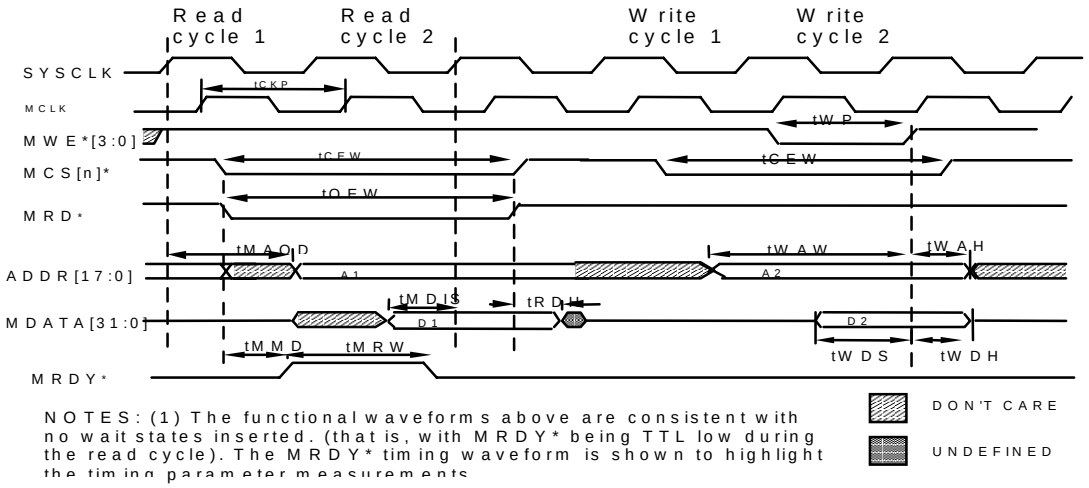
MII Management Data Signals

Memory Interface

15 nS SRAM AC Timing

Parameter	Description	MIN	Typ	MAX	Units
tCKP	MCLK period		20		ns
tMAOD	MADDR[17:0] delay from SYSCLK (read, first cycle)			21	ns
tCEW	MCS*[3:0] pulse width	35			ns
tOEW	MRD* pulse width	35			ns
tMDIS	MDATA[31:0] setup to SYSCLK (read, second cycle)	4			ns
tRDH	MDATA[31:0] hold from MRD* rise (read, second cycle) MDATA[31:0] hold from MCS*[3:0] rise (read, second cycle)	0			ns
tWAW	MADDR[17:0] setup to MWE*[3:0] rise	20			ns
tWAH	MADDR[17:0] hold from MWE*[3:0] rise	0			ns
tWP	MWE*[3:0] pulse width (write, second cycle)	15			ns
tWDS	MDATA[31:0] setup to MWE*[3:0] rise	8			ns
tWDH	MDATA[31:0] hold to MWE*[3:0] rise	0			ns

SRAM Timing: Read/Write cycle

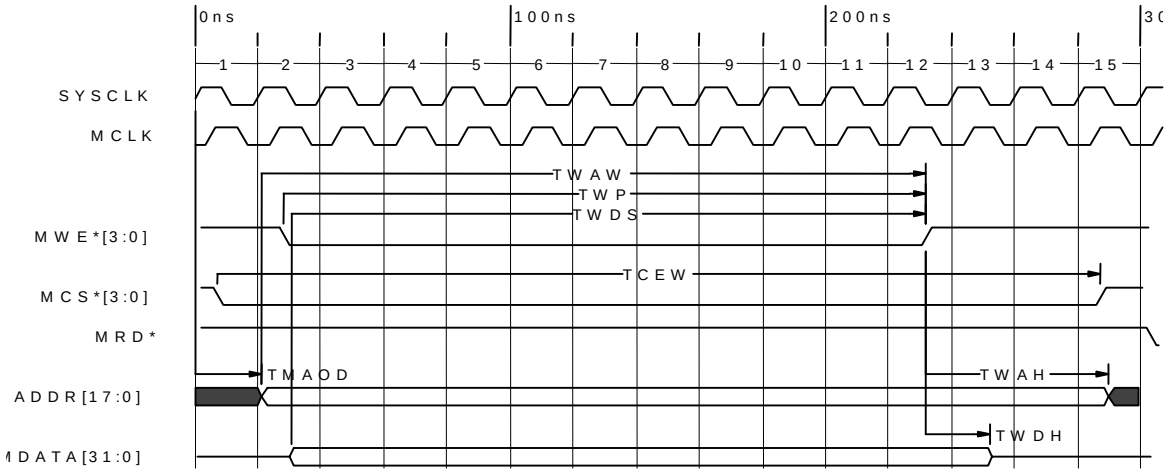


NOTES: (1) The functional waveforms above are consistent with no wait states inserted. (that is, with MRDY* being TTL low during the read cycle). The MRDY* timing waveform is shown to highlight the timing parameter measurements.

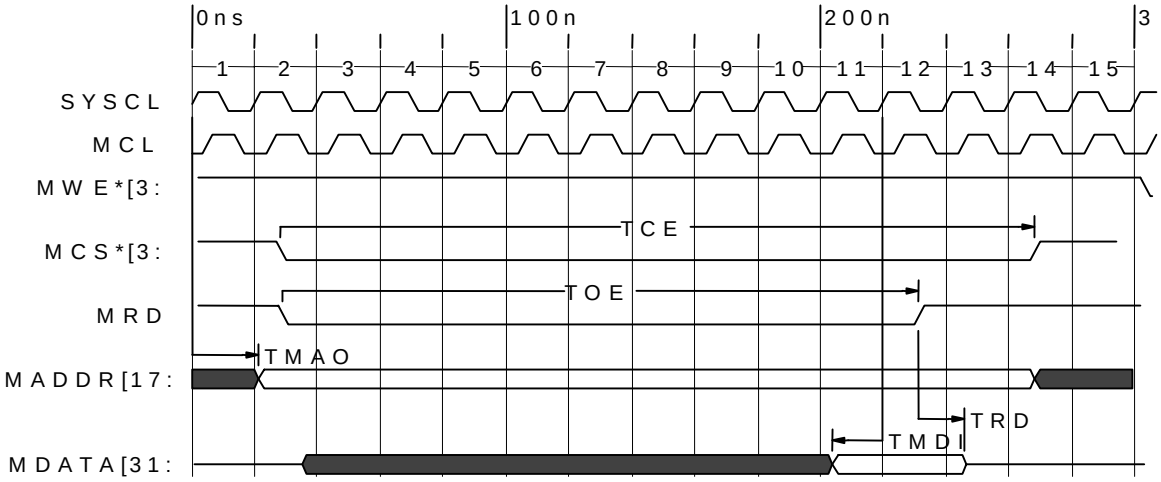
150 ns EEPROM/EPROM AC Timing

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

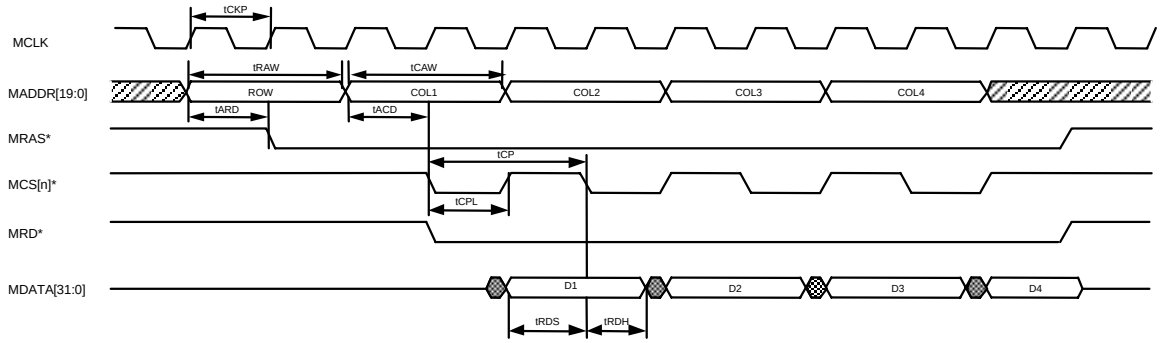


EDO DRAM (not used in standard switching applications)

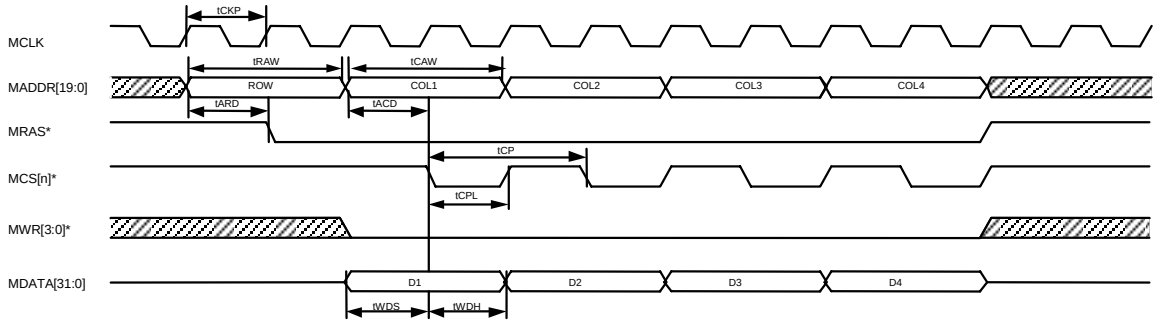
60 ns EDO DRAM AC Timing

Parameter	Description	MIN	Typ	MAX	Units
tCKP	MCLK period		20		nsec
tARD	Min Row address stable to MRAS* fall delay		15		nsec
tRAW	Min Row address width		35		nsec
tACD	Min Column address stable to CAS* fall delay		15		nsec
tCAW	Min Column address width		35		nsec
tCP	Min CAS* period		35		nsec
tCPL	Min CAS* low time		15		nsec
tWDS	Min Write data setup to CAS* fall		15		nsec
tWDH	Min Write data hold to CAS* fall		15		nsec
tRDS	Min Read data setup to CAS* fall		20		nsec
tRDH	Min Read data hold from CAS* fall		0		nsec
tREC	Min Read data HiZ to write data drive		15		nsec

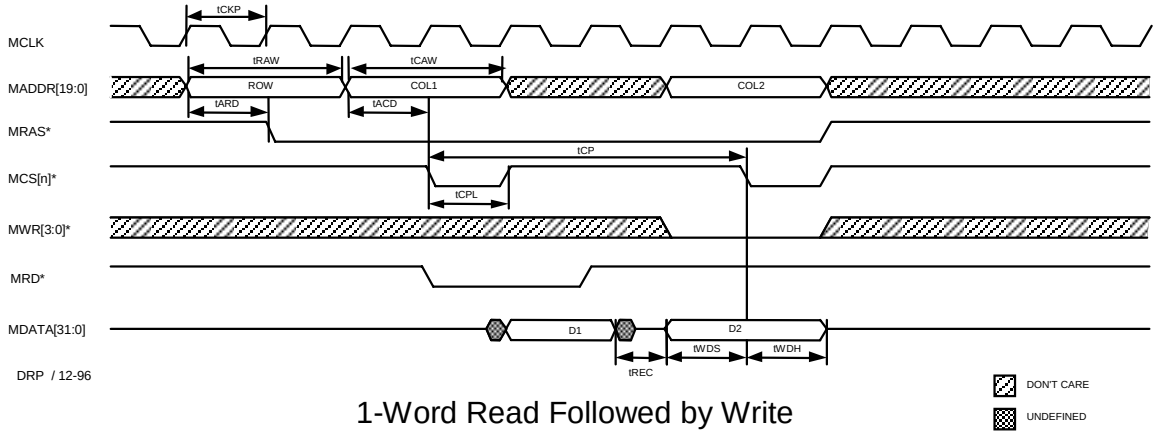
60 ns Single CAS EDO DRAM



4-Word Burst Read



4-Word Burst Write



1-Word Read Followed by Write

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]	5	50.0	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
Fck25	CLK25 frequency[3]		Fck/2		MHz
Trst	RST* active time after PCICLK and SYSCLK stable	10			usec

Notes:

1. The minimum clock frequency for production test is 5 MHz. In actual operation, the minimum SYSCLK frequency is 1 MHz if DRAM present (this is to refresh local DRAM memory); otherwise the minimum SYSCLK frequency is 0 MHz, reflecting completely static operation. 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. Minimum production test at 5.0 MHz
3. Guaranteed by design.

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 PM3351 ELAN 1x100 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 1x100 device, it is first necessary to discuss the operating environment that is intended to be built around it. A typical simple switching subsystem utilizing the ELAN 1x100 device consists of:

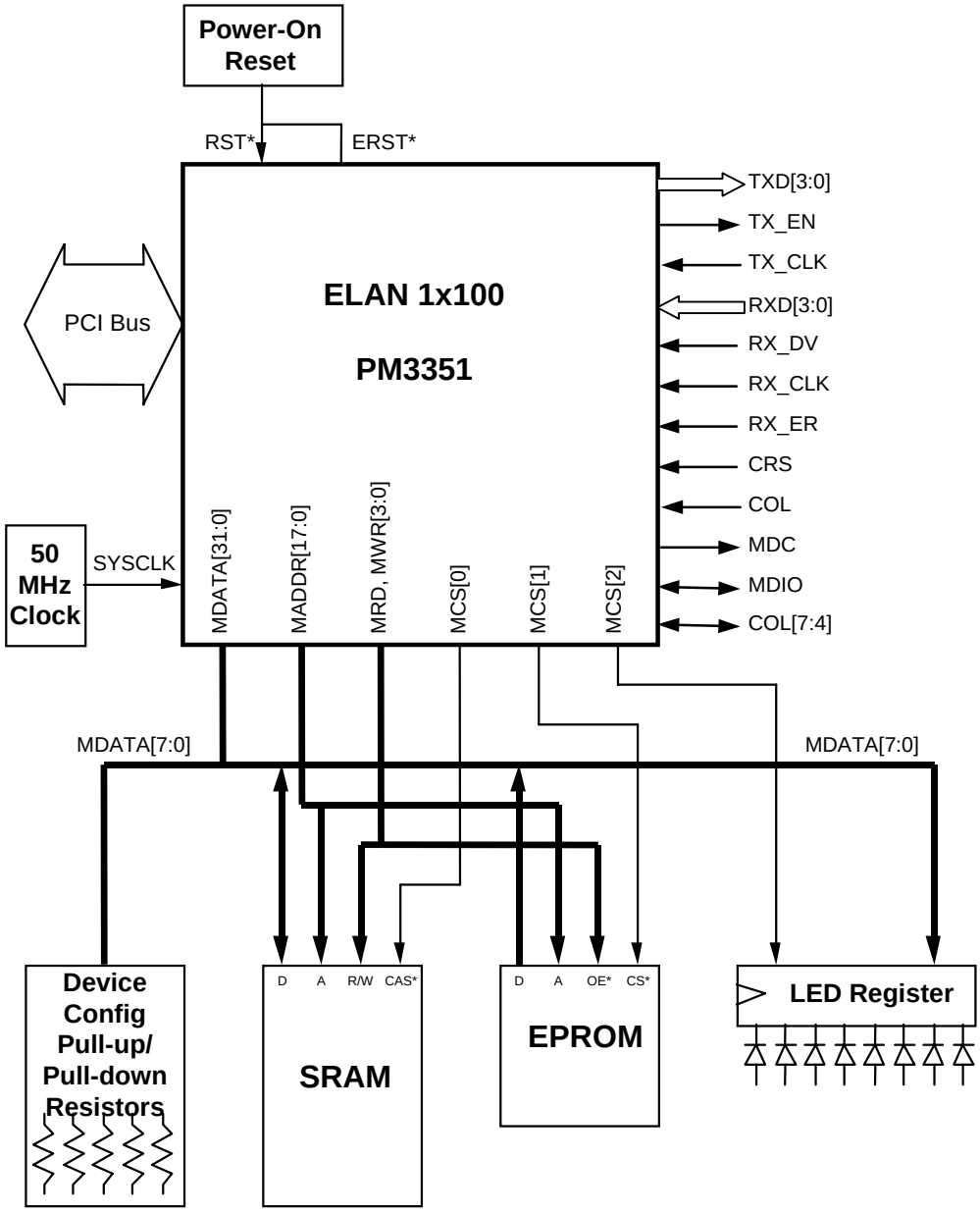
1. The ELAN 1x100 device itself. (Clearly, a complete system will include more than a single ELAN 1x100 device, but this is not shown in the diagram below.)
2. A set of SRAM devices that provide the operating memory for the ELAN 1x100. This memory should be mapped to the lowest bank of the ELAN 1x100 address space (i.e., addresses starting at 0x000000 hex). The amount of SRAM provided is a system-dependent parameter; a minimum system with about 150 kB of buffer space and around 1200 MAC addresses in the address table requires 512kB of SRAM. SRAM requirements for this and other system configurations may be computed using parameters supplied later in this document.
3. An EPROM containing the bootstrap image for the ELAN 1x100 (i.e., the operating firmware, operating parameters, and system initialization code) if this ELAN device is designated as the system master. This EPROM is mapped to the second lowest bank of the ELAN 1x100 memory address space (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 1x100, with the minimum requirement being 32 kB.
4. External transceiver devices that implement the PHY layer functions required for 100BaseT 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 1x100 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 desired, 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 1x100 data bus in order to drive device configuration information on to the data bus during system reset.

Note that the above subsystem description details a single ELAN 1x100 device in the system. This subsystem is expected to work in tandem with similar ELAN 1x100 or ELAN 8x10 subsystems to create an actual switch; however, the fundamental system operation does not change as additional ELAN compatible devices are added.

System Block Diagram

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



In the diagram above, the PHY device, the EPROM, the SRAM and the 50 MHz clock oscillator are all implemented 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 512 kilobyte of SRAM and 32 kbytes of EPROM are used. A typical implementation would use four 128k x 8-bit 15 ns SRAM, 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 SRAM and EPROM device types and speeds are defined by the setting of the pull-up / pull-down resistors on the memory data bus during 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 1x100 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, used to signal some external processor that the ELAN 1x100 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 2.7k 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 1x100. Eight LEDs may be connected to the outputs of this register to present the diagnostic status codes output by the ELAN 1x100 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.

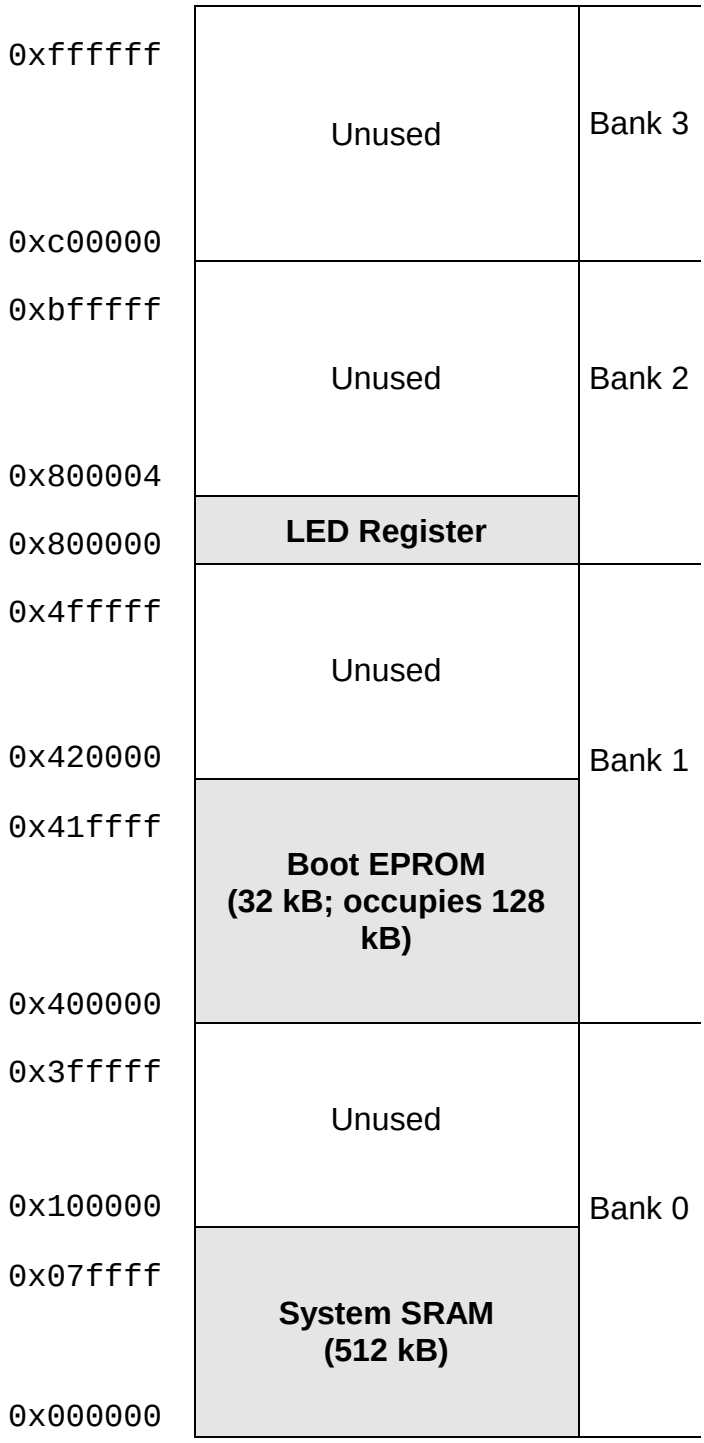
The first three chip select lines from the ELAN 1x100 (i.e., MCS[0]*, MCS[1]* and MCS[2]*) are tied to the SRAM, the EPROM and the LED register, respectively; the remaining chip select is unused. This maps the SRAM into the first bank of address space, the EPROM into the second bank, and the LED register 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 1x100 may be found in the relevant application notes and reference design documents.

System Memory Map

The ELAN 1x100 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

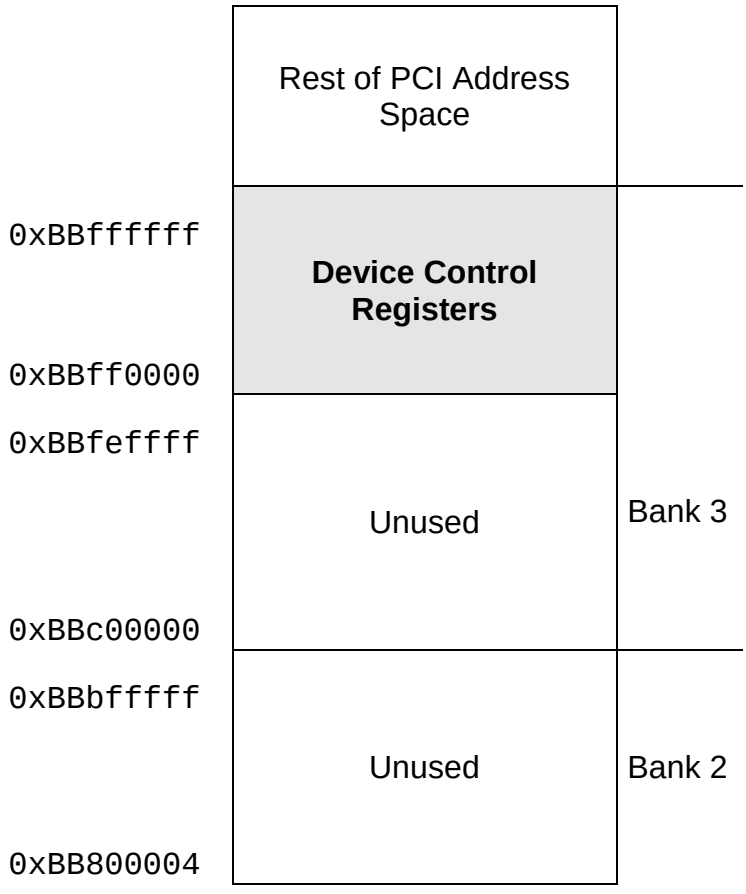
local memory bus, it is as follows (note that the memory addresses shown increase upwards):

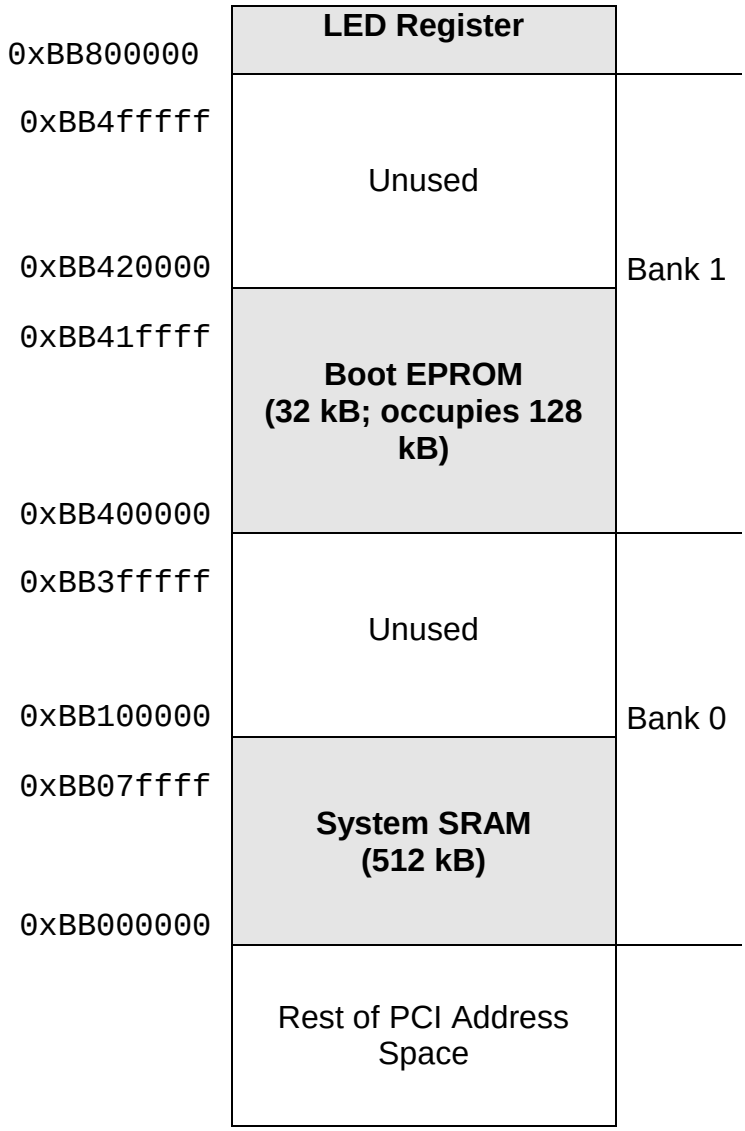


Note that the 32 kB boot EPROM actually occupies 128 kB of address space. This is because the EPROM is only 8 bits wide (i.e., a 32k x 8-bit configuration), and so is connected to the least significant byte lane of the memory data bus. Hence each byte of the EPROM takes up a full 32 bits worth of address 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 1x100 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 1x100 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 1x100 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 1x100 device (note that the memory addresses shown increase upwards):





The start address of the block occupied by the ELAN 1x100 is defined by the setting of the memory base address register within the PCI configuration register space of the ELAN 1x100 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 1x100 operating firmware parameters are set consistently with the selected base address. Alternatively, the base address may be set to a value between 0x00 and 0x0F, during reset, by the configuration registers

Device Internal Blocks

The ELAN 1x100 consists of the following major components: a Switch Processor, one 100Mbit/s full-duplex MAC interface, a DMA Controller, a memory controller and a PCI expansion port.

Switch Processor

The Switch Processor is a 50 MHz proprietary RISC CPU that executes the firmware required for carrying out all the frame switching and device control functions of the ELAN 1x100. 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 1x100 device logic. The Switch Processor interfaces to the rest of the ELAN 1x100 device via several dedicated hardware ports:

- 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 1x100 units, read the unit status, set various operating parameters (such as address pointers), and perform device self-test.

- It implements a set of thirteen level-sensitive *hardware interrupts* that are connected to various blocks within the ELAN 1x100; 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.

- 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 1x100 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.

- 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 1x100 functional units to the Switch Processor firmware.

The internal and external registers implemented by the Switch Processor and the associated ELAN 1x100 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.

100Mbit/s Ethernet MAC Interface

A complete Ethernet/IEEE MAC-layer interface is built into the ELAN 1x100 chip, communicating with external 100 Mbit/s transceivers via the on-chip MII port. The MAC-layer interface logic performs most of the processing tasks required by the IEEE standard in hardware. In addition, the MAC interfaces contain FIFO buffers that enhance throughput and reduce bandwidth loss due to frame delays. The MAC layer interfaces also permits loopback testing of the external transceiver.

The PM3351 also supports the MII management interface to an MII compliant PHY device. This is accomplished by using the MII serial management pins (MDIO and MDCLK) in conjunction with firmware on the Switch Processor. This allows the PM3351 to support auto-negotiation, link status, and other management operations on the attached PHY device(s). At system initialization time, the Switch Processor polls the PHY device to determine the starting configuration resulting from the autonegotiation process, prior to beginning normal operation; thereafter, the Switch Processor periodically re-interrogates the PHY device to determine whether the configuration is still valid, or whether a configuration change has occurred.

The ELAN 1x100 performs only the 802.3 MAC-sublayer functions and provides a simple, inexpensive interconnection to an attached physical layer (PHY) device through the Medium Independent Interface (MII) as described in 802.3u, clause 22. The attached 100Mbit PHY device is responsible for performing the Physical Coding Sublayer, Physical Medium Attachment, and Physical Medium Dependent sublayers; that is, it is the interface between the actual Ethernet media on one side and the MII interface on the other side.

The MAC interface unit consists of the following functional blocks:

- A signal interface compliant with the MII logical and electrical specifications. The interface communicates with the PHY device using the TXD[3:0], TXEN and TX_CLK signals for transmit data, and the RXD[3:0], RX_DV, RX_ER and RX_CLK signals for receive data. In addition, the CRS and COL signals are used for medial access during half-duplex operation. Note that the TX_ER signal is not directly supported by the ELAN 1x100; the TX_ER input on the PHY device, if present, should be tied LOW.1

[1] The ELAN 1x100 does not provide a TX_ER signal as the device, being store-and-forward, will never transmit an errored frame; in addition, the logical implementation is such that it is impossible to have a data starvation condition occur during transmit.

- A series of two unidirectional buffer FIFOs for receiving data from the MII media: the first is a 32-deep nibble-wide FIFO that receives framed nibbles from the MII receive channel pins; the second is the primary 64 byte receive FIFO that assembles the nibbles into 32-bit data words and buffers them prior to transfer to the external SRAM. Additional control logic is provided to strip off the preamble and Start Frame Delimiter (SFD) from the receive data, to assemble the framed data nibbles into octets, and to perform clock synchronization between the MII receive clock and the internal system clock. The assembled and buffered data words are then passed to the DMA Controller, which in turn writes them out to local memory.

  Note that the MAC interface unit can begin receiving the next incoming frame while the DMA is still servicing the previous frame; control information is inserted into the FIFOs to ensure that the two frames are properly separated before transfer to memory.

- A 2048-byte transmit FIFO is provided to buffer transmit data received from other ELAN devices over the PCI expansion bus interface prior to transmission on the MAC transmit channel. This FIFO is large enough to hold more than one maximum sized frame of 1518 bytes (plus an additional 512 bytes of the next frame), or up to 32 minimum sized (64-byte) frames, allowing frames to be stacked back-to-back for continuous transmission in the presence of PCI bus latencies.

  Frame transmission on the MII port only starts if an entire frame is present in the transmit FIFO, and the Switch Processor has indicated that the frame is to be transmitted. During transmission, data octets are read from the transmit FIFO by the transmit MAC interface logic, synchronized to the MII transmit clock, and then output on the MII transmit data nibble pins. Additional logic in the transmit interface implements the lower-level frame transmission process described in the 802.3 specification (preamble and SFD insertion, interframe gap timing, collision detection and carrier deference for half-duplex, etc.).

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

MAC transmit sequence

During transmission, 8-bit data octets to be transmitted are passed from the transmit FIFO to the link transmit interface where they are synchronized to the MII transmit clock (TX_CLK). After any required deference or interframe gap time, the MII framing signal TX_EN is asserted and the preamble and SFD are sent on the TXD[3:0] pins. After the SFD, the data octets comprising the Ethernet frame header and payload are disassembled into nibbles, which are then transmitted on the TXD[3:0] pins, followed

ultimately by the Frame Check Sequence (FCS) word. The MAC interface unit will then insert an interframe gap before proceeding to the next frame.

A 2-part interframe gap timing algorithm (functioning according to the IEEE 802.3 specification) is implemented. If a two-part interframe gap is not desired, it is possible to set the Part 1 timing to zero, in which case the device will implement the Ethernet V2.0 (Blue Book) interframe gap behavior. The MAC logic also implements an internal jabber counter to time out excessively long transmissions; thus the ELAN 1x100 does not require the external PHY to provide jabber protection.

Half-duplex Normal Collision

In full-duplex mode, the MII signals COL (collision detect) and CRS (carrier sense) are treated as don't cares, and have no affect on frame transmission and reception. In half-duplex mode, however, the COL and CRS signals are used by the MAC transmit logic to implement the required collision sensing and deference processes as outlined in the 802.3 specification.

If a collision is detected (as reported by the external PHY on the COL pin) prior to the first 512 bits of the frame being transmitted, a normal collision is declared. The MAC port immediately ceases frame data transmission; instead, it forces transmission of a 32-bit (8-nibble) jam sequence, flushes the transmit data disassembly registers, and signals to the Switch Processor that a collision has occurred, as well as an indication that a normal collision was detected. The jam sequence used by the MAC is a series of 8 nibbles with TXD[3:0] of 5 hexadecimal (0101 binary).

In response to the collision indication, the Switch Processor will load an internal backoff timer with the appropriate pseudorandom value that denotes the backoff time to be counted out by the MAC transmit logic. When the backoff period is timed out, the MAC logic will automatically re-attempt transmission of the frame, which will be retained in the transmit FIFO. If sixteen transmission attempts of a single frame have ended in collisions, however, the Switch Processor will discard the frame and clear it from the transmit FIFO.

Half-Duplex Late Colsion

A late collision is declared if the collision was detected after 512 bits of the Ethernet frame, including the preamble and SFD, had already been transmitted. In this case, the MAC will abort the transmission of the frame and send the jam sequence as usual, but the Switch Processor will (upon notification of a late collision) cause the frame to be dropped from the transmit FIFO and not retried. This is required as a consequence of the dynamic re-allocation of space within the transmit FIFO; once more than 64 bytes of a frame have been transmitted (i.e., the late collision threshold has been crossed), the space occupied by the transmitted portion of the frame in the transmit FIFO will begin to be continuously deallocated and used to fetch and buffer the next frame. As a result, a

late collision will usually preclude the retransmission of the original frame data, as it is no longer present within the FIFO.

MAC Receive Sequence

Data nibbles are received from the MII receive data signal pins (RXD[3:0], in conjunction with the clock, framing and error signals) and assembled into 8-bit data bytes, which are then written to the receive FIFO. The data words in the receive FIFO is subsequently read out by the DMA Controller as a stream of 32-bit words, and transferred to buffers in the external memory.

At the end of frame reception, the MAC logic provides the DMA Controller with the received frame status: frame too long (i.e., the received frame exceeded the configurable maximum frame size), frame too short (i.e., the frame length was below the configurable minimum frame size), misaligned frame (a framing error was detected during nibble-to-byte conversion), or FIFO overrun (data was lost because the receive FIFO was not drained at a sufficiently high rate). The DMA Controller combines this status with its own internally generated error conditions when supplying the Switch Processor with the general received frame status.

Oversize receive frames are truncated by the MAC logic to prevent buffer exhaustion during the reception of a jabber. Assertion of the MII RX_ER signal while RX_DV is asserted will force a CRC error if the data portion of the frame is being received, and will cause the frame to be dropped if the preamble or SFD is being received.

The ELAN 1x100 MAC interface is configurable via a set of configuration registers that are loaded at initialization time. The following parameters are configurable:

Parameter	Min	Max	Units	Default	
				Numeric Setting	Value
Interframe spacing, Part 1	0	255	nibble times[1]	15	60 bit times
Interframe spacing, Part 2	0	255	nibble times	9	36 bit tmes
Preamble length (not including SFD)	0	255	bits	52	56 bits
Collision jam length	0	255	byte times	4	32 bits
Maximum receive/transmit frame size	0	2048	bytes	1518	1518 bytes
Minimum receive frame size	0	255	bytes	64	64 bytes
Late collision threshold	0	255	byte times	64	512 bit times
Receiver blind time	0	255	nibble times	0	0 bit times

Frame Type Recognition

The MAC interface (in conjunction with the DMA Controller receive channel and the Switch Processor) also allows selected frames to be received and specially handled by the ELAN 1x100, based on the EtherType field within the frame. (Note that this capability applies only to frames coded according to the Ethernet V2.0 standard, not the IEEE 802.3 standard with LLC coding.) This facility can be used, for example, to separate Address Resolution Protocol (ARP) frames from normal TCP/IP traffic, or for recognizing and diverting MAC Control frames for flow control and other purposes.

The frame type recognition capability is implemented using a 16-bit register, referred to as the ETYPE register, which is used to compare the EtherType fields of all received frames to a predefined value. If a match is found, an indication is passed to the Switch Processor by the DMA Controller at the time that the former is notified of the presence of the received frame. The ETYPE register is normally set to 0x8808 hex after system reset (i.e., the EtherType value corresponding to full-duplex MAC Control frames as per IEEE 802.3x, clause 31); the Switch Processor may, however, modify this register if some other type is to be recognized. Note that only one ETYPE register is provided, and hence only one EtherType value may be checked for at any time. (Additional type checking is performed by the Switch Processor firmware as required.)

Half-Duplex Flow Control: Backpressure

Flow control is supported as an option during half-duplex operation by means of a backpressure mechanism, activated by the Switch Processor when an out-of-buffers condition is detected during a high volume of received traffic. Backpressure is

[1] A nibble time is one RX_CLK or TX_CLK period on the MII interface (nominally 40 ns).

implemented by the MAC logic 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 1x100 (e.g., a frame was received over the PCI expansion bus and 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: it should be noted that this 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.

Full-Duplex Flow Control: PAUSE MAC Control frames

Backpressure flow control relies on collisions to shut off frame reception, and hence cannot be used during full duplex operation. PAUSE MAC Control frames, as specified in IEEE 802.3x, annex 31B, are employed instead. PAUSE frame transmission is controlled by the Switch Processor: if the Switch Processor determines that the amount of available receive buffer space has fallen below a threshold, it will initiate the transmit of a MAC Control frame that is formatted as a PAUSE frame according to the IEEE 802.3x standard, with a "pause_time" field set to a configurable number of slot times (nominally 100). Additional PAUSE frames, with the same pause timer value, are transmitted if the condition persists after the pause period ends.

The MAC logic also supports full duplex flow control attempts by downstream entities when they are unable to accept data transmitted by the ELAN 1x100. In this case, PAUSE frames are **received** with their EtherType fields set to 0x8808 hexadecimal, indicating a MAC Control frame. The Switch Processor will then verify that the frame is valid, shut off frame transmission at the next inter-frame boundary, and time out the requested pause time (using a software-based method). If no more PAUSE frames are received before the pause timer expires, transmission will recommence.

CLK25

The ELAN 1x100 provides a CLK25 output that can be tied directly to the 25 MHz input clock used by most 100 Mbit/s Ethernet PHY devices.

Multichannel DMA Controller

The on-chip DMA Controller contains four independent and concurrently operating channels: two for the 100 Mbit/s MII port (one for receive and one for transmit) and two dedicated to the 1 Gbit/s expansion port. The DMA Controller, operating under the control of the switch processor, is responsible for performing all data block transfers within the ELAN 1x100 made between the MII port, the local memory and the expansion port. The DMA Controller also computes 32-bit IEEE Frame Check Sequence (FCS) remainders over the received and transmitted frames, allowing the Switch Processor to filter frames with CRC errors on receive, and also allowing CRCs to be computed for injected management frames on transmit. In addition, the DMA Controller implements an address lookup process that attempts to map the 48-bit MAC addresses of received frames to entries in an address table maintained in local memory. The DMA Controller provides sufficient bandwidth to guarantee that there will be continuous and uninterrupted reception and transmission of frames at full wire rates.

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) These allocatable data structures are called *packet buffers*; they are of fixed (and configurable) length, and are chained together into linked-lists to hold Ethernet frames of different lengths. The DMA channel associated with the receive path is capable of creating these linked-lists of packet buffers automatically whenever frames are received.

The four DMA Controller channels are dedicated to various functions as follows:

- One channel (referred to as the receive channel) is used to perform data transfers between the MAC receive FIFO and the local memory, copying received frame data from the FIFO to the local memory. As mentioned, the receive channel generates a linked-list of packet buffers in external memory, automatically allocating packet buffers as needed from a central free pool. A 32-bit CRC is computed on the receive data transfers, and the CRC result is accessible to the Switch Processor firmware in the form of a CRC error status bit after a frame has been received.

- One channel (referred to as the transmit channel) is used to perform block transfers over the external PCI expansion bus from external devices (other ELAN 1x100s, ELAN 8x10s, or any other device implementing the same protocol) to the 2048-byte transmit FIFO. This channel can follow a chain of remote source packet buffers, reading data over the PCI bus. Only the required frame data is written into the transmit FIFO; the headers of the packet buffers are stripped off by the channel transfer logic.

The transmit channel implements special logic that allows the Switch Processor to inspect the frame header and determine how the frame should be treated after it has

been read into the transmit FIFO. This consists of a small 16-entry queue (referred to as the disposition FIFO), that holds disposition commands generated by the Switch Processor. After the header of each frame has been read over the PCI expansion bus, the transmit channel notifies the Switch Processor via an interrupt; the Switch Processor is then responsible for checking the header data and writing a disposition command into the disposition FIFO. The disposition command instructs the transmit channel logic as to how the frame should be handled. In all cases, a disposition command applies only when a frame has been completely read into the transmit FIFO, and the first byte of the frame reaches the head of the transmit FIFO.

The purpose of the disposition FIFO is to improve the efficiency of the switching firmware when handling transmit frames. Each entry in the disposition FIFO corresponds to a specific frame that has been placed into the main 2048-byte transmit FIFO, and indicates how that frame is to be dealt with. Three possible actions may be requested by the firmware, by setting entries in the disposition FIFO to various codes:

1. Send: the frame can be automatically sent over the MAC port when it reaches the head of the transmit FIFO, i.e., the preceding frame, if any, has been deleted from the FIFO. This is the normal mode of operation, used for dealing with frames that should be transmitted without special processing.
2. Drop: the frame can be simply deleted from the transmit FIFO when it reaches the head of the FIFO, without transmitting it over the MAC port. This is used when the ELAN 1x100 decides to reject a particular frame that was queued for it from an external device, or for special firmware-controlled frame processing.
3. Stop: when the first byte of the frame reaches the head of the transmit FIFO, the DMA Controller should stop reading any more data out of the transmit FIFO, and instead interrupt the Switch Processor. The Switch Processor firmware may then choose to read data out of the transmit FIFO if required; when the firmware processing is done, the disposition FIFO entry may be changed to a drop code to delete the frame from the transmit FIFO, or to a send code to transmit the frame over the MAC port in the normal manner.

The use of the disposition FIFO permits the firmware to make early decisions about how the frame should be handled, even before the frame has been completely read into the transmit FIFO. These decisions are expressed in the form of the disposition codes (send, drop, and stop), which travel down the disposition FIFO in lockstep with the stored frames travelling down the transmit FIFO. The DMA Controller ensures that one entry is read from the disposition FIFO for each frame that is read out of (or dropped from) the transmit FIFO.

- One channel, referred to as the PCI access channel, is used to perform block transfers under control of firmware running on the Switch Processor. This channel is capable of performing PCI configuration read or write transactions as well as standard PCI memory read or write operations. It is not intended to handle frame data, but is generally used for control and inter-device communication functions.

- The last channel (referred to as the transfer handshake channel) facilitates the frame exchange handshake that takes place between multiple PM3350 and PM3351 devices across the PCI expansion 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 devices (or in an external device that implements the ELAN frame transfer protocol). The request and acknowledge counters are used to indicate the presence of a frame being forwarded and to acknowledge receipt of the frame, respectively. This channel also supports hardware that implements a ring of data descriptors that are used for the seven transfer queues (discussed in more detail later).

DMA Controller operations performed by the receive channel take place largely autonomously (after initial setup); the receive channel simply notifies the Switch Processor upon the completion of reception of each incoming frame, and supplies the necessary frame parameters to the Switch Processor via hardware registers. However, the remaining channels perform their transfers completely under firmware control by the Switch Processor. In operation, the Switch Processor loads the appropriate DMA control registers with the desired transfer parameters and then enables the channel. The hardware will then autonomously complete the transfer and notify the Switch Processor upon completion via an interrupt.

Address Lookup Hardware

The DMA Controller contains internal hardware that performs an address lookup on the source and destination MAC addresses in the headers of received Ethernet frames, and attempts to match them against entries in an address table maintained in external RAM.

During frame reception, the DMA receive channel extracts the 48-bit source and destination MAC addresses from every incoming frame. These are then passed through a hashing process that generates a 16-bit *hash key*; the key, in turn, is used as an index into a *hash array* in external RAM. The entries in the hash array are expected to point to zero or more *hash buckets*, with each hash bucket being assigned to a unique MAC address. If multiple hash buckets are associated with a single hash array entry, they are concatenated into a linked list, that is searched by the address lookup hardware to locate the correct hash bucket for the given source or destination MAC address. (The search is performed by comparing the actual MAC address against a 48-bit address field in the hash bucket.) If a match is found for either the source or the

destination MAC addresses, or both, the pointers to the appropriate hash buckets are passed to the Switch Processor along with the received frame.

A similar address lookup process is also performed by the transmit channel for a frame that is being read over the PCI expansion bus from a remote device. This lookup is restricted to the source MAC address only; further, the lookup is only done if required by the Switch Processor. (This address lookup is used during the address learning process, when it is desired to learn the source MAC addresses of frames that are being read over the PCI expansion bus.)

Memory Controller

External RAM is required for packet buffers used to hold received Ethernet packets, as well as data structures needed to perform switching and support system management. The external memory may also contain patch or extension code for the on-chip Switch Processor. The ELAN 1x100 therefore contains an integral memory controller unit which supports a variety of standard memories without glue logic.

The memory controller is capable of addressing a total of 16 megabytes of row/column multiplexed-address memories (i.e. DRAM), or a total of 4 megabytes of linear-address memories (i.e. SRAM, EPROM and EEPROM), in any combination.[1] The address space is divided into four banks, each of which has an independently programmable memory type and access time. This allows a mix of fast and slow devices to be used in the system. Programming of EEPROM devices is done under control of Switch Processor microcode. The memory controller is capable of performing up to 1 32-bit transfer every 2 clock cycles (40 nsec at the nominal clock frequency of 50 MHz) to or from an SRAM bank, for a maximum sustained throughput of 100 MB/s. Memory device types or speeds cannot be mixed within a bank (i.e., a bank cannot consist of part EEPROM and part SRAM, for example) without special logic that is outside the scope of this document. In general, memory devices of different types must be assigned to different banks, and selected with different chip selects.

The memory types used for standard switching by the PM3351 are summarized in the table below:

[1] The reduced space available for the non-multiplexed-address memories arises due to pin limitations: only 18 memory address lines are driven to the pins driving the external memory bus (MADDR[17:0]), whereas 20 address lines would actually be required (in conjunction with the four bank selects and the four byte-enables) to span the entire 16 MB address space.

Memory Type	Spee (nsec)	Latency (cycles)	Peak Bandwidth
Standard SRAM	15	2	100 MB/s
Standard EPROM/EEPROM	150	12	17 MB/s

The standard memory types that should be used with the ELAN 1x100 are 15 nsec SRAM and 120/150 nsec EPROM or EEPROMA memory ready input pin (MRDY*) is implemented for linear memory devices that have other speed requirements. This pin is sampled by the ELAN 1x100 when accessing memories in standard 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 sampling of the MRDY* input takes place at the end of the first clock cycle of a 2-cycle SRAM access. Accesses to banks configured for memory types other than 2-cycle SRAM cannot be controlled using MRDY*. The MRDY* pin should be tied LOW (i.e., logically deasserted) if it is not used.

The ELAN 1x100 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 1x100 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 input pin (MINTR*) is provided for special applications. A TTL logic 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 (i.e. logically deasserted) if it is not to be used.

PCI Expansion Port

The ELAN 1x100 includes a PCI v2.1 compatible bus master and slave interface, which serves as an expansion port allowing multiple ELAN 1x100 and/or ELAN 8x10 chips to be interconnected in the same system to create switches with larger numbers of ports. The expansion port supports a maximum bus clock of 40 MHz (resulting in a >1 Gbit/s peak transfer rate and a sustained throughput of 500 Mbit/s), and contains several FIFOs to increase burst throughput and perform clock synchronization.

The PCI expansion port may be used for either expanding a switch built around ELAN switch devices (to a maximum of eight PM3351 and PM3350 chips) or to allow the ELAN 1x100 to be used in special applications requiring an intelligent 100 Mb/s Ethernet interface. In these applications, the on-chip DMA Controller uses the PCI expansion port master interface to notify other ELAN switch devices or the host CPU of the presence of packets to be transferred, and to copy packets or data structures under control of the Switch Processor from external ELAN switch devices to the local memory

space via the PCI bus. 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 conforms to the requirements of 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 1x100 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 1x100 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 also conforms to 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 1x100 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 1x100 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 1x100 and ELAN 8x10 devices via the PCI bus. This hardware takes the form of eigth request counters and eight acknowledge counters. Each request/ acknowledge counter pair is dedicated to supporting communications with a specific external ELAN device. An external ELAN 1x100 device will increment a request counter to signal that a frame or message data is available to be read, and will increment an acknowledge counter to acknowledge that it (the external ELAN 1x100) has read a message or frame from the local ELAN 1x100. 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 1x100 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 devices.

The PCI bus interface within the expansion port runs synchronously to the PCI bus clock, which must be supplied on the PCICLK input pin. This clock may range from 25 MHz to 40 MHz, with the duty cycle specified in the PCI Local Bus Specification. The ELAN 1x100 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 1x100 device (in the PCI configuration register space) is **11F8** hexadecimal, while the device number is **3351** hexadecimal. The class code for the ELAN 1x100 is set to 0x020000 hexadecimal.

Slave Read Prefetching

As already mentioned, the PCI bus slave logic contains a 128-byte read FIFO buffer to speed up reads made from this ELAN 1x100 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 that writes are not allowed to be completed out-of-sequence with reads, and vice versa.

Watchdog Timer Facility

The ELAN 1x100 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 1x100 device should be performed if the ERST* output goes LOW (logically asserted).

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.

Device Configuration

Basic device and system configuration (i.e., memory types and speeds for various banks, the PCI base address for this ELAN 1x100 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 1x100 transitions high, and sets up the ELAN 1x100 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 1x100; 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 tristate buffer or tristatable register may be used to drive configuration information on to the data bus during reset. Care should be taken to remove the data by tristating 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]	**Error! No index entries found.** MXSEL / Column Address Bits / DRAM Configurations Supported 00 / 8 / 64K x N & 128K x N 01 / 9 / 256K x N & 512K x N 10 / 10 / 1024K x N & 2048K x N 11 / 11 / 4 Meg x N & 8 Meg x N
MDATA[13]	MSLO	The MSLO bit extends read and write cycles to accommodate slower DRAM devices in local memory devices. If MSLO is high, 80ns DRAM is expected. If MSLO is low, 60ns DRAM is expected. (MSLO must be a logic 0 if EDO DRAM is used with the PM3351 since the device is intended to work with 60 ns DRAM; as previously stated, DRAM, if used, is intended for non-switching applications).
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. (Again, as DRAM is generally not used for standard switching, these bits will usually be don't-cares).
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 / 15ns SRAM 011 / Reserved 100 / Reserved 101 / 150ns (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 1x100 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, **and it must start on a 64 kB boundary**. The ELAN 1x100 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 1x100 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	XXXXXXC7
+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 1x100. 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 1x100 expects to find executable firmware code within the boot image that will perform the actual initialization process. However, the boot image is generally only 8 bits wide, and thus the firmware code supplied within it cannot be directly executed by the ELAN 1x100 Switch Processor, which uses 32-bit instructions located on 32-bit boundaries. It is thus necessary for the ELAN 1x100 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 1x100 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	Address to begin execution at after copy is complete
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 1x100 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; otherwise, indicates a 32-bit boot image in memory. If this bit is not set (indicating a 32-bit-wide boot image), the CpyInt and JmpInt flags must also be cleared.

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). As a 32-bit-wide boot image, can be executed directly, no copy is done, internal or otherwise.

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). It is assumed that a 32-bit-wide boot image will be executed directly from it's external memory location.

If the EightBit flag is set, the ELAN 1x100 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 that can be executed. The target RAM may be either external (SRAM or DRAM) or the internal instruction RAM within the Switch Processor, according to whether the CpyInt flag is set.

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 external SRAM or internal instruction RAM in order to convert the 8-bit boot image to an executable 32-bit image. After the copy and conversion has been done, the BOOTSTART field denotes a 24-bit address from which the ELAN 1x100 Switch Processor will begin executing code, either inside the internal instruction RAM (as is generally the case), or external RAM. The code at this location is responsible for initializing the ELAN 1x100 system and environment. The JmpInt bit signals whether BOOTSTART refers to internal or external RAM.

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 1x100 will consider the boot image as invalid, and will terminate the system initialization and startup process and report an error.

Note that the boot image header above is described as it would appear from the switch processor. In the view from the 8-bit image contained within an EPROM or EEPROM, the addresses would be divided by 4. This is because the 8 bit wide memory 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 1x100. Note that this header only uses the least significant byte even if a 32 bit boot memory is being used. An alternative view of the boot header, giving the components of the header as they would appear inside the 8-bit-wide ROM, is given below:

0x59	**0x37**	**0xa8**	**0xc7**	**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**

Typical Usage

In the standard EPROM or EEPROM bootstrap image, the HDRFLAGS field is set to 0xE0 hex, indicating an 8-bit-wide boot image that must be copied into the Switch Processor's internal instruction RAM, and then executed. The boot image that is copied into the internal RAM includes the normal frame switching firmware, as well as a small bootstrap loader routine. The bootstrap loader (which will be pointed to by BOOTSTART) will then copy the remainder of the 8-bit-wide boot ROM into external memory, and finally jump to an entry point in the code copied into external memory that will configure the rest of the device and the system if necessary.

A 32-bit-wide boot image must always set the HDRFLAGS field to 0x00, or else the bootstrap process will fail. In this case, the CPYTARGET, CPYSIZE, and CPYFROM fields are ignored, as no copying is done. It is expected that the BOOTSTART value will be a valid address in the 32-bit-wide memory containing the boot image where execution must begin in order to initialize and start the system.

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 devices, the MAC address and IP address assigned to this ELAN 1x100, and so on. The boot image must also supply the operating firmware that is required by the ELAN 1x100 for frame switching and management. The general layout of the standard boot image supplied by PMC is shown below (note that the memory addresses increase downwards):

0	Boot Image Signature
	Boot Image Header
	ELAN 1x100 Switching Firmware
	Bootstrap Loader Routine
	System Table
	ELAN 8x10 Switching Firmware
	Boot Tag Space
32 kbytes	Boot Tag Processing Code
	RTOS Firmware Code (optional)
	UDP/IP Stack Firmware Code (optional)
256 kbytes	SNMP Agent/MIB Firmware Code (optional)

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 1x100 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.

The actual bootstrap firmware is divided into two portions, referred to as the *boot tags* and the *boot tag processing code*. The boot tags essentially comprise a set of parameter blocks that define various data structures or hardware entities that must be initialized in a particular way, as well as the data values required for the initialization. The boot tag processing code scans over the array of boot tags; for each class of boot tag, an associated routine is invoked to actually perform the required initialization. This approach permits the initialization process to be extended very simply to cover new types of structures or new hardware features: additional boot tags (together with their associated routines) can be included in order to perform new types of initialization in a well-structured manner, without having to be concerned about interactions between different portions of the bootstrap firmware.

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.

Master/Slave System Initialization

In a multiple-ELAN 1x100 and ELAN 8x10 system, a single ELAN 8x10 or ELAN 1x100 device may be designated as a master and possess a boot image in an EPROM or EEPROM. This ELAN 1x100 is then responsible for initializing and configuring all the other ELAN 1x100s 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 1x100s in the system. In this application, the master ELAN 1x100 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 1x100 can then configure itself, download boot information to the RAM interfaced to the remaining ELAN 1x100s, and finally enable all of the slave ELAN 1x100s to start running. The slave devices can then initialize themselves using the downloaded information.

Note that the firmware images for both the ELAN 1x100 and ELAN 8x10 devices are present in the boot image. This is done to permit a single boot image (and consequently EPROM or EEPROM) to be used to initialize an entire system of ELAN devices of either type. The System Table is used for this purpose: it contains information regarding the initialization and operating code requirements of every device in the system, and the bootstrap firmware running on the system master device uses this table to properly initialize and start the slave devices. Slave initialization is carried out by the master device after the master has loaded its boot image, set up its private environment, and determined that it is indeed the master; at this time, it copies the bootstrap firmware, the appropriate block of switching firmware (depending on whether the target slave device is an ELAN 1x100 or an ELAN 8x10) and the appropriate set of boot tags into the slave device's local RAM space across the PCI bus. The master then causes the slave to begin executing the bootstrap firmware and initialize itself. Once initialization is complete, the slaves notify the master, and the system is ready for frame processing.

Host-Controlled System Boot

In a host-controlled system, it is expected that all of the ELAN 1x100 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 can be done by converting the bootstrap firmware to run on the host system. The bootstrap firmware should then perform the same general functions (reading each boot tag and implementing the required initialization function) to pass the appropriate information to the ELAN devices. Contact the factory for more information and sample code.

Stand-Alone System Boot

Typically an ELAN 1x100 device will be part of a multiple ELAN device system, where the master device is responsible for controlling the initialization of the slaves; or work in conjunction with a host processor, in which case the host is required to take over these initialization and start-up functions. However, it is possible for a system to be designed where each ELAN device has its own EPROM or EEPROM containing a private boot image, and it is not necessary for a master entity to initialize other devices. In such a system, some elements of the multi-device boot ROM are unnecessary; the resulting boot image would be laid out as follows:

0	Boot Image Signature
	Boot Image Header
	ELAN Switching Firmware
	Bootstrap Loader Routine
	Boot Tag Space
32 kbytes	Boot Tag Processing Code
	RTOS Firmware Code (optional)
	UDP/IP Stack Firmware Code (optional)
256 kbytes	SNMP Agent/MIB Firmware Code (optional)

Contact the factory for further information and sample boot images.

Configuration Parameters

A number of configuration parameters can be adjusted by the system implementer to create a boot image for various types of target systems. The configuration parameters are located in a single header file that is read automatically whenever a boot image is

created using the development kit. The following table gives the user-accessible configuration parameters:

General System Parameters	
MstrChipNum	Chip number assigned to master device (generally 0)
HashMask	Mask setting size of hash table array (0x1fff max)
NumChips	Total number of ELAN 1x100 and ELAN 8x10 devices
ChipBase0	Upper 8 bits of base address of device 0 on PCI bus
ChipBase1	Upper 8 bits of base address of device 1 on PCI bus
ChipBase2	Upper 8 bits of base address of device 2 on PCI bus
Error! No index entries found.	Upper 8 bits of base address of device 3 on PCI bus
ChipBase4	Upper 8 bits of base address of device 4 on PCI bus
ChipBase5	Upper 8 bits of base address of device 5 on PCI bus
ChipBase6	Upper 8 bits of base address of device 6 on PCI bus
ChipBase7	Upper 8 bits of base address of device 7 on PCI bus
Boot Image Header Parameters	
BOOT_START	Target area reserved for copying boot image from ROM
BOOT_ENTRY	Entry point of bootstrap loader/firmware in copied image
BOOT_SIZE	Size of boot image in 64-byte blocks (i.e., ROM copy size)
LED_START	Address of LED status register for signaling status codes
ROM_LOC	Boot image EPROM/EEPROM base address (copy source)

Parameters Specific to ELAN 1x100	
BIGSTRUCTst	Start of pool of free packet buffers
nBIGSTRUCTs	Number of free packet buffers in pool
HASHSTRUCTst	Start of pool of free hash bucket structures
nHASHSTRUCTs	Number of free hash bucket structures in pool
SMALLSTRUCTst	Start of pool of free forwarding tag structures
nSMALLSTRUCTs	Number of free forwarding tags in pool
DISPSTRUCTst	Start of pool of free data descriptor structures for messages
nDISPSTRUCTs	Number of free message descriptors in pool
DDRINGst	Start of transfer ring structures
nDDsPERRING	Number of elements per transfer ring
nDDsPERRINGshift	Log_2 of number of elements per transfer ring
Parameters Specific to ELAN 8x10	
XPD_DD	Address of first data descriptor attached to transfer queues
FreeHB	Start of pool of free hash bucket structures
FreePB	Start of pool of free packet buffer structures
FreeDD	Start of pool of free data descriptor structures
num_FreePBs	Number of free hash bucket structures in pool
LclPBLim	Buffer limit per port

Note that further information on ELAN 1x100 data structures can be found later in this document.

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 1x100 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 1x100 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 precomputed 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 pseudorandom 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 1x100 operating firmware also performs numerous checks of its internal state during normal system operation. If an unrecoverable error is detected, the ELAN 1x100 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 1x100 will reset the entire system; otherwise, the ERST* pin should be monitored by an external system master to determine when the ELAN 1x100 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 LEDs also serve to output special codes during system initialization and during normal operation. In this case, LEDs 6 and 7 will not be lighted. LED 5 indicates whether the status pertains to system initialization or normal operation. Codes output during system initialization indicate the index of the bootstrap tag being processed during the initialization process, and range from 0 to the maximum number of bootstrap tags in the boot image.

The codes written out to the LED register are defined below:

Description	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0
Bootstrap Tag Index	Off	Off	Off	Tag[4]	Tag[3]	Tag[2]	Tag[1]	Tag[0]
Operational Status Indication	Off	Off	On	Info[4]	Info[3]	Info[2]	Info[1]	Info[0]
Boot Checksum Failure	Blinking	On	Off	Off	Off	Off	Off	On
RAM Self-test Failure	Blinking	On	Off	Off	Off	Off	On	Off
Operational Failure	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 1x100 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 signalling tag processing information, operational information and operational errors during operation are TBD.

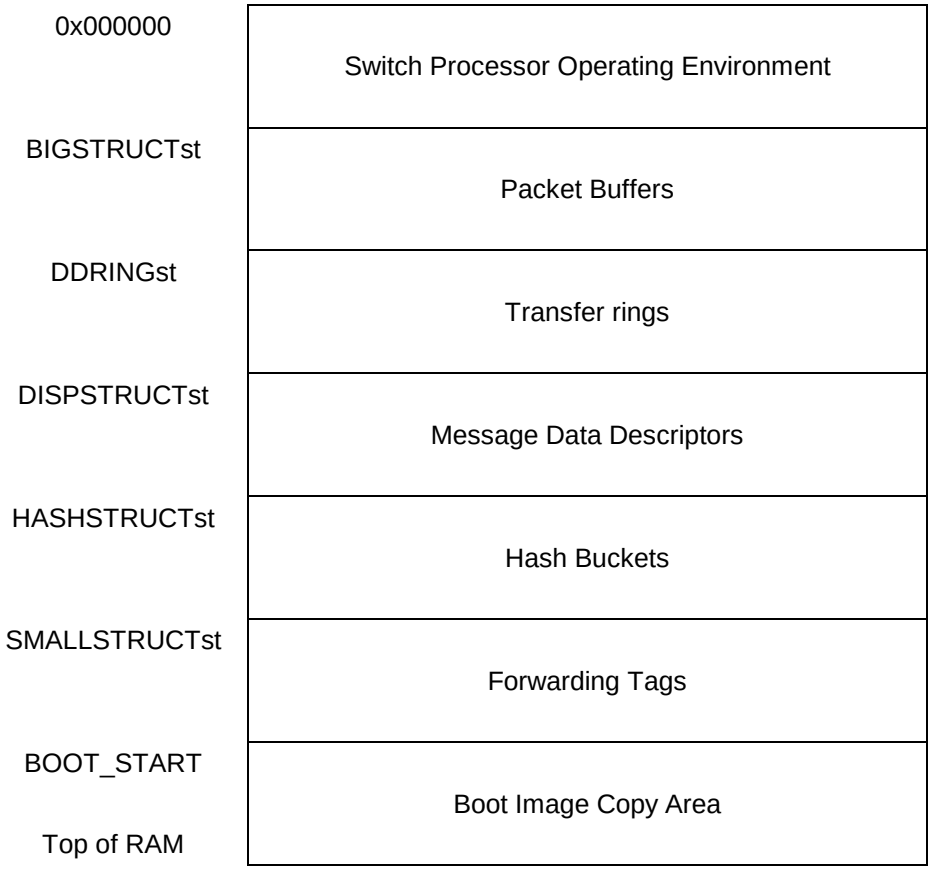
Data Structures

It is assumed that an earlier bootstrap initialization and configuration phase will have set up some predefined data structures in ELAN 1x100 local memory. These are the Switch Processor operating environment (stacks, memory pool, local variables, etc.), Transfer rings, free pools (containing Packet Buffers, Data Descriptors for messages, Hash Buckets and Forwarding Tags), 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.

The general layout of the various data structures in RAM is shown below. Note that addresses increase downwards. The actual addresses delimiting various regions of memory are implementation-specific, and not provided here; instead, the names of the

parameters supplied during the generation of a boot image that control these addresses is given next to the memory regions. A description of how these addresses may be computed, as well as a sample memory map for a typical system configuration, is provided later in this section.



Switch Processor Operating Environment

The Switch Processor requires a small operating environment (i.e., data structures and variables) for performing basic frame switching functions. This environment is set up in the external RAM during the bootstrap initialization and configuration phase. The entire operating environment occupies about 43 kbytes of space, and must be located starting at address 0x000000 hex in the external memory. The operating environment excludes the space used by Ethernet frame buffers and the queueing structures used to track them, and also does not include the code, data or stack spaces that are needed by the (optional) RTOS, UDP/IP stack, or SNMP agent firmware. The general layout of the operating environment is as follows:

0x000000 0x001FFF	Reserved for switching firmware
0x002000 0x00207f	Frequently used variables and parameters (cached)
0x002080 0x0020bf	Save space for switching firmware (cached)
0x0020c0 0x0020ff	Local Port Descriptor (cached)
0x002100 0x0021ff	Background Queues (cached)
0x002200 0x0022ff	Random Number Generator Table (cached)
0x002300 0x0023ff	Expansion Port Descriptors (cached)
0x002400 0x0024ff	Dispatch Table
0x002500 0x0025ff	Miscellaneous variables and tables
0x002600 0x00267f	Port Descriptor Error Counters
0x002680 0x0028ff	Transmit Distribution FIFO
0x002900 0x00a8ff	Hash Table

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 1x100.

2. The next 128 bytes are used to hold various frequently referenced, global 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 64 bytes is reserved for use by the switching firmware as a register save space during interrupts. This area is also cached.

4. A Local Port Descriptor is placed in the next 64 bytes. The Local Port Descriptor also resides 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 Descriptor contains statistics and control information used during frame switching, and is described in further detail below.

5. A set of 16 Background Queues are placed in the next 256 bytes. These queues are used to transfer frames and messages from the foreground switching tasks to background and management tasks. The Background Queues reside in the cached memory area.

6. The next 256 bytes are reserved for the Random Number Generation Table. This table contains an array of seed values that are used to generate pseudo-random numbers during the computation of backoff timer values for half-duplex collision handling.

7. A set of eight Expansion Port Descriptors occupies another 256 bytes. These data structures hold information pertaining to the devices accessible via the PCI expansion bus. 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.

8. A Dispatch Table, consisting of an array of pointers to firmware routines that acta as handlers for special situations or for error recovery purposes. A block of 256 bytes is reserved for the Dispatch Table. Note that the Dispatch Table is located immediately following the cached region of the Switch Processor operating environment; this and all subsequent data structures are not cached.

9. The next 256 bytes are reserved for miscellaneous variables utilized on an infrequent basis by the switching firmware. This includes things such as a generic hash bucket structure and debug variables.

10. A Port Descriptor Error Counter structure is provided to hold error and collision statistics for the local (MAC) port. The counter block requires 96 bytes of

storage; the remaining 32 bytes being reserved for future applications. These counters are also described in more detail below.

11. The next 640 bytes are used for a firmware-maintained Transmit Distribution FIFO. This FIFO holds flags and address table information for frames that are queued for transmit in the internal 2048-byte transmit FIFO, and is used by the Switch Processor for post-transmit processing.

12. The remaining 32768 bytes of space 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 frame 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 (ranging from 0x2000 through 0x23ff hex, in the memory map above) 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

The Switch Processor operating environment contains a number of variables and tables used during normal operation. Some of these memory locations (i.e., those located between addresses 0x2000 and 0x20c0 hex 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. The following is a brief listing of the significant variables in the operating environment:

Variable Name	Width (bits)	Description
cdHStruct	24	Pointer to head of linked-list of free hash buckets
cdSStruct	24	Pointer to head of linked-list of forwarding tags
cdDFIFOwr	24	Pointer to head entry in Distribution FIFO
Error! No index entries found.	**Error! No index entries found.**	**Error! No index entries found.**
cdTotCollide	32	Total number of collisions encountered on MAC port
cdDStruct	24	Pointer to head of free data descriptor pool reserved for holding messages
cdCurrentXPD	24	Temporary storage: holds address of expansion port descriptor currently in use
cdLNR	32	Temporary save space for LNR register in Switch Processor
cdFirstHash	24	Temporary storage: holds the address of the first hash bucket or forwarding tag in a chain of address table entries
cdHashIdxMask	16	Bitmask used to restrict the range of the hash key computed as an index into the hash array
cdRefMask	16	Global bitmask restricting devices and ports to which multicasts and broadcasts may be directed * 8 LSBs are unused, set to zero * 8 MSBs correspond to the eight possible devices in a system
cdChipNum	8	Logical chip number assigned to this ELAN 1x100
cdNumChips	8	Total number of ELAN chips in system (both ELAN 1x100 and others)
cdPCIRefCount	8	Reference count to use during broadcasts
cdAgeTick	8	Aging task trigger flag
cdAgeEnable	8	Aging task enable flag
cdDDsInRingShift	16	Log_2 of number of elements in each transfer ring structure
cdFPullTl	24	Pointer to tail of management frame being extracted from internal transmit FIFO by firmware
cdFPullHd	24	Pointer to head of management frame being extracted from internal transmit FIFO by firmware
cdFPullBytes	16	Number of bytes remaining for management frame being read out of internal transmit FIFO by firmware
cdDispatchPCI	24	Pointer to next free message data descriptor to use for queueing messages for background processing
cdFPush	24	Pointer to management frame being written into internal transmit FIFO by firmware

Packet Buffers

Ethernet frames are stored in the external SRAM by the ELAN 1x100 chip in small, fixed length *packet buffers*; with multiple packet buffers being chained in a linked list to hold a complete Ethernet frame. The size of every 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 fields, 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 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
Last Size	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	PayloadByte0	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 frame: gives the number of Data Descriptor Ring queues that are pointing to this packet buffer.

LastSize:

8-bit count of total number of valid payload bytes (not 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 a 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 and hence the end of the linked list. 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, not 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 the MAC channel of the ELAN 1x100 (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 frames, messages and other pieces of data being transferred between devices within an ELAN 1x100 system, or between tasks on a single ELAN 1x100, are tracked by means of 16-byte *data descriptors* . (When a data descriptor refers to an Ethernet frame that contains management data to be processed by the ELAN 1x100 system itself, or to buffers holding control and status data that are exchanged for messaging purposes within the ELAN 1x100 system, then it is sometimes referred to as a *message descriptor*.)

Data descriptors form the primary queueing elements in the ELAN 1x100 device: in addition to forming the individual elements of transfer rings (see below), they are used to hold message information and to create special frame handling queues for management functions. 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 frame or a control/status message, flags fields that indicate the processing to be performed on the frame or message, and other information required for processing the payload.

In general, 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 message queue or transfer ring.

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 or message data. If NULL (zero), no packet buffer chain exists for this data descriptor; this is only valid for special message data descriptors where all message data can be contained within the 16-byte descriptor structure itself.

NumPBs:

8-bit count of packet buffers in chain pointed to by FirstPB. If FirstPB is NULL (zero), then this field can be used for message-specific purposes.

LastPB:

24-bit pointer to last packet buffer in linked-list of packet buffers pointed to by FirstPB. If FirstPB is NULL (zero), then this field can be used for message-specific purposes.

SrcPort:

8-bit source port index within device: indicates the MAC channel upon which the frame was received. For the ELAN 1x100, this field wil be 0 since the device only has one port. Only valid if the descriptor points to an Ethernet frame that was actually received on a MAC channel; otherwise, can be used for message-specific purposes.

HashBkt:

24-bit pointer to source or destination hash bucket. Only valid for Ethernet frames; if the descriptor indicates a message, then this field can be used for message-specific purposes.

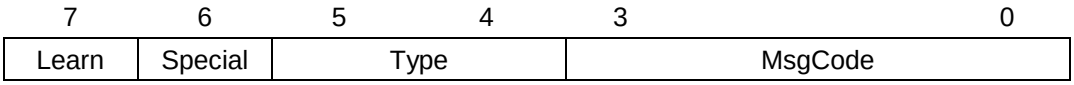
SrcChip:

8-bit index of source device that received frame. For the ELAN 1x100 system,

this must range from 0 through 7. Only valid for Ethernet frames; if the descriptor indicates a message, then this field can be used for message-specific purposes.

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 and hence the end of the linked-list.

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 1x100 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, or between firmware tasks on an ELAN 1x100.

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.

MsgCode:

The 4-bit MsgCode field is only valid if the Special bit is set; in this case, it is used to carry a message code that identifies the type of message carried by the packet buffers to which this data descriptor points.

Msgcode	Interpretation
0000	reserved
0001	reserved
0010	reserved
0011	reserved
0100	Topology change notification message
0101	Forwarding tag deletion request message
0110	reserved
0111	ARP broadcast frame message
1000	reserved
1001	reserved
1010	reserved
1011	reserved
1100	reserved
1101	reserved
1110	reserved
1111	reserved

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 1x100 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 value in the SrcPort field will be always be a constant 0 as the ELAN 1x100 only has one MAC channel; the value placed in the SrcChip field by any ELAN 1x100 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. These fields are generally not valid for message descriptors, in which case they contain message-specific information.

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 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 (i.e., is a message descriptor); in this case, it contains message-specific information.

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 or ring hardware may be read at any time via the PCI expansion port.

Transfer Rings

A set of fixed-size data structures, called *transfer rings*, are used to control the transfer of Ethernet frames and control/status messages to other devices interfaced to the ELAN 1x100 via the PCI expansion bus. Each transfer ring is dedicated to a particular external device, and points to frame or message data that must be sequentially transferred to that device across the PCI bus. Transfer rings thus operate under a FIFO queueing discipline. The general structure of the transfer rings is given below:

DDRINGst	Element 0
	Element 1
	Element 2
	Transfer Ring 0 Element 3
	.
	.
	Element (N-1)
DDRINGst + 16*N	Element 0
	Element 1
	Element 2
	Transfer Ring 1 Element 3
	.
	.
	Element (N-1)
DDRINGst + 32*N	.
	.
	.
	.
DDRINGst + 112*N	Element 0
	Element 1
	Element 2
	Transfer Ring 7 Element 3
	.
	.
	Element (N-1)

Each transfer ring consists of *N* elements, where *N* is a power of 2. (The actual value of *N* is an implementation parameter, and is determined at system initialization time by configuration parameters in the boot image.) Each element in a transfer ring is formatted as a data descriptor structure; the NextDD field in each data descriptor is set up by the bootstrap firmware to point to the next consecutive element in the same transfer ring, with the last element in a transfer ring being set up to point back to the first. The elements in a given transfer ring therefore create a closed linked-list; if the NextDD pointers are followed starting from any element in the ring, the path followed will eventually return to the starting point. The remainder of the fields in each element (beyond the NextDD pointer) are left uninitialized at system startup.

When an Ethernet frame or a control/status message is to be transferred to another device in the system, the first free element in the transfer ring corresponding to that device is set up appropriately to point to the frame or message. (If the message is sufficiently small, then the message data may simply be written into fields within the element instead.) Additional frames or control messages are queued for the same device by simply setting up consecutive elements in the transfer ring as required; when the last element in a ring has been assigned to a frame or message, the index of the next free element is wrapped back to the start of the ring. The transfer rings are hence used as a true ring buffer.

The receiving entity (i.e., external device on the PCI bus) is expected to retrieve messages from the appropriate transfer ring when notified to do so. Message retrieval is performed in the same order that the messages were placed on the ring. Since each element is formatted as a data descriptor structure, the receiving entity can obtain all of the information necessary to retrieve the Ethernet frame or control/status message payload by simply reading the contents of the element. After a given element has been read by the receiving entity, it is expected to notify the ELAN 1x100 that the element is now available to hold another Ethernet frame or control/status message.

The actual notification process whereby the ELAN 1x100 signals a remote device that one or more elements contain information about Ethernet frames or messages that are ready for transfer is supported by special communication hardware described later. Similarly, the notification of retrieved elements by remote devices to the ELAN 1x100 is also performed with hardware support described in a later section. The ELAN 1x100 also implements hardware that allows the Switch Processor to efficiently queue frames and messages to multiple transfer rings, for use during broadcasts and multicasts.

Note that a unicast message or frame (i.e., information that is directed towards only one external device on the PCI expansion bus by the ELAN 1x100) results in only one data descriptor being allocated and formatted on a single transfer ring. A multicast message or frame, however, requires multiple elements to be set up in as many transfer rings as there are recipients of the message.

The sizes of the various transfer rings determine the maximum number of frames or messages that can be queued at any one time for the corresponding devices. The selection of these sizes therefore has a substantial impact on the efficiency and performance of the system. The number of elements within each transfer rign may range from 32 to 65,536, in powers of 2 (representing actual memory sizes of 512 to 1,048,576 bytes). Criteria for selecting the sizes of the transfer rings are given later in this section.

None of the elements in any transfer ring is cached by the Switch Processor, and hence they can be read safely at any time over the PCI bus.

Local Port Descriptor and Port Descriptor Counters

A data structure, called the *Local Port Descriptor,* is associated with the MAC interface within the ELAN 1x100. The local port descriptor contains various control/status and frame statistics fields, and is updated whenever a frame is received or transmitted by that MAC interface.

Associated with the local port descriptor is another structure, referred to as the *Port Descriptor Counter Structure*, that contains a set of frame error statistics fields which track various error counts updated on an infrequent basis during normal operation. Both of these structures are under the sole control of the Switch Processor, and never modified or read by the rest of the ELAN 1x100 hardware.

Both the local port descriptor and the port descriptor counter structure are located in the Switch Processor operating environment, and are indicated in the previously supplied memory map. Note that both structures are completely under the control of the Switch Processor firmware, and are never read or modified by the ELAN 1x100 hardware.

Local Port Descriptor Structure

The local port descriptor structure occupies 64 bytes, and is formatted as follows:

31 24	23 16	15 8	7 0	Byte Offset
NumCollide	reserved			0
0x00	reserved			4
MulticastMask		MaxBuffers		8
PortFlags	reserved	BackoffMask		12
TXFrames				16
TXOctets				20
TXFirstDefer				24
reserved				28
RXUcstFrames				32
RXValidOctets				36
RXFrames64				40
RXFrames65-127				44
RXFrames128-255				48
RXFrames256-511				52
RXFrames512-1023				56
RXFrames1024-1518				60

NumCollide:

8-bit count of collisions experienced so far while attempting to transmit the frame at the head of the transmit FIFO cleared to zero before starting the first transmission attempt for each frame. Note that this field is only used in half-duplex operation.

MaxBuffers:

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

MulticastMask:

16-bit mask used to restrict broadcasts and multicasts received on the ELAN 1x100 MAC channel to a subset of the external devices in the system (that is, to restrict frame forwarding over the PCI expansion bus). The lower 8 bits of this mask are always zero.

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 control flags used for controlling switching of frames received over the ELAN 1x100 MAC port, or transmitted over the port.

TXFrames:

32-bit count of frames (unicast, multicast or broadcast) successfully transmitted on this port without errors.

TXOctets:

32-bit count of total number of bytes successfully transmitted on this port without errors.

TXFirstDefer:

32-bit count of total number of frames that were forced to defer to incoming (receive) traffic during their first transmission attempt. This counter will always be zero if the MAC port is configured for full-duplex operation.

RXUcstFrames:

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

RXValidOctets:

32-bit count of non-errored bytes received on this port.

RXFrames64:

32-bit count of frames received on this port, whether errored or non-errored, that were 64 bytes in size.

RXFrames65-127:

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

RXFrames128-255:

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

RXFrames256-511:

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

RXFrames512-1023:

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

RXFrames1024-1518:

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

The NumCollide field is used to track the number of consecutive collisions encountered when attempting to transmit a given frame in half-duplex mode. 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 NumCollide field will always be zero if the MAC port is in full-duplex mode.

MaxBuffers is expected to be initialized (during the system boot-up process) with the maximum number of packet buffers that may be allocated to hold Ethernet frames by the ELAN 1x100 during reception. This field is decremented by the number of packet buffers used to store 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. MaxBuffers is incremented whenever an Ethernet frame that was received on this MAC port has been completely transferred to all external ELAN devices over the PCI expansion bus, and its packet buffers are freed. If backpressure (for a half-duplex link) or PAUSE flow control (for a full-duplex link) is enabled, then a globally-configurable threshold is defined; if the value of the MaxBuffers field falls below this threshold, then the MAC channel will start the appropriate flow control mechanism, which will consist of either channel jamming in half-duplex mode, or transmission of a PAUSE control frame to the upstream station in full-duplex mode.

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 8 LSBs of this mask are always set to zero on the ELAN 1x100, while the upper 8 bits of the mask correspond on a one-to-one basis to the eight possible devices in the system. A device is permitted to be part of a multicast if its corresponding bit in the MulticastMask field is set to a '1'. (Note that the bit in the MulticastMask corresponding to the ELAN 1x100 device itself is always set to zero at initialization time.) 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 to remove loops in the broadcast topology.

A 16-bit BackoffMask field is provided to allow the range of collision backoff values to be adjusted via management access. The BackoffMask field contents are logically ANDed with the standard pseudorandom 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	Special	Type	reserved	TXBlkd	RXBlkd	BackPRn

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.

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.

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.

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

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.

The Switch Processor caches the local port descriptor during normal operation. The in-memory copy of the local port descriptor, therefore, is likely to be 'stale' (i.e., outdated by information in the Switch Processor's data cache). The local port descriptor should preferably be read, if desired, by using the message interface maintained by the Switch Processor. If the contents of the local port descriptor 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, which is described later.

Port Descriptor Counter Structure

The 96-byte port descriptor counter structure is considered to be an auxiliary data structure to the local port descriptor. This structure 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
DribbleFrames	28
OVFErrors	32
OversizeErrors	36
JabberErrors	40
ShortErrors	44
FragErrors	48
LimitErrors	52
TXErrors	56
reserved	60
RXBcstFrames	64
RXBcstOctets	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. Only valid in half-duplex mode.

MultCollide:

32-bit count of multiple collisions encountered when attempting to transmit frames out this MAC port. Only valid in half-duplex mode.

LateCollide:

32-bit count of late collisions encountered when attempting to transmit frames out this MAC port. Only valid in half-duplex mode.

CollideAbort:

32-bit count of transmit frames that were discarded due to excessive collisions for this MAC port. Only valid in half-duplex mode.

RXErrorOctets:

32-bit count of total number of octets for received frames that exhibited an error: this includes frames having alignment, CRC, FIFO overflow, oversize, jabber, short, collision fragment or buffer limit errors.

AlignErrors:

32-bit count of frames received on this MAC port with alignment errors. A frame is deemed to have an alignment error if it is of valid length, and had an incorrect CRC together with an odd number of framed data nibbles received on the MII interface. In this context, "valid length" means an actual frame length, excluding the preamble and SFD, between 64 and 1518 octets, inclusive.

CRCErrors:

32-bit count of frames received on this MAC port with CRC errors. A CRC error is declared if a received frame is of valid length, had an even number of framed data nibbles as received on the MII interface, and failed the FCS check.

DribbleFrames:

32-bit count of the number of frames that were received with dribble errors. A frame is considered to have a dribble error if it has an odd number of framed data nibbles as received on the MII interface, but was otherwise non-errored (including the FCS check). Note that frames with dribble errors will still be accepted and forwarded by the switching firmware, and will hence cause the RXValidOctets counter to be incremented.

OVFErrors:

32-bit count of frames received on this MAC port that were otherwise valid, but dropped due to receive FIFO overflow.

OversizeErrors:

32-bit count of frames received on this MAC port that passed the FCS check but exceeded the preset maximum frame length limit (nominally1518 octets).

JabberErrors:

32-bit count of frames received on this MAC port that failed the FCS check and was longer than the preset maximum frame length.

ShortErrors:

32-bit count of frames received on this MAC port that passed the FCS check but were below the preset minimum frame length (nominally 64 octets).

FragErrors:

32-bit count of frames received on this MAC port that failed the FCS check and were below the preset minimum frame length. These frames are typically collision fragments.

LimitErrors:

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

LengthErrors:

32-bit count of frames received on this MAC port that failed the IEEE 802.3 length check (i.e., the Length field within the frame was between 45 and 1499 inclusive, but did not match the actual frame length).

TXErrors:

32-bit count of frames that were dropped while being transmitted out this MAC port. This counter is always zero during full-duplex operation; in half-duplex mode, it is incremented whenever a valid frame is dropped from the transmit FIFO due to late collision, excessive collisions, or excessive carrier deference.

RxBcstFrames:

32-bit count of valid broadcast frames received on this port (that is, the destination address of the received frame is all-ones, corresponding to the IEEE 802.3 broadcast address).

RXBcastOctets:

32-bit count of octets in valid broadcast frames received on this port.

RxFloodFrames:

32-bit count of valid frames received on this port that were flooded to all ports on the spanning tree. A frame is flooded if the 48-bit destination address is not an IEEE 802.3 Group/Functional address (i.e., the LSB of the address is zero, indicating a unicast destination) and an entry for the destination address is not present in the address table.

RXFloodOctets:
32-bit count of octets in valid flood frames received on this port.

RxMcstFrames:
32-bit count of valid multicast frames received on this port (i.e., the 48-bit destination addresses of the counted frames are IEEE 802.3 Group/Functional addresses, but not the broadcast address of all-ones).

RXMcastOctets:
32-bit count of octets in valid multicast frames received on this port.

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

The port descriptor counter structure occupies the memory locations described in the address map previously given. As the port descriptor counter structure is not cached, it may be read at any time via PCI bus accesses.

Expansion Port Descriptor Table

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 1x100 devices (or compatible devices, such as the ELAN 8x10) in the system. The expansion port descriptor table is used in conjunction with the transfer rings by the switching firmware to transfer frames and messages to other devices within the system.

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 ELAN 1x100 itself), only seven of the eight expansion port descriptors are used, and the eighth descriptor is left untouched. Each expansion port descriptor is hence effectively 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.

Note that the use of the expansion port descriptors is not limited to communicating with ELAN 1x100 devices. Any device that implements the same transfer protocol can be interfaced to the ELAN 1x100 PCI bus interface and assigned an expansion port descriptor (with the associated frame transfer queue), and the ELAN 1x100 device will transfer frames to this device without any special considerations. A detailed description of the frame transfer protocol, and the use of the expansion port descriptor and transfer

ring structures to sequence the transfer of data across the PCI bus, is given in a subsequent section.

An expansion port descriptor occupies 16 bytes, and is formatted as follows:

31 24	23 16	15 8	7 0	Byte Offset
ChipNum	RingBase			0
CtrMask	reserved			4
reserved		reserved		8
RemDD				12

RingBase:

24-bit pointer to first data descriptor in the transfer ring corresponding to this expansion port descriptor. This field is initialized with the base address fo the appropriate transfer ring during system boot, and remains unchanged thereafter.

ChipNum:

8-bit index assigned to remote chip corresponding to this expansion port descriptor. This field is set up at initialization time to contain a '0' for the first expansion port descriptor (i.e., with index 0), a '1' for the second expansion port descriptor, and so on.

CtrMask:

8-bit mask that must be passed to the DMA Controller increment channel hardware when attempting to increment a request or acknowledge counter in the remote device.

RemDD:

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 ChipNum and CtrMask fields are statically configured during initialization time. 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 set, and all of the other bits cleared. For example, CtrMask in expansion port descriptor zero (the first descriptor) will be set to 00000001 binary; that in the second expansion port descriptor will be set to 00000010 binary; and so on. The contents of CtrMask are loaded into the DMA Controller increment channel parameter registers when the switching firmware desires

to increment either the request or acknowledge counter in the remote device during expansion port frame transfers.

The RemDD field holds the 32-bit PCI address of the data descriptor at the head of the transfer queue or ring assigned to this ELAN 1x100 in the remote device. This field is set up at system initialization time: the uppermost 8 bits are set to the 8 MSBs of the PCI base address of the remote device, while the lower 24 bits are set up to be equal to the value loaded into the RingBase field of the expansion port descriptor in the remote device (i.e., the address of the first data descriptor in the remote device's transfer queue or ring). After frame transfer from the remote device begins, the uppermost 8 bits of the RemDD 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 RemDD field.

As already noted, the expansion port descriptor corresponding to the index assigned to the ELAN 1x100 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.

Address Hash Table

The ELAN 1x100 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, i.e., not

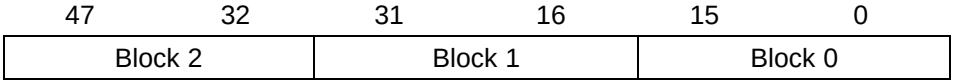
previously seen or set up destinations must be flooded to all ports, or those determined using the standard IEEE 802.1d spanning tree.) The hash table also serves to hold per-host statistics information.

Note that when multiple ELAN 1x100 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 1x100 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 8192 pointers; a set of 64-byte data structures called *hash buckets*, that hold the actual per-MAC information for locally reachable hosts; and a set of 16-byte *forwarding tags* that hold abbreviated per-MAC information for hosts reachable via external devices.

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 pointer array: if there are 8,192 entries in the array, 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 computed 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.

On the ELAN 1x100, hash lookups are initiated at two different points in the frame switching process. Firstly, the source and destination MAC addresses of all frames received on the MAC port are looked up in the address table automatically by the DMA Controller; this results in two hash lookup operations being performed for each incoming frame. Secondly, another address table search is done if a frame was copied over the PCI expansion bus and the Learn flag bit is set in the Flags field of the data descriptor associated with the frame. This hash lookup is done on the source MAC address within the frame.[1]

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 described by the address map given previously. Initially, all the pointers in the array are set to zero (NULL), indicating that no MAC addresses are present in the address table. As MAC addresses are learned (or statically created), the hash buckets or forwarding tags corresponding to these addresses are inserted into the address 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

[1] The setting of the Learn bit is an indication by the remote device that it has determined that the source MAC address of the frame under question was not seen before by the system, and needs to be entered into the address table; a hash lookup on the source MAC address is done first as a precaution, to ensure that the MAC address does not already exist in the table.

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 1x100 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:

Error! No index entries found.

24	23	16	15	8	7	0	Byte Offset
reserved			NextHB				0
MACAddr[31:16]					MACAddr[47:32]		4
MulticastMask					MACAddr[15:0]		8
HBFlags			LPDPtr				12
TXFrames							16
TXOctets							20
RXErrorFrames							24
RXMcstFrames							28
RXFloodFrames							32
RXBcstFrames							36
RXUcstFrames							40
RXOctets							44
ReceiveTimeStamp[31:0]							48
CreateTimeStamp[31:0]							52
ReceiveTimeStamp[47:32]					CreateTimeStamp[47:32]		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 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 1x100 port associated with this MAC address (always the same for all hash buckets).

HBFlags:

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

TXFrames:

32-bit count of unicast 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 frames (i.e., frames directed to IEEE 802.3 Group/Functional addresses, excluding the broadcast address) received on this port.

RXFloodFrames:

32-bit count of valid flood frames received on this port. A frame is considered to be flooded if it is a unicast, but is directed to all ports in the system because the ELAN 1x100 does not have an entry for the destination MAC address in its address table.

RXBcstFrames:

32-bit count of valid broadcast frames received on this port. Broadcast frames are classified as frames with a destination MAC address of all-ones.

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.

ReceiveTimeStamp[31:0]:

Lower 32 bits of timestamp indicating when a frame was last received from this MAC address. These bits are obtained directly from the CLOCK register pair within the Switch Processor.

ReceiveTimeStamp[47:32]:

Upper 16 bits of timestamp indicating when a frame was last received from this MAC address. These bits are incremented under Switch Processor firmware control whenever a carry out of ReceiveTimeStamp[31:0] is detected.

CreateTimeStamp[31:0]:

Lower 32 bits of timestamp indicating when this hash bucket was originally created. These bits are obtained directly from the CLOCK register pair within the Switch Processor.

CreateTimeStamp[47:32]:

Upper 16 bits of timestamp indicating when this hash bucket was originally created. These bits are set by the Switch Processor firmware using an internally extended version of the CLOCK register pair.

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 frame 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 MACAddr field is set to an IEEE Group/Functional address). The lower 8 bits of this field are always zero, and the upper 8 bits of the mask correspond to the eight possible devices in the system. A device is permitted to be part of a multicast if its corresponding bit in the MulticastMask field is set to a '1'. (Note that the bit in the MulticastMask corresponding to the ELAN 1x100 device itself is always set to zero at initialization time.) The MulticastMask field is logically ANDed with the global restriction mask managed by the spanning tree protocol entity, as well as the MulticastMask field in the local port descriptor structure, to implement multicasts on a per-MAC-address as well as per-port 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 structure. It is provided for cross-compatibility purposes with the ELAN 8x10 device.

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:

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

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 local MAC channel
10	Invalid for hash buckets, reserved for forwarding tags
11	Broadcast MAC address reachable via 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.

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 1x100.

The ReceiveTimeStamp field holds a 48-bit timestamp that records the absolute time at which 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 3258 days. 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 ReceiveTimeStamp field whenever a valid Ethernet frame is received and the hash lookup engine resolves the source MAC address in the frame to the given hash bucket. The upper 16 bits of the timestamp (bits 47:32) are written with the software maintained real time clock. The 48-bit timestamp value is generated by concatenating the 32-bit CLOCK real-time clock register in the

Switch Processor with a firmware-maintained 16-bit variable that holds the carries out of the CLOCK register.

The CreateTimeStamp field holds a 48-bit timestamp indicating when the hash bucket was created and inserted into the address table. This value is generated in a similar manner to the ReceiveTimeStamp field, using the 32-bit hardware CLOCK register coupled with a firmware count of carries to extend the CLOCK register value to 48 bits. It is principally used as part of the RMON MIB.

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 is created whenever it is necessary to record that a particular MAC address is reachable by an external device (such as another ELAN 1x100), and frames directed to this MAC address should be directed to that device via the PCI expansion port. Forwarding tags have the following format:

Error! No index entries found.

24	23	16	15	8	7	0	Byte Offset
ChipNum		NextFT					0
MACAddr[31:16]			MACAddr[47:32]				4
Reserved			MACAddr[15:0]				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.

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 frame 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 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 1x100 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 1x100 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 1x100 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 1x100. 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.

Free Pools

During normal frame switching operations, the ELAN 1x100 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 four free memory pools: a **packet buffer pool**, a **message descriptor pool**, a **hash bucket pool** and a **forwarding tag 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 Controller; an internal hardware register in the DMA Controller is set up at initialization time to point to the first free buffer in the linked-list, and the DMA Controller allocates blocks from the list as required to hold frames received on the MAC port. Deallocation of the packet buffers to the free pool is performed by the Switch Processor firmware after frames have been completely transferred across the PCI bus (that is, the frame has been acknowledged to be read by all remote devices to which the frame was to be forwarded), or been dropped for assorted reasons.

The **free message descriptor pool** is used for supplying message descriptors, used for tracking management frames or control/status messages. It is created from a linked-list of 16-byte fixed-size memory blocks, and is maintained entirely by the Switch Processor firmware. Blocks are allocated from this pool as required when a message descriptor must be created to hold information for messages received from external devices over the PCI expansion bus, or for management frames that have been received either on the MAC port or from a device on the PCI bus.

The **hash bucket pool** is composed of a linked-list of 64-byte blocks that are intended to be used to create hash buckets 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 deallocated 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.

The **forwarding tag pool** is constructed from a linked-list of 16-byte blocks, and supplies forwarding tags that are required during the learning of MAC addresses reachable by remote devices. Forwarding tags are allocated by the Switch Processor firmware during the learning process, and deallocated (again by the Switch Processor firmware) when they are removed from the address table. Removal of forwarding tags takes place only after their parent source hash buckets (on the remote devices) are removed.

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.

Pool-Empty Handling

The parameters supplied during the initialization process (such as the number of addresses to be learned, the packet buffers that can be consumed, etc.) should be selected for the system such that none of the four free pools will be permitted to become empty during normal operation. This is not, however, always guaranteed. The ELAN 1x100 switching firmware thus checks for pool-empty conditions and performs recovery processing as required to deal with the situation.

It is illegal for the free packet buffer pool to become empty at any point, as the DMA Controller hardware always requires that at least one free packet buffer be present in this linked-list.[1] The DMA Controller hardware itself will not violate this rule; if at any time, the DMA Controller determines that there is only one packet buffer left in the free packet buffer pool, then that packet buffer will not be allocated. (If a frame is being received, or is subsequently received, then a receive overflow error will be declared.) If, however, the free packet buffer pool becomes completely empty (e.g., due to some system failure), then the DMA Controller will notify the Switch Processor, which will in turn declare a fatal system error and attempt to initiate a hardware reset to recover from this condition.

The free message descriptor pool, hash bucket pool, and forwarding tag pool are all controlled entirely by the Switch Processor firmware. The recovery process for one or more of these pools becoming empty during operation is less drastic than a hardware reset; in most cases, normal system operation can continue, with some degradation in system response.

If the firmware detects that the free message descriptor pool is empty, then it will not accept any more messages or management frames from other devices, and will discard

[1] This is due to the mechanism used by the DMA Controller to allocate and deallocate packet buffers from the free pool.

any frames that are received but need to be handled specially. It will not, however, cause the remote devices on the PCI bus to discard messages that have been queued for this ELAN 1x100; instead, it will refuse to read any more data from the devices in question.

If the free hash bucket pool is emptied (i.e., the maximum number of MAC addresses have been learned from frames arriving over the local MAC port), then the switching firmware will simply cease to learn any more MAC addresses from the local port. Frames arriving with new source MAC addresses will be forwarded as usual; however, frames that are subsequently directed towards these new (and unlearned) MAC addresses will be flooded, as no entries will exist for them in the address table. This effect is comparatively benign, as no frames are lost, and only a small performance degradation is encountered.

If the free forwarding tag pool is exhausted, then a similar process is used to handle this condition. In this case, however, the ELAN 1x100 will refuse to learn MAC addresses contained within frames arriving over the PCI expansion port (i.e., with the Learn bit in the data descriptor Flags field set). Again, frames that are subsequently directed to these unlearned addresses will simply be converted into floods.

Spanning Tree Support Data Structures

Upon start-up, if spanning tree is enabled, the ELAN 1x100 creates a set of data structures required for implementing the IEEE 802.1d spanning tree algorithm. (In a system comprised of multiple ELAN 1x100 and ELAN 8x10 devices, 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 1x100 to limit packet broadcasts in response to the results of the spanning tree algorithm. This mask consists of sixteen bits: eight bits correspond to a port mask and eight bits correspond to a device mask. The ELAN 1x100 has only one physical port, and hence the lower 8 bits of the broadcast restriction mask are set to zero. The remaining eight bits are assigned to the eight possible external 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	0	0	0	0	0	0	0	0

The eight device-related bits are denoted as D0 through D7. Note that the device-related bit corresponding to the given ELAN 1x100 device itself must always be zero, as it is not permissible for an ELAN 1x100 device to send broadcasts to itself across the PCI bus. In addition, bits corresponding to non-existent devices 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 1x100 inspects the broadcast restriction mask to determine the permissible destinations to which the frame 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 1x100 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 is 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, the transfer rings, and the free pools (i.e., the free packet buffer pool, the free data descriptor pool, the free hash bucket pool and the free 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, each	Quantity Required	Suggested Quantity
Operating Environment	43 kB	Single block of memory starting at address 0x000000	1
Packet Buffers	80 bytes	Per-port buffer capacity (equals number of packet buffers multiplied by the size of the data payload within a packet buffer)	About 50% of available memory
Transfer Rings	16 bytes (per element)	Number of frames or messages that can be queued for device corresponding to each transfer ring, multiplied by number of devices	One per packet buffer in system
Message Descriptors	16 bytes	Number of control/status messages and messages that can be accepted at a time	512
Hash Buckets	64 bytes	Number of MAC addresses that can be learned from frames received on the local MAC port	About 25% or avail. mem.
Forwarding Tags	16 bytes	Number of MAC addresses that can be learned from frames received over the PCI bus	Equal to hash buckets
Boot Image Copy	32 kB	Temporary use; held reserved for simplicity	1

The computation of the number of elements (data descriptors) required for each of the transfer rings used to queue frames destined for external devices is somewhat difficult. The upper bound on this number, obviously, is the number of packet buffers that are present in the free pool; if it is assumed that the number of messages that do not

require any packet buffers is negligible compared with the number of frames and messages requiring packet buffers, then clearly the maximum number of entries that can be made on any one transfer ring is limited to the number of packet buffers that can be queued, which in turn cannot exceed the number of packet buffers in the local memory. This assumes the worst case of one packet buffer per frame; if larger frames are being queued, the number of elements used in the transfer rings drops proportionally.

In order to guarantee that under no circumstances would the ELAN 1x100 have to drop a frame due to the lack of queueing resources, therefore, the number of elements in each of the transfer rings should be equal to the number of packet buffers. This is especially significant in systems where flow-control is enabled; if the local port buffer limit does not reach the flow-control threshold before the transfer rings run out of elements and frames begin to be dropped, it is likely that flow-control will never be initiated but continuous frame loss will take place.

In addition, the number of elements in each transfer ring should be a power of 2. The total number of rings required is one less than the number of devices in the system; thus a system with 8 devices would require 7 rings to be set up.

As an example, consider the memory requirements for an ELAN 1x100 in a typical system having the following parameters:

- At least 160 kB of frame buffer storage for the local MAC port
- An address table capacity of up to 2048 MAC addresses
- A total of four devices in the system (e.g., two ELAN 1x100 chips coupled with two ELAN 8x10 chips)

Assuming that the recommended values for the various types of memory regions are followed, the parameters for such a system would be:

Structure	Qty Needed	Storage
Operating Environment	1	43 kB
Packet Buffers	2048	160 kB
Transfer Rings (3)	2048/ring	96 kB
Message Descriptors	512	8 kB
Hash Buckets	2048	128 kB
Forwarding Tags	2048	32 kB
Boot Image Copy Area	1	32 kB
	TOTAL	499 kB

This amount of memory fits well into a base configuration with 512 kB of SRAM. An example memory map is as follows (note that addresses increase downwards):

The small amount of wasted space at the top of the address map is not significant. 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 1x100.

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.

Frame Switching Process

If a frame received on a physical port of a ELAN 1x100 device is addressed to a destination host connected to a MAC port of remote device (such as an ELAN 8x10 or ELAN 1x100), then the frame will be passed to the external device via the PCI expansion bus interface. The frame will also be passed to one or more external devices if it is a broadcast, multicast or flood (i.e., the destination MAC address is unknown). Otherwise, the frame will be filtered.

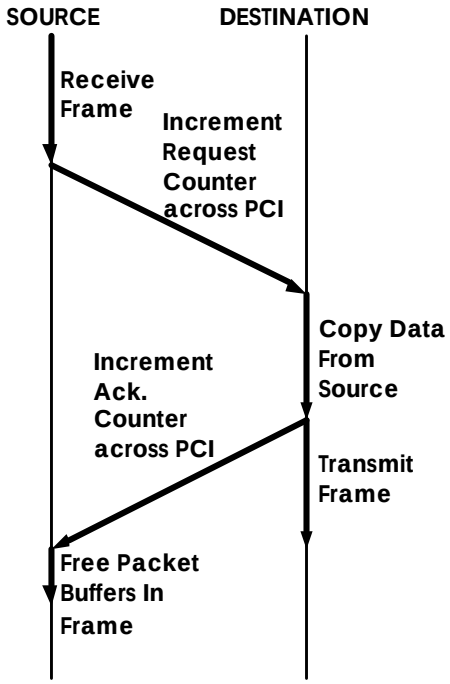
Frame switching on the ELAN 1x100 is carried out in two stages: the source ELAN 1x100 receiving the frame appends the frame to the transfer ring that it maintains for the destination device (and signals the destination device that another frame is available); the destination device is then expected to copy the data from the source ELAN 1x100 to its own memory space, and notify the source ELAN 1x100 when the transfer is complete (i.e., acknowledge the transfer). The destination device is thereafter 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 port or ports within the destination devices, without any further use of the PCI bus. A bandwidth multiplication effect is thus obtained when the destination device has multiple physical MAC ports (such as the ELAN 8x10 chip), as a single copy of the frame over the PCI bus can result in multiple transmissions within the destination device.

Request and Acknowledge Counters and Protocol

The ELAN 1x100 (and other devices that expect to communicate with the ELAN 1x100) implements a set of hardware counters that assist in the signalling 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 1x100 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 1x100 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 ELAN 1x100 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 (via a PCI memory read command) 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.

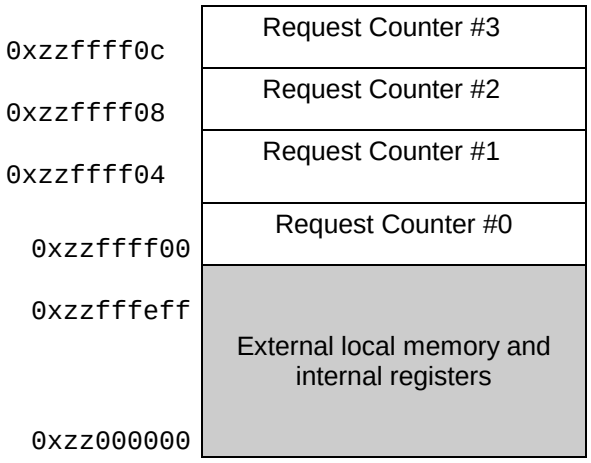
When the source device detects that the acknowledge counter has been incremented, it knows that the destination device has successfully copied the frame data out of the source buffers, across the PCI bus, and into local buffers at the destination. The source

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



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 hex from the base address assigned to the device in PCI address space; the eight acknowledge counters occupy locations from an offset of 0xffff20 to 0xffff3c hex. The address of acknowledge counter #0 may thus be obtained by adding 0x20 hex (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 (signalling 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.

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 1x100 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 external device, the 32-bit PCI address from which the data transfer must be performed is defined by the 32-bit RemDD 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 system is set by the number of request and acknowledge counters that are implemented in the ELAN 1x100 and 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 device.

Hardware Support for Transfer Protocol

In addition to the request and acknowledge counters used to set up the transfer handshake between two ELAN switch devices. the ELAN 1x100 also contains special hardware that increases the efficiency of the switching firmware when initiating and controlling this transfer. This hardware is built into the DMA Controller increment channel logic.

When a frame is to be forwarded to a remote ELAN switch device, the Switch Processor in the ELAN 1x100 does not actually write information into the physical memory holding the transfer ring corresponding to the remote device. Instead, it writes the information for the various fields of the data descriptor corresponding to the frame or message into a special set of registers in the DMA Controller, and notifies the DMA Controller of the target transfer queue to update by copying the ChipMask field from the corresponding expansion port descriptor into another special register. Once this is done, the involvement of the switching firmware in the transfer notification has ended.

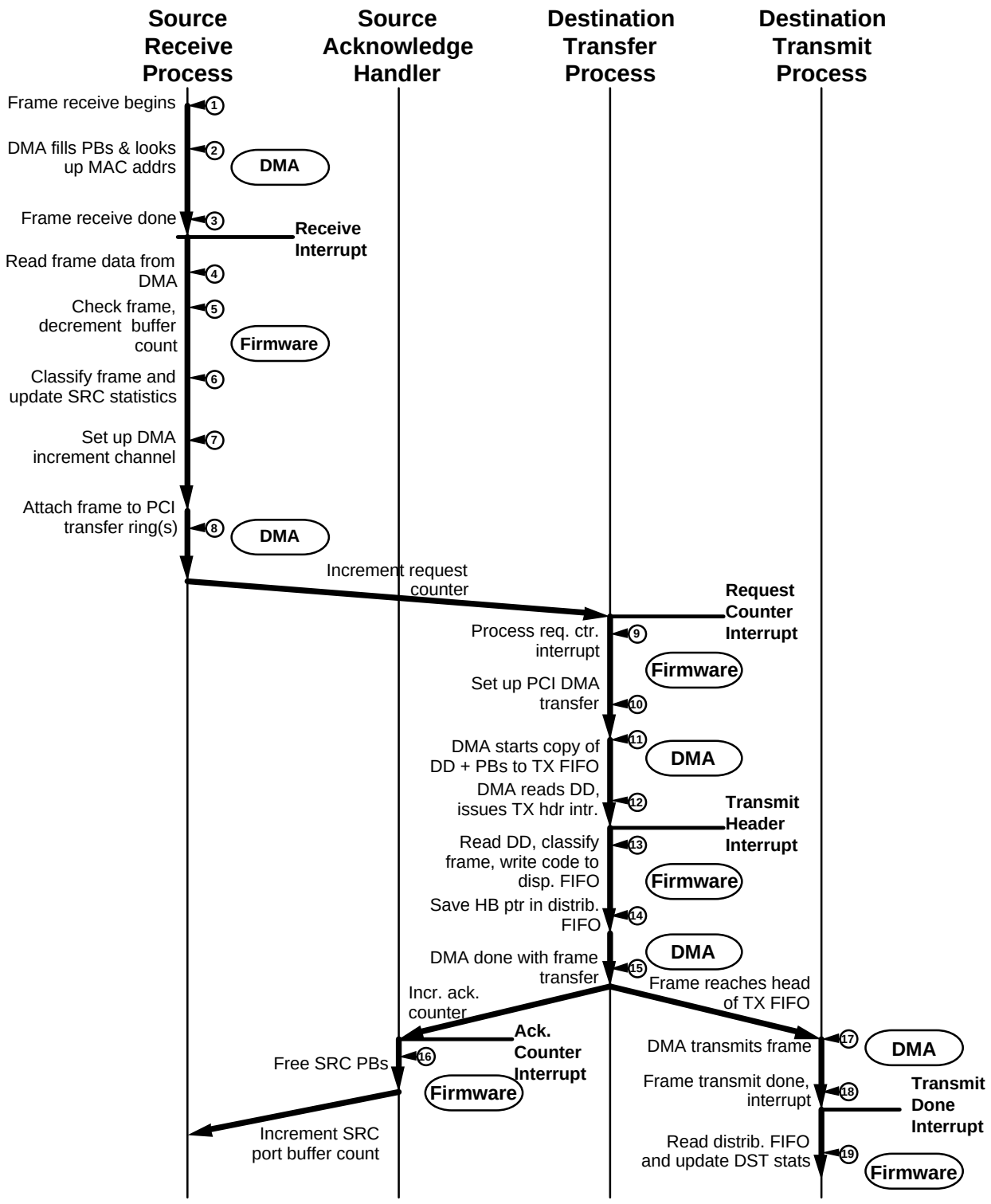
The DMA Controller will then determine the next free element in the specified transfer queue to use and write the information supplied by the Switch Processor into the element to create a data descriptor in the correct format. It will then perform a write operation across the PCI bus to increment the proper request counter in the specified device. This will complete the request portion of the handshake; the remainder of the handshake proceeds as previously described.

To facilitate the updating of the transfer rings in the proper sequence, the DMA Controller maintains hardware pointers to the first and last element in each transfer ring. When an element is allocated and formatted into a data descriptor, the write pointer associated with that transfer ring is incremented by 16 to point to the next element; when an acknowledge has been received from the remote device after successfully completing the data transfer, the DMA Controller will increment the read pointer for that transfer ring, again by 16. The hardware read and write pointers are used by the DMA Controller to ensure that unread elements in the transfer rings are never overwritten.

Multicast and broadcast frame queueing is supported by an additional feature. When a multicast or broadcast frame is to be queued on the transfer rings, the Switch Processor firmware writes the required contents of the data descriptors to be created on the transfer rings to DMA Controller registers (as before). It then writes a bitmask to a special register; the bitmask contains a '1' bit for each of the devices to which the multicast or broadcast is directed. (The bitmask is taken directly from the upper 8 bits of the MulticastMask fields.) The DMA Controller will then update all of the transfer rings as indicated by the bitmask with the proper information, and will increment the request counters in all of the target devices.

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 1x100 to a MAC port on a remote ELAN switch device via the expansion bus. (The same general sequence holds even if the destination device is not an ELAN 1x100 or ELAN 8x10 chip, as all devices that expect to communicate with the ELAN 1x100 must follow the same protocol.) Note that if a broadcast frame is sent to two or external 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 switch device.


Source Receive Process
Source Acknowledge Handler
Destination Transfer Process
Destination Transmit Process
Frame receive begins
1
DMA fills PBs & looks up MAC addrs
2
DMA
Frame receive done
3
Receive Interrupt
Read frame data from DMA
4
Check frame, decrement buffer count
5
Firmware
Classify frame and update SRC statistics
6
Set up DMA increment channel
7
Attach frame to PCI transfer ring(s)
8
DMA
Increment request counter
Request Counter Interrupt
Process req. ctr. interrupt
9
Firmware
Set up PCI DMA transfer
10
DMA starts copy of DD + PBs to TX FIFO
11
DMA
DMA reads DD, issues TX hdr intr.
12
Transmit Header Interrupt
Read DD, classify frame, write code to disp. FIFO
13
Firmware
Save HB ptr in distrib. FIFO
14
DMA done with frame transfer
15
DMA
Frame reaches head of TX FIFO
Incr. ack. counter
Ack. Counter Interrupt
Free SRC PBs
16
Firmware
Increment SRC port buffer count
DMA transmits frame
17
DMA
Frame transmit done, interrupt
18
Transmit Done Interrupt
Read distrib. FIFO and update DST stats
19
Firmware

1. The source ELAN 1x100 device in the system begins receiving an Ethernet frame on its MAC channel; the frame is intended for a destination entity connected to a port on another ELAN 1x100 device (the destination device). The ELAN 1x100 MAC channel converts and places the incoming data into the MAC interface receive FIFO 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 on the MII bus, the DMA Controller flushes all frame data from the MAC receive FIFO to the packet buffers and notifies the Switch Processor of the received frame via an interrupt, referred to as the *receive frame interrupt*.. The DMA receive channel will not start reading frame data for another Ethernet frame until the switch processor re-enables it; this is to prevent the next receive frame from overwriting any registers associated with the prior frame.

4. The Switch Processor executes the receive frame interrupt service routine in the switching firmware in response to this interrupt. The first action of the service routine is to read out the contents of the appropriate DMA Controller receive channel registers, and then re-enable the DMA channel to allow it to receive more frames. The receive interrupt service firmware then checks the status code returned by the DMA to determine whether the frame is errored, and performs the necessary processing as indicated. Note that the MAC channel hardware can begin receiving the next frame into its 64-byte FIFO buffer even if the switching firmware has not yet re-enabled the DMA Controller after the reception of a previous frame; a receive error will only occur if the MAC channel FIFO overflows before the DMA Controller has been re-enabled.

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. If the allocation limit has been exceeded, and flow control has been enabled, the appropriate flow control procedure will be performed, depending on whether the channel is operating in full-duplex or half-duplex mode. If flow control has not been enabled, then the frame will simply be dropped (and statistics updated) when the allocation limit is exceeded. If the frame is invalid, then the appropriate statistics are updated and the frame is simply dropped.

6. The firmware then proceeds to check the source hash bucket pointer returned by the DMA Controller 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; if they are, the frame will be dropped, as it pertains to a destination on the same segment as the source.

   The destination address table entry is also inspected; for a forwarded frame, this must be a forwarding tag indicating a remote device as the target of the frame, or a hash bucket indicating a multicast, or will be non-existent (in which case the frame will be flooded, and must be sent across the PCI bus to all 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.

   - 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.

7. The firmware now inspects the status of the DMA Controller increment channel, containing the hardware described in the previous section that supports PCI data transfers. If the channel is not busy, then the information required to create a data descriptor is written into the control registers in the increment channel, together with a bitmask identifying the remote device(s) to which the frame is to be forwarded; the DMA Controller increment channel is then enabled.

8. The DMA Controller then writes out the data descriptor to the specified transfer ring(s) for each target device indicated by the bitmask in the previous step. After updating the transfer ring(s), the DMA Controller increments the appropriate request counter in the destination ELAN switch device via a PCI memory write command to the address of the request counter. The data written to the counter is not significant, and is specified by a pair of internal registers set up at initialization time.

9. The write to the request counter in the destination ELAN switch device automatically causes the request counter in the destination (i.e. remote) device to increment. This request counter increment causes, in turn, a *request counter 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 request counter interrupt service routine firmware that is intended to handle these frame transfer requests.

   To reduce the amount of overhead involved in processing the request counter interrupt, the request counter hardware in the ELAN 1x100 implements an interlock that prevents the request counter interrupt from being asserted if the DMA Controller transmit channel is busy. This prevents needless busy-waiting by the switching firmware, as the first action of the request counter interrupt service routine is to set up the DMA Controller transmit channel to copy the Ethernet frame across the PCI bus.

10. The request counter interrupt service routine in each remote device will then set up the associated DMA Controller transmit channel to copy the element at the head of the appropriate transfer ring from the source device to the destination device. Once the setup has been completed, the request counter interrupt service routine will decrement the request counter to remove the interrupting condition, and exit.

11. The DMA Controller transmit channel will first read the data descriptor contents out of the transfer ring, and then use the information in the data descriptor to locate the chain of packet buffers holding the Ethernet frame and copy the packet buffer chain over the PCI bus as well. The information in the copied data descriptor is placed into internal registers within the remote device for use by the Switch Processor firmware; the payloads of the packet buffers, however, are written directly into the 2048-byte internal transmit FIFO in the remote device. Note that the copy of packet buffers across the PCI bus is only performed if the transmit FIFO has space available to hold at least one maximum-length packet buffer payload (nominally 72 bytes).

12. The DMA Controller will read the data descriptor contents, as well as the entire linked-list of packet buffers (if any) over the PCI bus in on continuous operation, as described in Step 10. However, it will issue an interrupt to the Switch Processor immediately after the data descriptor as well as the 14-byte Ethernet frame header have been read. This interrupt, which is referred to as the *transmit header interrupt*, is intended to allow the Switch Processor to begin the necessary processing to determine the disposition of the frame currently being transferred well in advance of the completion of the copy into the transmit FIFO, thereby reducing or eliminating delays between transferring successive frames. The DMA Controller will continue to transfer the remainder of the frame data without stopping while the Switch Processor responds to the transmit header

interrupt and decides what must be done with the frame.

In order to permit the Switch Processor firmware to determine the nature of the frame (and thus its ultimate fate), the DMA Controller loads the entire 16-byte data descriptor contents into eight 16-bit hardware registers that are accessible to the firmware. The information of interest to the firmware include the data descriptor Flags field (which indicates whether the information being transferred represents a normal frame, a management frame, or a control/status message), the source device ID and source port ID, the hash bucket field, and so on. The FirstPB and LastPB pointers are not relevant, as they are only necessary for locating the packet buffer chain in the source device's memory space.

13. The first action of the Switch Processor when executing the transmit header interrupt service routine is to read the data descriptor associated with the frame being transferred out of the hardware registers in the DMA Controller and to inspect the descriptor Flags field. Depending on the coding of the Flags field, the transmit header interrupt service routine will perform the following actions:

    - If the frame being transferred is a normal unicast Ethernet frame, then the firmware will simply set up the frame to pass through the transmit FIFO and ultimately be transmitted in the normal manner.

    - If the frame being transferred is a normal broadcast or multicast Ethernet frame, the firmware will check the Learn bit in the Flags field of the data descriptor to determine whether the frame contains a new source MAC address. If the Learn bit is set, then the firmware will verify that the source MAC address is not already present (by checking the results of the hash lookup performed by the DMA Controller on the source MAC address). If the MAC address is not represented by an entry in the address table, it will create a forwarding tag for the MAC address, with the SrcHashBkt field of the forwarding tag being set up from the HashBkt field of the data descriptor and the 48-bit MAC address in the forwarding tag being obtained from internall hardware registers maintained by the DMA Controller.

    - If the data being transferred is a message or a management frame, this will be indicated by the Special bit in the data descriptor Flags field being set. In this case, the firmware will have to wait until the entire frame or message has been read into the transmit FIFO, and then perform a software copy of the frame or message payload into buffers in external SRAM. The firmware thus sets the DMA Controller up to continue normal processing until the frame has reached the head of the transmit FIFO (as described below), and then interrupt the Switch Processor. The service routine for the latter interrupt will be actually responsible for copying the data out of the FIFO: it will allocate the

necessary quantity of free packet buffers, plus a free message descriptor to point to these packet buffers, copy the data out of the transmit FIFO into the free packet buffers, and queue the entire message for background processing using the free message descriptor.

- This two-stage method of retrieving messages or management frames is adopted principally because messages and management frames are much rarer than normal unicast and multicast frames. For highest throughput, therefore, the normal frames should be switched with maximum efficiency, while management frames and messages can tolerate a slight loss of efficiency during processing.

- In many cases, the entire contents of a message may fit completely within the fields of a data descriptor. In this case, no packet buffers will be attached to the data descriptor; the transmit header interrupt will simply allocate a message descriptor from the free message descriptor pool and copy the contents of the data descriptor out of the hardware registers into the newly allocated message descriptor, which will be queued for background processing.

As described in detail in a previous section, the DMA Controller also contains another hardware mechanism, called the *disposition FIFO*, that improves the efficiency of the switching firmware when handling transmit frames. Each of the 16 entries in this FIFO corresponds to a frame that has been placed into the 2048-byte transmit FIFO, and indicates how that frame is to be dealt with by the DMA Controller when it reaches the head of the transmit FIFO. Three possible actions may be requested by the firmware via codes written into the disposition FIFO: **send**, **drop**, and **stop**. The expected uses of these codes are as follows:

- A **send** is written into the disposition FIFO if the frame should be sent over the MAC port when it reaches the head of the transmit FIFO. This is the normal mode of operation, used for dealing with frames that should be transmitted without special processing. This is the typical action performed when the data descriptor Flags field does not have the Special bit set, thus indicating that no special processing needs to be performed for the frame.

- A **stop** is used by the transmit header interrupt service routine when it detects that the Special bit in the data descriptor Flags field is set, and the message-specific type code indicates a control/status message, or a management frame that was received from some other port and passed to this ELAN 1x100 device for special handling. The latter situation occurs only if this ELAN 1x100 is the master device in the system, and hence responsible for processing management and

spanning tree frames. When the first byte of the frame or message reaches the head of the transmit FIFO, the DMA Controller will stop reading any more data out of the transmit FIFO, and interrupt the Switch Processor with a *stop interrupt*, indicating that the message or management frame is ready to be read out of the transmit FIFO. The Switch Processor firmware will then read the necessary data out of the transmit FIFO; when the firmware processing is done, the disposition FIFO entry may be either changed to a **drop** code to delete the frame from the transmit FIFO (in the case of a control/status message or an SNMP management PDU), or to a **send** code to transmit the frame over the MAC port in the normal manner (typically in the case of broadcast control messages such as ARP or ICMP).

- An entry in the disposition FIFO is set to a **drop** code to indicate that the frame can be simply deleted from the transmit FIFO without transmitting it over the MAC port. This is normally used in conjunction with the **stop** code; the transmit header interrupt firmware writes the **stop** code into the disposition FIFO, and the stop interrupt service routine later changes this code to **drop** after the frame or message reaches the head of the FIFO and has been read out by the Switch Processor firmware.

14. If the frame is a normal unicast, then it is necessary to update RMON host statistics pertaining to the transmission of the frame to the target host. However, it is possible that errors during transmission (late or excess collisions in half-duplex mode) can result in the transmission terminating in failure; this is not known until the transmission attempt is actually completed. To facilitate the proper update of RMON statistics, therefore, the firmware maintains a small ring buffer of entries referred to as the *distribution FIFO*. This FIFO (which is handled entirely by software) holds pointers to hash buckets in the address table of the ELAN 1x100 that correspond to the destination MAC addresses contained within the frames stored within the main transmit FIFO.

    When the transmit header interrupt service routine determines that the frame being transferred across the PCI bus is a normal unicast, it places an entry into the tail of the distribution FIFO that contains the HashBkt field of the data descriptor for the frame. When the transmission attempts for that frame have been completed, the entry for the frame (which will be at the head of the distribution FIFO) will be read out and used to update the indicated hash bucket with the proper statistics.

    At this step, therefore, the firmware places an entry into the distribution FIFO corresponding to the frame being processed.

15. When the frame data has been completely read into the transmit FIFO by the destination ELAN 1x100, the DMA Controller will automatically send an acknowledge back to the source device, notifying it that the data copy is complete and the memory resources used in the source device may be freed. This is done by initiating a single-word PCI write transaction to the proper acknowledge counter in the source device; in a manner identical to request counters, the acknowledge counter in the source device will automatically increment when a write is performed to its assigned PCI address. Once the acknowledge counter in the source has been incremented, the transfer handshake is complete; the destination and source devices will autonomously perform the necessary final processing required to transmit the frame and free the resources, respectively.

16. The acknowledge counter increment performed by the DMA Controller in the destination device to the source device in the preceding step will generate an *acknowledge counter interrupt* to the Switch Processor in the source device. The Switch Processor will then execute the acknowledge counter interrupt service routine, which will first verify that the acknowledge notification was valid (i.e., at least one frame was queued for transfer to the destination device making the notification). If the acknowledge was expected, the firmware will then remove the entry at the head of the transfer ring for the destination device, updating the corresponding expansion port descriptor device accordingly.

    If the transfer ring entry indicates a unicast frame, the packet buffer chain for the frame is immediately freed (returned to the free packet buffer pool); if a multicast or broadcast is indicated, then the reference count field (RefCount) in the last packet buffer is decremented first, and the packet buffer chain is only freed if the reference count goes to zero. If the packet buffer chain is freed, the MaxBuffers field in the original source local port descriptor is incremented by the count of buffers freed, to permit the free buffers to be re-used for future received traffic. The firmware also notifies the DMA Controller that an entry in the transmit ring has been freed; the DMA Controller adjusts its internal pointer registers to take this into account.

17. The DMA Controller constantly reads data out of the head of the transmit FIFO and passes them to the MAC logic for transmission on the medium. When the frame that was transferred in steps 11 through 15 reaches the head of the transmit FIFO, then it will check the disposition FIFO entry for that frame. If the FIFO entry indicates a **send** code, the DMA Controller will read the frame data out of the transmit FIFO and present them to the MAC logic. The MAC logic will initiate frame transmission over the medium according to the IEEE 802.3u specification, beginning with the preamble and SFD and ending with the 4-byte CRC field. The MAC logic is completely responsible for inserting any required interframe gap, counting off backoff intervals for half-duplex, and regulating the data rate; the DMA Controller simply reads data out of the transmit FIFO and

presents them to the MAC logic whenever the MAC logic is ready to accept data. This process continues until the end of the current frame (as indicated by an end-of-frame delimiter being read out of the transmit FIFO) has been reached. At this point, the DMA Controller notifies the MAC logic that no more data are available for this frame.

18. After all of the data for the frame have been transferred to the external PHY device (effectively terminating frame transmission), the DMA Controller asserts a *transmit done interrupt* to the Switch Processor. The functions of the DMA Controller in the destination device with regard to frame processing will be completed at this point.

19. In response to the transmit done interrupt, the Switch Processor invokes the transmit done interrupt service routine: the primary function of this routine is to update the per-port and per-host statistics based on the final transmit status. The per-port statistics are held in the local port descriptor data structure (as well as error counters in the port descriptor counter structure), and are updated for all transmission attempts of any type of frame. The per-host statistics reside in the hash bucket corresponding to the MAC address, and hence these are updated only for successful transmissions of unicast frames. The hash bucket to be updated is indicated by the head entry in the distribution FIFO. The count of octets transmitted for the current frame is supplied by the DMA Controller via a hardware register, and is used for updating the octet-level statistics counters for both the port and host statistics.

    It should be noted that the DMA Controller will not wait for the transmit done interrupt to be serviced before proceeding to the next frame in the transmit FIFO. If the transmit done interrupt for a given frame is not serviced before the next frame completes transmission, however, the DMA Controller will halt the transmission of more frames from the transmit FIFO until the Switch Processor executes the transmit done interrupt service routine.

If the MAC port is operating in half duplex mode, and a collision is sensed (i.e., the MII COL signal is asserted) during the transmission of a frame, the MAC logic immediately stops further data transmission for that frame and performs the standard IEEE 802.3 collision jam and backoff sequence. On the completion of the jam, a *collision interrupt* is asserted to the Switch Processor, which then executes the collision interrupt service handler to perform the remainder of the IEEE 802.3 collision functions in firmware. The sequence of operations performed by the collision interrupt handler is as follows:

1. The firmware first increments the count of collisions experienced by this frame; this count is maintained in the NumCollide field in the local port descriptor, and tracks the number of transmit attempts made for a particular frame that ended in collision. The NumCollide field is always zero prior to starting the first transmit attempt for each frame.

2. If the incremented NumCollide field is 16 (indicating that sixteen consecutive collisions occurred for the given frame), then the Ethernet frame is discarded. This is done by setting the head entry in the disposition FIFO to the **drop** code; the DMA Controller will automatically purge the frame from the transmit FIFO, and proceed to the next one. Otherwise, the head entry in the disposition FIFO is left untouched (i.e., it will be a **send** code), and another transmission attempt will be made after the backoff period times out.

3. The backoff counter reload register in the MAC logic is then set with a new random backoff value, computed according to the IEEE 802.3 truncated binary exponential backoff algorithm. Note that the backoff counter reload register is always set in advance with the *next* backoff timer to use on the next collision; thus it is loaded before beginning the first transmit attempt for a frame with the backoff value to use for the first collision; successively loaded with random values drawn from a greater and greater range until the fifteenth collision; as the sixteenth collision will cause the frame to be discarded, it is reset back to the minimum range backoff value prior to beginning the sixteenth transmission attempt. (This mechanism is employed to make the collision backoff times insensitive to the latency incurred by the Switch Processor firmware in setting up the backoff counter reload register.)

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 9 through 18 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 1x100 consults its broadcast restriction mask, set up statically at configuration time or dynamically by the spanning tree algorithm, to determine the specific external devices to which the frame may be broadcast. It then queues the same frame for transmission on all of the required expansion port transmit frame queues (using multiple data descriptors, but 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 1x100 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 1x100 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 1x100.

Address learning in a multi-device ELAN-compatible 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, if any.

Local Address Learning

Address learning within the ELAN 1x100 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 TimeStamp 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 1x100, 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. This completes the local address learning procedure.

7. If the MAC address cannot be learned by the ELAN 1x100 due to lack of resources, the Switch Processor instead uses a pointer to a default hash bucket in place of the source hash bucket pointer in the data descriptor, and does not perform the remote address learning procedure. 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 1x100 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 1x100 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 and the frame is a broadcast. 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 forwarding tag pool and formats it. It writes 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 FTFlags field of the forwarding tag with the default value.

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 1x100 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 Bucket

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 one dummy (default) hash bucket within the Switch Processor operating environment at system start-up. This hash bucket is set up to correspond to the single physical port of the ELAN 1x100. This invalid hash bucket is **not** placed into the address table, and its MAC address field is invalid. The dummy hash bucket 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 to point to the default hash bucket. 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 1x100 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 ports on different ELAN 1x100 devices) 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 ReceiveTimeStamp field in the hash bucket data structures. This field is updated with an absolute time in microseconds by the Switch Processor whenever a valid frame has been received from the corresponding source entity. The ReceiveTimeStamp field thus indicates the last time at which the source was active. The aging process periodically scans the entire address table, comparing the ReceiveTimeStamp 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 switch device, an additional process is followed to ensure that the forwarding tags on all of the other ELAN 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 1x100 is set by the MaxAge variable, and may vary between 2 and 1 million 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 1 million 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 ReceiveTimeStamp 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 receive interrupt service routine that performs the primary frame dispatching tasks is responsible for updating the ReceiveTimeStamp field in the source hash bucket whenever a frame is received from the corresponding source. The receive / 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 ReceiveTimeStamp 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.

4. 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 48-bit internal real-time clock is read and preserved as the current time against which the ReceiveTimeStamp 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 ReceiveTimeStamp 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 1x100 or other) must access its own address table, locate the proper forwarding tag using the data specified in the message, and remove and deallocate the forwarding tag.

Buffer and Queue Management

The Switch Processor implements all of the buffer and queue management in an ELAN 1x100 device, using mechanisms implemented as firmware routines. Buffer management is carried out by means of a port buffer allocation limit, plus a 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 1x100 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 1x100 device. The first mechanism attempts to control the number of receive frames that are accepted for queueing to ensure that in no case will the buffering capacity of the system be exceeded (i.e., the DMA Controller will never run out of free packet buffers). 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 overconsumption of system resources by large amounts of broadcasts or multicasts.

Port Buffer Allocation Limits

The first level of buffer allocation control uses the 16-bit MaxBuffers field in the local port descriptor structure. This field is set up at initialization time with the maximum number of (fixed-size) packet buffers that the MAC port is allowed to consume. Whenever a valid frame is received on the MAC channel, the Switch Processor firmware subtracts the number of packet buffers required to hold the frame from the MaxBuffers field in the local port descriptor. If the result is less than or equal to zero, it indicates that insufficient buffer capacity is available 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 transferred to another device across the PCI bus or processed and discarded. The permissible range of the MaxBuffers fields is between 44 and 32767, depending on the available packet buffer memory. This buffer allocation limit guarantees that the DMA Controller will not run out of buffers in the free packet buffer pool.

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 either begin jamming the medium by continuously transmitting a bit pattern in half-duplex mode, or to send a flow control frame in full-duplex mode.

Note that the MaxBuffers variable in the local port descriptor is 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 this variable 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 1x100 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 devices in the system) may have to be buffered for a long time, and will also require a large number of entries in the transfer rings in order to be transferred to all of the other ELAN switch devices. 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 1x100 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 expansion port queues), 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 FIFO Management

The ELAN 1x100 implements a mechanism to monitor the transmit FIFO to ensure that frames do not persist in the transmit FIFO for too long before being transmitted. This can happen if the medium associated with the 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 FIFO will fill up with one or more frames that will never be sent out, blocking the corresponding transfer queues in the source devices that need to pass frames out the ELAN 1x100 MAC port. 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 1x100 Switch Processor implements a two-

stage procedure to determine whether a transmit FIFO has been blocked for a long time.

The first part of the procedure is implemented by the transmit header interrupt service routine, that is invoked whenever the header of a frame has been read into the transmit FIFO. This routine writes the contents of the CLOCK register pair in the Switch Processor, which implements the real-time clock, into the LastTransmit memory variable as part of its processing. The LastTransmit variable thus records the last time that a frame was read into the transmit FIFO.

The second portion of the transmit FIFO management process is a background task that is periodically invoked when the Switch Processor is idle. This background task checks the current status of the request counters implemented by the ELAN 1x100 for the PCI transfer handshake. If any of the request counters is non-zero, it indicates that one or more frames are pending to be transmitted out the ELAN 1x100 MAC port. In this case, the background task subtracts the contents of LastTransmit variable from the current value of the CLOCK register pair; if the difference is greater than a pre-set threshold, this indicates that the transmit FIFO has been blocked for a considerable amount of time, and a system malfunction has occurred. In this case, the transmit FIFO management process will attempt to clear out the frames from the transmit FIFO (by setting their disposition FIFO entries to **drop**) to advance the transfer queues in the source devices and thereby reclaim the buffer resources.

The threshold determining the maximum time that the transmit FIFO is permitted to be blocked is a configurable parameter. The default value for this threshold is 1 second.

Broadcast Rate Limiting

It is desirable to provide some form of limit on the rate of throughput of broadcasts handled by an ELAN 1x100 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 1x100 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 1x100 device.

A background task is periodically executed every ten milliseconds by the ELAN 1x100 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 1x100 device collects a number of SNMP and RMON MIB statistics during normal operation. These statistics are maintained in the local port descriptor, the port descriptor counter structure, 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 1x100 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 1x100 device on a per-port basis are described in the following table:

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[1]
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[2]
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

[1] The MAC logic in the ELAN 1x100 **truncates** receive frames that are longer than the preset maximum frame length before the CRC is computed. Thus received frames larger than the preset maximum frame length will also cause a CRC error, and hence oversize frames will actually be included in the JabberErrors counter and not the OversizeErrors counter.

[2] Due to the truncation of oversize frames, the count of octets received will not increase past one greater than the preset maximum frame size. Thus frames that are larger than the preset maximum frame length will increment the RXErrorOctets counter by exactly the one more than the preset maximum frame length, not the actual frame length. For example, if the maximum frame length is set to 1518 bytes, all oversize frames will increment the RXErrorOctets counter by 1519, regardless of the actual number of bytes in the frame.

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, while "CtrBlk" denotes the port descriptor counter structure. The per-port statistics maintained in the port descriptor counter structure may be retrieved at any time by direct access via the PCI bus interface; however, the statistics in the port descriptor are cached by the Switch Processor, and hence can only be retrieved via the message interface.

The ELAN 1x100 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 (ReceiveTimeStamp)	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.

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 1x100s (or members of the ELAN family) may be interfaced to the ELAN 1x100 device via the PCI bus. In general, such devices must follow the defined ELAN 1x100 messaging and frame transfer protocol for smooth operation. If the protocol is followed, then the external devices appear to the ELAN 1x100 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.

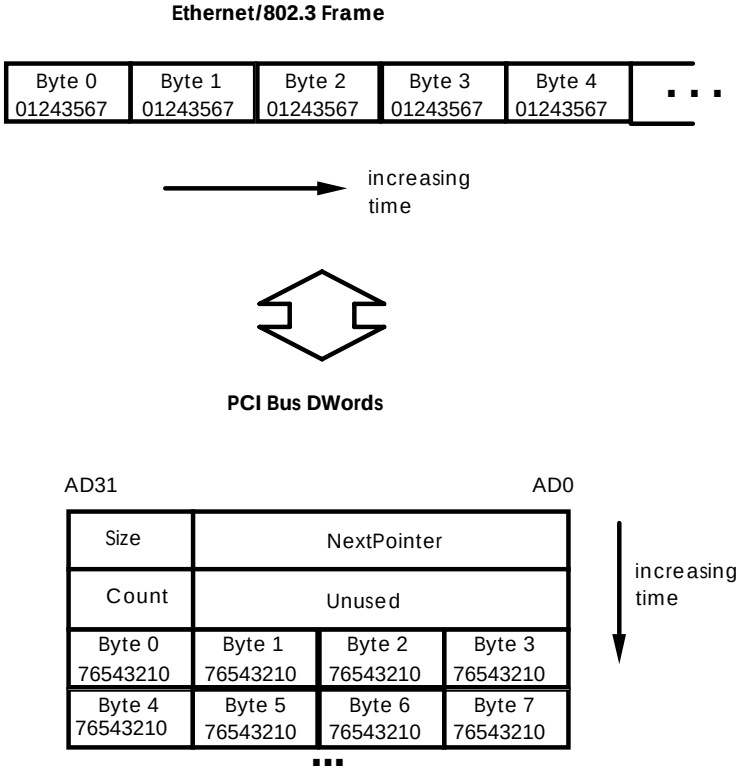
Byte Ordering

The ELAN 1x100 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 1x100 device. To properly align the various fields within a received Ethernet frame for handling by the Switch Processor, therefore, the ELAN 1x100 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:

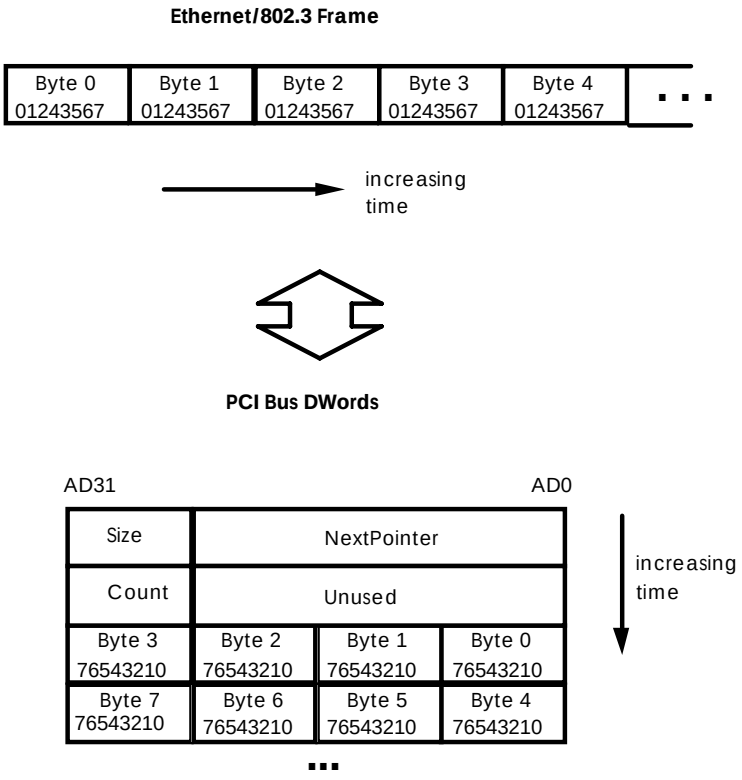
Default Byte Ordering



To permit interfacing of non-ELAN 1x100 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 1x100 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).

Example 4.1:

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

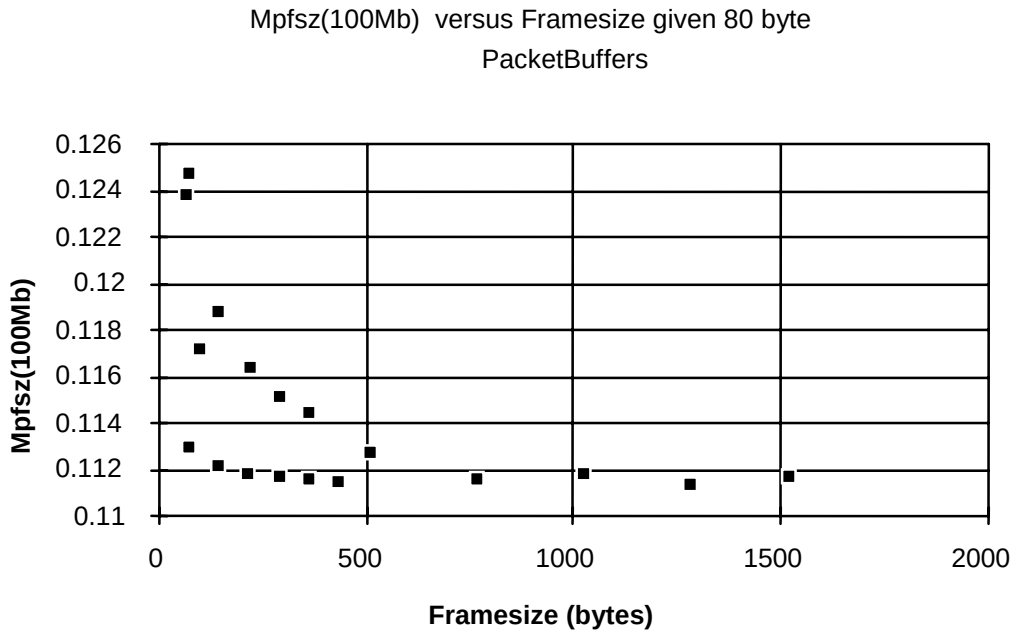
So Mpfsz is (104 bytes * 8 bits/bytes) / 6720 ns = 0.124 bits/ns.

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 1x100 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 1x100 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 1x100 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 PM3351 to determine the programming state of the block.
3. Writeable 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 PM3351 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 PM3351 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 to indicate an ELAN 1x100 device.

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 to 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 PM3351's ability to respond to PCI accesses. When a 0 is written to all bits in the command register, the PM3351 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 PM3351 does not implement I/O space.

MSen:

When the Memory Space Enable (MSen) bit is set to one, the PM3351 will respond to PCI bus memory accesses. Clearing MSen disables memory accesses.

Men:

When the Master Enable (Men) bit is set to one, the PM3351 can act as an master. Clearing Men disables the PM3351 from generating PCI accesses.

SCen:

The PM3351 does not decode PCI special cycles. The SCen bit is hardwired to 0.

MWIen:

The PM3351 does not generate memory-write-and-invalidate commands. The MWIen bit is hardwired to 0.

VPs:

The PM3351 is not a VGA device. The VPs bit is hardwired to 0.

PEen:

When the PEen bit is set to one, the PM3351 can report parity errors. Clearing the PEen bit causes the PM3351 to ignore parity errors.

Wcyc:

The PM3351 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 PM3351 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 PM3351 supports fast back-to-back transactions with other targets.

DPR:

The Data Parity Reported (DPR) bit is set high if the PM3351 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 PM3351 as a target. These bits are hardwired to zeros as the PM3351 asserts DEVSEL using fast (1-cycle) response timing.

STA:

The Signaled Target Abort (STA) bit is hardwired to zero because the PM3351 will never generate a Target Abort during a slave access cycle.

RTA:

The Received Target Abort (RTA) bit is set high by the PM3351 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 PM3351 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 PM3351 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 PM3351 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
Bits 31 to 24	R	CCODE[23:16]	0x02
Bits 23 to 16	R	CCODE[15:8]	0x00
Bits 15 to 8	R	CCODE[7:0]	0x00
Bits 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
Bits 31 to 24	R	BIST[7:0]	0x00
Bits 23 to 16	R	HDTYPE[7:0]	0x00
Bits 15 to 8	R/W	LT[7:0]	0x00 or 0x20
Bits 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 PM3351 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 PM3351. 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 deasserted. 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
Bits 31 to 24	R/W	MemBase[27:20]	MDATA[25:22] or 0x00
Bits 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 PM3351 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 PM3351 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 PM3351 register/memory resources.

FixBase[19:0]:

The Fixed Base Address (FixBase[19:0]) bits define the size of the memory space required for the PM3351 resources.

This 20-bit subfield is hardcoded to zero indicating the PM3351 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 PM3351 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 PM3351 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 deasserted. 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] (which represents CHIPID[3:0).

Register 0x30: Expansion ROM Base Address

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

XROMBase[31:0]:

The PM3351 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
Bits 31 to 24	R	MAX_LAT[7:0]	0x00
Bits 23 to 16	R	MIN_GNT[7:0]	0x03
Bits 15 to 8	R	INTPIN[7:0]	0x01
Bits 7 to 0	R/W	INTLINE[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 PM3351 uses. Because the PM3351 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 PM3351 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 PM3351 has no specific requirements in this regard; this register always returns zero.

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

Bit	Type	Function	Default
Bits 31 to 25	R	Reserved	0x00
Bits 24	R/W	TestDiscardTimer	0
Bits 23 to 16	R/W	Tdiscon	0x6E
Bits 15 to 10	R	Reserved	0x00
Bit 9	R	Mrd	0
Bit 8	R	Merror	0
Bits 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
Bits 31 to 0	R	CurrMasterReadAddr	0x00000000

CurrMasterReadAddr[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
Bits 31 to 0	R	CurrMasterWriteAddr	0x00000000

CurrMasterWriteAddr[31:0]:

Current write target address used by PCI bus master.

PCI Memory Space

The following table summarizes the mapping of the device resources to the PCI address space:

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
DBGCTRL	0xFF00A0	DEBUG functional block control register
	0xFF00A4- 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
Bits 31 to 12		Unused	X
Bits 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
Bits 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	h_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	RINTMASK	0
Bit 4	R/W	EXTMASK	0
Bit 3	R/W	RHALTMASK	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

[1]The default value is the negation of the logical value on the specified pin when RST* is deasserted.

h_EPBE:
Ethernet payload byteswap enable mode bit, in Host register space. The actual EPBE mode bit setting is the logical OR of h_EPBE and cr_EPBE (which is a similar mode bit in a control register accessible by the Switch Processor).

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

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
Bits 31 to 16		Unused	X
Bits 15 to 14	R/W	MXSEL[1:0]	MDATA[15:14][1,2]
Bit 13	R/W	MSLO	MDATA[13][1,2]
Bit 12	R/W	MDCAS	MDATA[12][1,2]
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. If EDO DRAM is used with the PM3351 for non-switching apllications it must 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 1x100 will generate control signals for 2-CAS DRAMs; otherwise, the ELAN 1x100 will assume 1-CAS DRAM devices

MTYPE3[2:0]:

Indicates the type of memory device selected with MCS[3]*:

000	Reserved
001	Reserved
010	15ns SRAM
011	Reserved

100 Reserved
101 150ns (E)EPROM
110 60 ns 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 deasserted.

Note 2: while these bits are implemented on the ELAN 1x100 device they are not intended to be used for normal switching. DRAM memory types are supported for management and special applications purposes only.

DCONFIG (Device Configuration register)

Bit	Type	Function	Default
Bits 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]
Bits 9 to 6	R/W	CHIPID[3:0]	MDATA[25:22][1]
Bits 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 dvice 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 deasserted, placing the ELAN 1x100 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 1x100 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 1x100 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. Because the logical implementation of the divider the value of RTCDIV[5:0] is set to must be set to the numeric value of the SYSCLK frequency in megahertz less 1. That is, RTCDIV[5:0] = (f_{SYSCLK} –1); e.g., 49 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 deasserted.

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	Type	Function	Default
Bit 31	R/W	vector force recognized	0
Bits 30 to 9		unused, reserved	0
Bit 8	R/W	vector force enable	0
Bit 7	R/W	instruction cache flush	0
Bit 6	R/W	data cache flush	0
Bit 5	R/W	RISC execption disable	0
Bits 4 to 3		unused, reserved	0
Bit 2	R/W	RISC breakpoint halt enable	0
Bit 1	R/W	breakpoint #2 enable	0
Bit 0	R/W	breakpoint #1 enable	0

Registers 0x00FF00A0: DEBUG functional block control register

Refer to the DBGCTRL register for a description of this register.

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 outlines the interface between the PM3351 Switch Processor and the various internal functional blocks. This interface consists of the following:

- the control registers that are accessible by the Switch Processor
- the interrupts to the Switch Processor
- the mapping of the general purpose input lines (GPI) from the internal functional blocks to the Switch Processor
- the mapping of the general purpose output lines (GPO) from the Switch Processor to the internal functional blocks
- the mapping of the coprocessor test condition bits from the internal functional blocks to the Switch Processor

Normative reference on functional block names with respect to their description elsewhere in this document:

DPI this refers to the **"DMA transmit channel"**. This functional block is used to read a frame across the PCI expansion bus and write it to the TX_FIFO for transmit.

DMA unless otherwise noted, the DMA functional block refers to the **"DMA receive channel".**

DPO this refers to the **"DMA transfer handshake channel".** This is the name of the functional block that supports the eight Transfer rings and the Request and Acknowledge counter increments to remote devices over the PCI expansion bus.

RPCIM
this refers to the **"PCI access channel"** which allows the Switch Processor to initiate PCI bus master transactions.

PCI_BIU
PCI Bus Interface Unit. The functional block that implements a PCI revision 2.1 compatible PCI bus master, bus slave (target), and configuration registers.

TX_FIFO
Transmit Fifo. This functional block queues up Ethernet frames that have

been read by the DMA transmit channel before transmitting on the Link media.

Local Memory

The ELAN 1x100 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, 24-bit tri-byte, or 16-bit halfword.

Register Map

General-Purpose Registers

There are two banks of sixteen 16-bit general registers, *gr0* through *gr15*. They are used for all arithmetic operations, the source/destination of memory load/store operations, and for branch condition testing. Access to these general registers is via CPU instructions. All general registers are identical; they are all readable and writeable by software.

In general, one bank of 16-general registers is used for mainline code and the second bank of 16-general register is used during exception (i.e. interrupt routine) processing. The register bank is switched automatically when either taking or returning from an interrupt. Switching of the register banks can also be done under explicit control of firmware by executing REGSW instruction. The alternate registers are accessed with the same indices as the normal register set.

Special-Purpose Registers

There are a total of 32 special-purpose registers that are used by the Switch Processor that are used solely by the processor for implementing the core processor instruction set: these include memory pointers, subroutine return addresses, exception control and status information, support ofthe microcode instructions, and implementation of the the real-time clock and alarms. The set of special purpose registers implemented by the ELAN 1x100 (PM3351) is the same as that implemented by the ELAN 8x10 device (PM3350).

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

There are a total of 96 16-bit control registers, *cr0* through *cr95*, that provide control and status of the functional blocks that comprise the PM3351. This set of control is, in general, device specific. For examplemost of the control register set of the ELAN 1x100 (PM3351) is different than that of the ELAN 8x10 device (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.

Registers that are marked as “reserved” are not to be written to. Reads from reserved registers have no side-effects.

In the table below the “Address” column refers to the explicit address that is used by the Switch Processor for accessing the register via MOVE and LDI instructions; the “CR” column refers to the control register number that is used for accessing the register when writing assembly code. For example, the DMSTAT register Address is 0x21 but to MOVE the contents of the register to the general register (say gr5), the assembly code is: "move cr1, gr5"

Mnemonic	CR	Address	Register
DMCTRL	cr0	0x20	DMA control word
DMSTAT	cr1	0x21	DMA status register
INSTCSR	cr2	0x22	RAM/ROM instruction control register
INSTDATA	cr3	0x23	RAM/ROM instruction data register
DMFLISTL	cr4	0x24	DMA free PacketBuffer list pointer low (address bits [15:0])
DMFLISTH	cr5	0x25	DMA free PacketBuffer list pointer high (address bits [23:16])
MREG_ADDR	cr6	0x26	Memory Interface Register Address register
MREG_DATA	cr7	0x27	Memory Interface Register Data register
SMR	cr8	0x28	Reserved
REQID	cr9	0x29	Request info
ACKID	cr10	0x2A	Acknowledge info
TASKCTR	cr11	0x2B	Task counter
DMNFREEL	cr12	0x2C	DMA next free PacketBuffer low
DMNFREEH	cr13	0x2D	DMA next free PacketBuffer high
CTRSEL	cr14	0x2E	Request/ack counter select reg
CTRDATA	cr15	0x2F	Request/ack counter data reg
RPCICMD	cr16	0x30	RPCIM command register

RPCIDATA	cr17	0x31	RPCIM Data register
RPCIADDRL	cr18	0x32	RPCIM Address register high
RPCIADDRH	cr19	0x33	RPCIM Address register low
RPCISTAT	cr20	0x34	RPCIM status register
WTIMER	cr21	0x35	RST module: Watchdog Timer
FRST	cr22	0x36	RST module: Firmware RST register
DBGCTRL	cr23	0x37	DBG module control register
DPI_DD0L	cr24	0x38	DPI: data descriptor word 0, low halfword (bits [15:0])
DPI_DD0H	cr25	0x39	DPI: data descriptor word 0, high halfword (bits [31:16])
DPI_DD1L	cr26	0x3A	DPI: data descriptor word 1, low halfword (bits [15:0])
DPI_DD1H	cr27	0x3B	DPI: data descriptor word 1, high halfword (bits [31:16])
DPI_DD2L	cr28	0x3C	DPI: data descriptor word 2, low halfword (bits [15:0])
DPI_DD2H	cr29	0x3D	DPI: data descriptor word 2, high halfword (bits [31:16])
DPI_DD3L	cr30	0x3E	DPI: data descriptor word 3, low halfword (bits [15:0])
DPI_DD3H	cr31	0x3F	DPI: data descriptor word 3, high halfword (bits [31:16])
DPI_CTRL	cr32	0x40	DPI: control/status register
TXF_WSIZE	cr33	0x41	TX FIFO: write size register (at FIFO tail)
DPI_RDDL	cr34	0x42	DPI: Remote DD address, low halfword (bits [15:0])
DPI_RDDH	cr35	0x43	DPI: Remote DD address, high halfword (bits [31:16])
TXF_CSR	cr36	0x44	TX FIFO control/status reg
TXF_RADDR	cr37	0x45	TX FIFO read address pointer (within FIFO)
TXF_WADDR	cr38	0x46	TX FIFO write address pointer (within FIFO)
TXF_DSIZE	cr39	0x47	TX FIFO: size of frame transmitted on MAC transmit link media
TXF_DATAL	cr40	0x48	TX FIFO: Switch Processor read/write data reg, low halfword
TXF_DATAH	cr41	0x49	TX FIFO: Switch Processor read/write data reg, high halfword
TXF_VALID	cr42	0x4A	TX FIFO: Write/Read byte enables
PBSIZE	cr43	0x4B	Packet Buffer size register. Used for PacketBuffers in device's local memory
RPBSIZE	cr44	0x4C	Remote Packet Buffer size register. Used for reading PacketBuffers over PCI expansion bus.
IPCIDATAL	cr45	0x4D	DPO: PCI increment counter write data, low halfword
IPCIDATAH	cr46	0x4E	DPO: PCI increment counter write data, high halfword
ICHIP_MASK	cr47	0x4F	DPO: transfer handshake channel chip mask
IDD0L	cr48	0x50	DPO: transfer handshake channel Data Descriptor word 0, bits [15:0]
IDD0H	cr49	0x51	DPO: transfer handshake channel Data Descriptor word 0, bits [23:16]
IDD1L	cr50	0x52	DPO: transfer handshake channel Data Descriptor word 1,

			bits [15:0]
IDD1H	cr51	0x53	DPO: transfer handshake channel Data Descriptor word 1, bits [23:16]
IDD2L	cr52	0x54	DPO: transfer handshake channel Data Descriptor word 2, bits [15:0]
IDD2H	cr53	0x55	DPO: transfer handshake channel Data Descriptor word 2, bits [23:16]
IDD3L	cr54	0x56	DPO: transfer handshake channel Data Descriptor word 3, bits [15:0]
IDD3H	cr55	0x57	DPO: transfer handshake channel Data Descriptor word 3, bits [23:16]
IDD0FLAG	cr56	0x58	DPO: transfer handshake channel Data Descriptor word 0, bits [31:24]
IDD1FLAG	cr57	0x59	DPO: transfer handshake channel Data Descriptor word 1, bits [31:24]
IDD2FLAG	cr58	0x5A	DPO: transfer handshake channel Data Descriptor word 2, bits [31:24]
IDD3FLAG	cr59	0x5B	DPO: transfer handshake channel Data Descriptor word 3, bits [31:24]
IC_CTRL	cr60	0x5C	DPO: transfer handshake channel control/status register
IC_DATA	cr61	0x5D	DPO: transfer handshake channel ring data access register
DSTAIL	cr62	0x5E	TX FIFO logic: disposition FIFO, tail register (write interface)
DSHEAD	cr63	0x5F	TX FIFO logic: disposition FIFO, head register (read interface)
LC_SEL	cr64	0x60	LINK: link constant register address
LC_DATA	cr65	0x61	LINK: link constant register data
HASHCTRL	cr66	0x62	Hash logic: control/status register
HMASK	cr67	0x63	Hash array size mask
HBASEL	cr68	0x64	Hash array base low (address bits [15:0])
HBASEH	cr69	0x65	Hash array base high (address bits [23:16])
HDPISRCRL	cr70	0x66	Hash result register: DPI source address, low (address bits [15:0])
HDPISRCRH	cr71	0x67	Hash result register: DPI source address, high(address bits [23:16])
HDMADSTRL	cr72	0x68	Hash result register: Link RX frame destination address, low (address bits [15:0])
HDMADSTRH	cr73	0x69	Hash result register: Link RX frame destination address, high (address bits [23:16])
HDMASRCRL	cr74	0x6A	Hash result register: Link RX frame source address, low (address bits [15:0])
HDMASRCRH	cr75	0x6B	Hash result register: Link RX source address, high (address bits [23:16])

HDPISRCL	cr76	0x6C	Hash DPI source MAC address, low 16-bits
HDPISRCM	cr77	0x6D	Hash DPI source MAC address, mid16-bits
HDPISRCH	cr78	0x6E	Hash DPI source MAC address, high 16-bits
HDMADSTL	cr79	0x6F	Hash Link RX destination MAC address, low 16-bits
HDMADSTM	cr80	0x70	Hash Link RX destination MAC address, mid 16-bits
HDMADSTH	cr81	0x71	Hash Link RX destination MAC address, high 16-bits
HDMASRCL	cr82	0x72	Hash Link RX source MAC address, low 16-bits
HDMASRCM	cr83	0x73	Hash Link RX source MAC address, mid 16-bits
HDMASRCH	cr84	0x74	Hash Link RX source MAC address, high 16-bits
LWCTRL	cr85	0x75	Link control register
LWSTAT	cr86	0x76	Link status register
LWBACK	cr87	0x77	Link transmit Backoff timer
LWNBUFS	cr88	0x78	Link receive: number of PacketBuffers in received frame
LWFIRSTL	cr89	0x79	Link receive: address of first PacketBuffer in received frame (address bits [15:0])
LWFIRSTH	cr90	0x7A	Link receive: address of first PacketBuffer in received frame (address bits [23:16])
LWLASTL	cr91	0x7B	Link receive: address of last PacketBuffer in received frame (address bits [15:0])
LWLASTH	cr92	0x7C	Link receive: address of last PacketBuffer in received frame (address bits [23:16])
LWDSIZE	cr93	0x7D	Link receive frame size (in bytes)
ACKRPTR	cr94	0x7E	DPO: acknowledge pointer for transfer ring Valid during acknowledge interrupt
TXF_RSIZE	cr95	0x7F	TX FIFO: read size register (at head of FIFO)

Register Descriptions

CPU Special Registers

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 (cr0): DMA control register (DMCTRL)

Bit	Type	Function	Default
15	R/W	reserved	0
14:13	R/W	cop_reg [15:14]	0
12:9	R/W	Unused	0
8	R/W	select_dmstat2	0
7:4	R	DMA state compare	0
3	R/W	cr_EPBE	0
2	R/W	LBK pin low true	0
1	R/W	LBK mode	0
0	R/W	halt DMA	1

cop_reg[15:14]:
software settable register bits that map to the Switch Processor coprocessor test condition bits [15:14].

select_dmsat2:
select for DMSTAT register.

(0) – DMSTAT1 register is read via DMSTAT control register.
(1) – DMSTAT2 register is read via DMSTAT control register.

DMA state compare:

this register field is compared to the value of the DMA state machine. The comparison is output on the dma_state_hit signal (0 is not equal; 1 is equal), which is observable via the DEBUG logic interface.

cr_EPBE:

Ethernet Packet big Endian mode bit, control register space.
The EPBE setting for the PM3351 is simply the logical OR of the EPBE mode bit that is in the HCTRL register (in PCI register space, h_EPBE) and this bit (in Switch Processor control register space).

LBK pin low true:

the logical state of the LBK pin on the PM3351 is the XOR of this bit with the LBK mode bit. This allows the LBK pin of the device to interface directly to Ethernet PHY devices that use a low-true or high-true version of LBK.

LBK mode:

when set, the link interface on the PM3351 is configured for loopback. This configuration affects both the logical value of the LBK pin of the device and the operation of the link interface. When the LBK mode bit is set the link will continuously transmit a single byte (see loopback byte link constant register) on the MII interface. The link receiver is unaffected by this mode bit. If the data on the MII receive interface is consistent with a valid receive frame (correct preamble and StartOfFrame delimiter) then the frame data will be received and formatted into PacketBuffers as is the case for normal frame reception.

Register 0x21 (cr1) DMA status register (DMSTAT)

this register is used to read one of two DMA status registers depending on the setting of the DMCTRL.select_dmstat2 register bit. Both DMSTAT1 and DMSTAT2 are implemented solely for device debug and are not required to be accessed by firmware during normal operation of the device.

DMSTAT1 register bit definition:

Bit	Type	Function	Default
15	R	Free PacketBuffer list lock state (Switch Processor sets this bit before accessing the DMFLIST register; this is typically done when the firmware is reallocating PacketBuffers during the acknowledge interrrupt service routine. The DMA hardware is locked out of accessing Free PacketBuffer pool when this bit is set).	0
14	R	Free list error interrupt state	0
13	R	Ethernet Packet Big Endian mode setting (Logical OR of HCTRL.h_EPBE and DMCTRL.cr_EPBE).	0
12	R	unimplemented	0
11	R	last_done state bit (set when last RXframe byte is read from the Link Receive FIFO; cleared in DMA state FRAME_DONE).	0
10	R	ln_first state bit (state bit that is set in-between receive frames. This bit is continuously set when the DMA state is IDLE. It is cleared as soon as the first data byte of a received frame is read from the Link Receive FIFO).	1
9:6	R	DMA state	0
5	R	RX_CTRL_FRAME	0
4	R	RX_FIFO_OVRUN	0
3	R	RXLONG	0
2	R	RXSHORT	0
1	R	RX_CRC_ERR	0
0	R	RX_MISALIGN	0

RX_CTRL_FRAME:

receive frame status: a frame was received having an Ethernet Length/Type field of the same value as the ETYPE Link control register. Status valid when a receive frame interrupt is asserted.

RX_FIFO_OVRUN:

receive frame status: a FIFO over-run occurred in the Link receiver during reception of this frame. Status valid when a receive frame interrupt is asserted.

RXLONG:

receive frame status: a frame was received with a length greater than that of

the MAX_SIZE Link control register value. Status valid when a receive frame interrupt is asserted.

RXSHORT:
receive frame status: a frame was received with a length less than that of the MIN_SIZE Link control register value. Status valid when a receive frame interrupt is asserted.

RX_CRC_ERR:
receive frame status: a frame was received having an incorrect 802.3 FrameCheckSequence field (that is, a CRC error). Status valid when a receive frame interrupt is asserted.

RX_MISALIGN:
receive frame status: this bit is set (1) if an odd number of framed data nibbles were received on the MII receive interface. The bit is clear (0) if an even number of framed data nibbles were received. Status valid when a receive frame interrupt is asserted.

DMSTAT2 register bit definition:

Bit	Type	Function	Default
15:12	R	unimplemented	0
11	R	start_xfr state bit (DMA sets this bit to initiate a burst access to the local memory interface. Valid in DMA state WAIT_XFR).	0
10	R	Link Receive FIFO read enable status	0
9	R	pbxfrdone state bit (DMA state bit that is set when all data has been written into the present PacketBuffer: either PacketBuffer is full or the last byte of the RX frame is in it.)	0
8	R	EndOfFrame status (DMA determines from Link Receive FIFO interface signals)	0
7:0	R	ln_size[7:0] (number of valid data bytes in PacketBuffer presently allocated to the DMA).	0

Register 0x23 (cr2): RAM/ROM Instruction Control register (INSTCSR)

Bit	Type	Function	Default
Bit 15	R/W	RAMPR	0
Bit 14	R/W	ROMPR	0
Bit 13		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 low 7 bits of the address are used for ROM accesses.

Register 0x23 (cr3): RAM/ROM Instruction Data register (INSTDATA)

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 (cr4): DMA freelist pointer low (DMFLISTL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	FLIST[15:0]	0

Register 0x25 (cr5): DMA freelist pointer high (DMFLISTH)

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 the Switch Processor or by the DMA logic.

During normal switching applications they are used as follows:

- the DMA deallocates PacketBuffers from the FreeLIST during reception of frames on the MII interface. It does this on a PacketBuffer –by-PacketBuffer basis by following the NextPTR field of the PacketBuffer that is pointed to by the current DMFLIST entry. The DMA keeps the pointer for the next PacketBuffer it will be using in the DMNFREE[23:0] control register.

- the Switch Processor reallocates the entire linked-list of PacketBuffers in a frame after the frame has been acknowledged to have been read across the PCI Expansion Bus by the required number of ELAN devices (which is when the RefCount field of the PacketBuffer has decremented to 0 - refer to the ELAN switch protocol discussion for additional information). PacketBuffers are reallocated on the head of the FreeLIST. Hence, the DMFLIST will be updated with a pointer to the first PacketBuffer in the frame that is being reallocated to the FreeLIST.

- there is a simple hardware interlock that is provided to allow the DMA and the Switch Processor to read and write the DMFLIST register. The Switch Processor does this by means of two flip instructions that set and clear a FreeLIST lock signal (flist_lock). The Switch Processor has priority for accessing the FreeLIST. If the FreeLIST lock is set while the DMA is currently accessing the FreeLIST, the DMA will abort the present access and will attempt to access the FreeLIST only after the FreeLIST lock is deasserted by the Switch Processor.

Register 0x26 (cr6): Memory Register Bank Address (MREG_ADDR)

Register 0x27 (cr7): 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.

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:
(1) the request snapshot register is zero.
(2) 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

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 (cr14): Request/ack counter select reg (CTRSEL)

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 (cr15): Request/ack counter data reg (CTRDATA)

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 (cr22): RST module: Firmware RST reg (FRST)

Bit	Type	Function	Default
Bit 15	R/W	f_ERST	0
Bit 14	R/W	Unused	0
Bit 13	R/W	MRST	0
Bit 12	R/W	DBG_RST	0
Bit 11 to 8	R/W	Unused	0
Bit 7	R/W	DPO_RST	0
Bit 6	R/W	TXF_RST	0
Bit 5	R/W	DPI_RST	0
Bit 4	R/W	LINK_RST	0
Bit 3	R/W	DMA_RST	0
Bit 2	R/W	RPCIM_RST	0
Bit 1	R/W	HASH_RST	0
Bit 0	R/W	CTR_BANK_RST	0

f_ERST:

If f_ERST transitions from logic 0 to logic 1, the ERST_ output pin is asserted (i.e. driven to TTL logic 0) for 10 ms (assuming 50MHz SYSCLK).

MRST:

if this bit is a logic 1, the internal PCI bus interface logic associated with being a bus master is reset.

DBG_RST:

if this bit is a logic 1, the logic associated with the DEBUG functional block is reset.

DPO_RST:

if this bit is a logic 1, the logic associated with the DPO (i.e. transfer handshake channel) functional block is reset.

TXF_RST:
If this bit is a logic 1, the Transmit FIFO functional block is reset.

DPI_RST:
if this bit is a logic 1, the logic associated with the DPI functional block is reset (the DPI is the abbreviated name of the DMA transmit channel).

LINK_RST:
If a LINK_RST bit is a logic 1, the Link 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.

RPICM_RST:
If this bit is a logic 1, the RPCIM 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.

CTR_BANK_RST:
If this bit is a logic 1, the Counter Bank functional block is held in its reset state.

DEBUG Functional Block

The DEBUG functional block is provided to assist in factory DEBUG of the device during system verification. It provides the facility for displaying certain internal device data on three pins of the PM3351. The hardware associated with the DEBUG module includes:

- a snapshot register that can capture and display the Switch Processor Program Counter, 2-bits at a time.
- a 128:1 serial mux for displaying one of 128 possible internal signals

Register 0x37: DEBUG 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 of the PM3351 drive valid logic levels; otherwise the DBG[2:0] outputs 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 Processor Load unit
001 Processor Instruction Fetch unit
010 DMA
011 HOSTIF (PCI slave accesses)
100 HASH
101 DPO (transfer ring support)
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 an alignment marker. The PC is output as described above for mode 001, but an active high pulse aligned to PC[3:2] is generated on the DBG[0] output.

011 This mode allows the current program counter (PC) to be output simultaneously with a selected internal signal. It differs from mode 001 in that only bits 11 through 2 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 until PC[11:10]. The DBG[0] output presents the internal signal selected by the SSEL[6:0] register bit.

100 This mode allows the current program counter (PC) to be output with an alignment marker. The PC is output as described above for mode 011, but an active high pulse aligned to PC[3:2] is generated on the DBG[0] output.

101 - 111 Reserved

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	cop_cond[13]	84	mrf_valid	41	gpo[3]
126	cop_cond[12]	83	DPI_mcf_wen	40	gpo[2]
125	cop_cond[11]	82	DPI_mrf_ren	39	gpo[1]
124	mcfg_en	81	dpwen	38	gpo[0]
123	oe_erst	80	dpweof	37	gpi[12]
122	core_reset_	79	dpi_learn_sa	36	gpi[11]
121	dporq	78	dpi_we1	35	gpi[10]
120	hmrq	77	dma_we1	34	gpi[9]
119	smrq	76	dma_we3	33	gpi[8]
118	dmrq	75	tx_ren	32	gpi[7]
117	imrq	74	tframe	31	gpi[6]
116	lmrq	73	tx_datainvalid	30	gpi[5]
115	hmrdy	72	txdsync_en	29	gpi[4]
114	smrdy	71	txf_set_transmit	28	gpi[3]
113	dmrdy	70	linkrframe	27	gpi[2]
112	imrdy	69	lrf_has_lastword	26	gpi[1]
111	lmrdy	68	dma2lrf_done	25	gpi[0]
110	sra_newaddr	67	dma_state_hit	24	cop_cond[10]
109	sra_active	66	linkren	23	cop_cond[9]
108	sra_done	65	flist_null	22	cop_cond[8]
107	sra_lastword	64	flist_lock	21	cop_cond[7]
106	swfvalid	63	gpo[29]	20	cop_cond[6]
105	srfwen	62	gpo[28]	19	cop_cond[5]
104	swfren	61	gpo[27]	18	cop_cond[4]
103	hwr	60	gpo[26]	17	cop_cond[3]
102	hrd	59	gpo[25]	16	cop_cond[2]
101	bkpt	58	gpo[24]	15	cop_cond[1]
100	merr	57	gpo[23]	14	cop_cond[0]
99	mperr	56	gpo[22]	13	intr[12]
98	rhalt	55	gpo[21]	12	intr[11]
97	rclk_en	54	gpo[20]	11	intr[10]
96	ramdis	53	gpo[19]	10	intr[9]
95	romdis	52	gpo[18]	9	intr[8]
94	req_lock	51	gpo[17]	8	intr[7]
93	holdoff_req_int	50	gpo[16]	7	intr[6]
92	req_present	49	gpo[15]	6	intr[5]

91	DPIprd_rq	48	gpo[14]	5	intr[4]
90	DPOpwr_rq	47	**gpo[10]**	4	intr[3]
89	DPIprd_gt	46	gpo[9]	3	intr[2]
88	DPOpwr_gt	45	**gpo[7]**	2	intr[1]
87	mwf_wen	44	gpo[6]	1	intr[0]
86	mcf_wen	43	gpo[5]	0	stall_
85	mrf_ren	42	gpo[4]		

Register 0x38 (cr24): DPI DataDescriptor word 0, low (DPI_DD0L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD0[15:0]	0

rDD0[15:0]:
Bits 15:0 of word 0 of Data Descriptor read over the PCI expansion bus.

Register 0x39 (cr25): DPI DataDescriptor word 0, high (DPI_DD0H)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD0[31:16]	0

rDD0[31:16]:
Bits 31:16 of word 0 of Data Descriptor read over the PCI expansion bus.

Register 0x3A (cr26): DPI DataDescriptor word 1, low (DPI_DD1L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD1[15:0]	0

rDD1[15:0]:
Bits 15:0 of word 1 of Data Descriptor read over the PCI expansion bus.

Register 0x3B (cr27): DPI DataDescriptor word 1, high (DPI_DD1H)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD1[31:16]	0

rDD1[31:16]:
Bits 31:16 of word 1 of Data Descriptor read over the PCI expansion bus.

Register 0x3C (cr28): DPI DataDescriptor word 2, low (DPI_DD2L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD2[15:0]	0

rDD2[15:0]:
Bits 15:0 of word 2 of Data Descriptor read over the PCI expansion bus.

Register 0x3D (cr29): DPI DataDescriptor word 2, high (DPI_DD2H)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD2[31:16]	0

rDD2[31:16]:
Bits 31:16 of word 2 of Data Descriptor read over the PCI expansion bus.

Register 0x3E (cr30): DPI DataDescriptor word 3, low (DPI_DD3L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD3[15:0]	0

rDD3[15:0]:
Bits 15:0 of word 3 of Data Descriptor read over the PCI expansion bus.

Register 0x3F (cr31): DPI DataDescriptor word 3, high (DPI_DD3H)

Bit	Type	Function	Default
Bit 15 to 0	R/W	rDD3[31:16]	0

rDD3[31:16]:
Bits 31:16 of word 3 of Data Descriptor read over the PCI expansion bus.

DPI Functional Block (DMA transmit channel)

The purpose of the DMA transmit channel (DPI functional block) is to read a frame of data over the PCI expansion bus. The Switch Processor writes the DPI_RDDL and DPI_RDDH registers with the address of the Data Descriptor that is to be read over the PCI expansion bus. The DPI logic then reads the frame data across the PCI expansion bus, PacketBuffer-by-PacketBuffer, automatically following the linked list of PacketBuffers. If the LEARN bit in the Data Descriptor flags field is set, then a hash lookup is done on the source address of the Ethernet frame (see description of HDPISRCL/M/H registers).

Both the Data Descriptor and the MAC source address of the Ethernet frame that has been read over the PCI expansion bus are held in control registers. The DPI functional block generates an interrupt to the Switch Processor after the Data Descriptor and MAC source and destination addresses have been read (TransmitFrame header in interrupt: DPI_HD_DONE). The NextDD field of the Data Descriptor is used by firmware to update the RemDD field of the Expansion Port Descriptor.

The data portion of the packet buffers (typically a MAC Ethernet frame with the preamble and start-frame-delimiter stripped) is read by the DPI from the internal PCI master read interface and writes the frame data into a data FIFO that is part of the TX_FIFO functional block.

In order to limit the processing that needs to be done by the Switch Processor the DPI functional block is capable of initiating an Acknowledge Counter increment after the frame has been read over the PCI expansion bus by interfacing directly with the transfer handshake channel.

Register 0x40 (cr32): DPI control/status register (DPI_CTRL)

Bit	Type	Function	Default
15 :14		Unused	0
13	R	no_PB	0
12:8	R	S[4:0]	0
7	R/W	Unused	0
6	R/W	enhpcirdmode	0
5	R/W	en_dpi_ack	0
4	R/W	en_dpi_learn_sa	0
3	R/W	sel_dpi_dd	0
2	R/W	dpi_dd_en	0
1	R/w	dpi_hd_en	1
0	R/W	dpi_done_en	0

no_PB:

state bit that is set if the data descriptor that was read acros the PCI expansion bus had a FirstPB field of 24'h0 (that is, the PacketBuffer pointer is a null pointer). This bit is valid during the DPI header in interrupt (DPI_HD_DONE) and is cleared when the DPI state machine returns to IDLE (which occurs in-between reading of each frame).

S[4:0]:

internal state vector for the DPI functional block. Present for observability.

enhpcirdmode:

If this bit is a logic 0 the DPI will read the entire packet buffers worth of data over the PCI expansion bus. If this bit is logic 1 the DPI will implement a more efficient transfer protocol: an Enhanced PCI read mode. 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	Size	NextPB
Word 1	Last_Size	Don't Care

Word 0 contains the normal information for the DPI. Word 1 contains the number of bytes in the last packet buffer in this frame. The number of PacketBuffers in the frame is pre-loaded from the NumPBs field of the Data

Descriptor as it is loaded in the DPI_DD1H register. As each PacketBuffer in the frame is read across the PCI bus (the actual number of bytes read across the PCI bus per PacketBuffer is given by the RPBSIZE register), the count of remaining PacketBuffers is decremented. When the count is 1 (indicating that this is the last PacketBuffer), the number of bytes read across the PCI bus is simply: (Size + 8). The number “8” accounts for the 8-bytes in the header of each PacketBuffer structure.

en_dpi_ack:

Enable for DPI initiated Acknowledge. If enabled, the DPI will initiate the Acknowledge counter increment over the PCI expansion bus without requiring intervention of the Switch Processor.

en_dpi_learn_sa:

Enable for the feature to allow the DPI to initiate a Hash lookup on the MAC source address of a frame if the LEARN bit was set (1) in the Data Descriptor that was read over the PCI expansion bus.

sel_dpi_dd:

the setting of this bit determines which of two possible interrupt conditions will be mapped to the DPI head-of-packet interrupt line (TransmitFrame header in interrupt: DPI_HD_DONE) that goes to the Switch Processor . One of two conditions can be selected. If sel_dpi_dd is clear (0): the interrupt is asserted when the Data Descriptor and the first 12 bytes of the first PacketBuffer are read over the PCI expansion bus (i.e. select the dpi_hd_intr signal as the source of the DPI_HD_DONE interrupt line). If sel_dpi_dd is set (1): the interrupt is asserted as soon as the Data Descriptor is read over the PCI bus (i.e. select the dpi_dd_intr signal as the source of the DPI_HD_DONE interrupt).

dpi_dd_en:

this bit is used as an enable for the dpi_dd_intr signal. For the dpi_dd_intr to be logically asserted the dpi_dd_en bit must be set (1).

dpi_hd_en:

this bit is used as an enable for the dpi_hd_intr signal. For the dpi_hd_intr to be logically asserted the dpi_hd_en bit must be set (1).

dpi_done_en:

this bit is used as an enable for the DPI_DONE interrupt signal. For DPI_DONE interrupt to be logically asserted the dpi_done_en bit must be set (1).

Register 0x41(cr33): TX FIFO write size register (at FIFO tail)

Bit	Type	Function	Default
Bit 15 to 11		Unused	0
Bit 10 to 0	R/W	TXF_WSIZE	0

TXF_WSIZE:

TXF_WSIZE control register: this is the number of valid data bytes that are in the frame written into the TX_FIFO data ram (either via the DPI module or by the Switch Processor writing a management frame) -OR- into the MAC Ctrl frame FIFO. The size includes the 4-byte CRC (regardless as to whether the CRC was appended by the TX_FIFO logic or was already present in the frame).

Register 0x42 (cr34): DPI Remote DataDescriptor address, low (DPI_RDDL)

Bit	Type	Function	Default
Bit 15 to 0	**R/W**	**RemDD[15:0]**	**0**

RemDD[15:0]:

this is the low 16 bits of the PCI address from which a Data Descriptor is to be read by the DPI.

Register 0x43 (cr35): DPI Remote DataDescriptor address, high (DPI_RDDH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	RemDD[31:16]	0

RemDD[31:16]:

this is the high 16 bits of the PCI address from which a Data Descriptor is to be read by the DPI.

TX_FIFO functional block:

The TX_FIFO functional block is intended to be a "smart " datapath from the DPI to the MAC 100-Mbit link. It presents a FIFO interface to the DPI and MAC Link functional blocks. This allows the DPI and MAC link interfaces to operate independently.

During frame transmission, the TX_FIFO read interface (head of fifo which interfaces to the MAC link transmit logic) will initiate re-transmission of the frame after collisions (assuming half-duplex configuration) if the disposition code of the frame is not changed (see below).

A running tally of the size of frames being written to or read from the TX_FIFO and being transmit on the MAC link media is kept (TXF_DSIZE, TXF_WSIZE, and TXF_RSIZE control registers). Additionally, the TransmitStatus of a frame that has been transmit (or has attempted to transmit) on the MAC link media is held in an internal register and is valid during assertion of either the **tx_done** or **collision** interrupts to the Switch Processor (see TXF_DSIZE register description).

Major components of the TX_FIFO datapath

1. a 512_word x 36-bit data ram.

   This ram holds frame data which is to be transmitted by the MAC link functional block. For normal data frame transmission, the DPI functional block transparently writes data into the TX_FIFO data ram and the transmit interface of the MAC link functional block reads data out of the data ram. The format of the data in the data ram is a 32-bit data word and a 4-bit byte-enable code. A byte-enable code of 4'h0 represents an End-Of-Frame delimiter. At the end of writing a frame into the data ram the TX_FIFO hardware autonomously writes a pointer to the next EndOfFrame delimiter. This is done to allow easy dropping of frames at the head of the fifo.

2. a 32_word x 4-bit disposition fifo (DSFIFO).

   Each and every frame in the TX_FIFO data ram has associated with it a 4-bit disposition code. This code is written into the DSFIFO either concurrent or after the frame has been written into the TX_FIFO data ram. When the frame comes to the "head" of the TX_FIFO data ram a state machine decodes the disposition code and takes appropriate action.

3. a 64-byte FIFO that allows the Switch Processor to format and transmit a MAC Control Frame (MCFIFO).

   The TX_FIFO allows both data frames and MAC control frames to be transmit. A mulitplexer selects whether to transmit a MAC control frame rather then a

data frame on the MAC link. The Switch Processor can write the Control Frame to the MCFIFO and then flip a bit to signal to the TX_FIFO hardware that the frame is to be transmit from the MCFIFO to the MAC link.

4. Support to allow the Switch Processor to write an Ethernet frame to the TX_FIFO data ram

   This is required for transmitting a "TX management frame"; that is a frame created by this PM3351 chip for transmission on the link media. Firmware running on the Switch Processor is able to write a data frame to the TX_FIFO along with its associated DSFIFO disposition code.

5. Support of firmware read of a frame from the TX_FIFO.

   This is required for an "RX management frame". What makes an "RX management" type frame special from a hardware perspective is that the frame is expected to be read out of the TX_FIFO data ram and will most likely not be transmitted on the link media.

TX_FIFO interrupts to Switch Processor

1. tx_done : the frame is done being transmitted on the link media without collision. This interrupt is conditional on the TXF_CSR.en_tx_done_intr bit being set.
2. txf_stop : a frame is at the head of the TX_FIFO and has a disposition code of type "STOP".

Disposition of Frames:

Disposition of frames can be done at both the tail and the head of the TX_FIFO datapath. This allows either a one-stage (disposition via DSTAIL register only) or two-stage disposition (disposition first via DSTAIL and second via DSHEAD) of the frame. The interface provided for doing this is a simple control register interface. The DSTAIL register writes disposition information into the DSFIFO ram. Each and every frame which is written into the TX_FIFO data ram is **required** to have **exactly one** write to the DSTAIL register. Normal data frames only require dispositioning via a write to the DSTAIL register.

From a firmware perspective there are 16 possible disposition codes. The hardware decodes these 16 disposition codes are mapped into one of three types of hardware action codes:

- TRANSMIT
- STOP
- DROP

The DSHEAD register is provided to allow a flexible, two-stage disposition of the frame. A frame that had a hardware action code of "STOP" written into the DSTAIL register will assert the **txf_stop** interrupt to the Switch Processor when the frame comes to the head of the fifo. The firmware routine for servicing **txf_stop** can then do the following:

- read the disposition code that was written when the frame was at the tail of the fifo (i.e. when the frame was being written into the TX_FIFO data ram).
- firmware can read the contents of the frame, if so desired, from the TX_FIFO data ram.
- the firmware can then modify the disposition code by writing a disposition code that maps to TRANSMIT or DROP into the DSHEAD fifo. This will result in the frame that had generated the **txf_stop** interrupt to be either transmitted or dropped.

Firmware restriction rules on writing the DSTAIL and DSHEAD registers:

These rules apply for each frame which is written to the TX_FIFO data ram, either by the DPI functional block (that is, a standard data frame read over the PCI expansion bus) or by firmware (that is, a TX management frame).

RULE 1:

Under no conditions should the DSHEAD control register be written to if the "txf_stop" interrupt is deasserted since this may yield an unrecoverable device error.

RULE 2:

This rule, which follows from rule 1 but is restated for clarity, shows that only a frame with a STOP code written into the DSTAIL disposition register can be modified when the frame reaches the head of the TX_FIFO data ram.

Sequence Description for Transmission of Ethernet Data Frames.

- the frame is read over the expansion bus using the DPI channel on the PM3351. Payload data from the frame is transparently written to the TX_FIFO data ram as the linked-list of PacketBuffers is read on the PCI bus. After the Data Descriptor and the first 12-bytes of the frame have been read across the PCI bus the "DPI_HD_DONE" interrupt line will be asserted. The firmware interrupt service routine writes one 4-bit disposition code into the DSTAIL control register; this write will cause the disposition code to be loaded into the 32_wordx4-bit disposition fifo (DSFIFO). For normal data frames this will be one of the seven firmware codes that map to "TRANSMIT". Note that the "DPI_HD_DONE" interrupt line will become asserted concurrent to frame data being written to the TX_FIFO data ram.

- if the TXF_CSR.fix_crc bit is set (1) and the frame that was written to the TX_FIFO had an incorrect or missing CRC, then a correct 4 byte "CRC" is appended (that is, the correct 802.3 FrameCheckSequence is appended).

- the frame will eventually make its way to the head of theTX_FIFO data ram as frames ahead of it are either transmitted or dropped. When the frame reaches the head of the TX_FIFO data ram the output state machine will read out the disposition code from the DSFIFO to the DSHEAD register. Since the code in this example is a TRANSMIT code, the state machine will autonomously load the frame into the TX interface of the MAC link functional block on a byte-by-byte basis. When the last byte of the frame has been successfully transmitted on the link media without experiencing a collision the TX_FIFO will do one of two things depending on the setting of the TXF_CSR.en_tx_done_intr bit:

- if the en_tx_done_intr bit is set then the TX_FIFO output state machine will enter a holding state until the GPO line "thaw_tx_done" is flipped. The TX_FIFO output state machine will then return to the IDLE state, from whence it will read out the next available frame to transmit.

- if the en_tx_done_intr bit is clr then the TX_FIFO output state machine will immediately return to IDLE.

The TX_FIFO output state machine leaves the IDLE state whenever there is at least one frame that is ready to be transmit from the TX_FIFO data ram (the entire frame has been written into the TX_FIFO data ram and has been dispositioned by writing to the DSTAIL control register) or a MAC control frame is ready for transmission.

If during the course of the transmit a collision (half-duplex only) occurs the TX_FIFO will hold in a state until the collision interrupt is serviced by the Switch Processor; this is done to allow the service routine for collision to change the disposition code in the

DSHEAD control register from TRANSMIT to DROP in the case of excess collisions (i.e. 16 consecutive collisions) and to update collision statistics.

Register 0x44 (cr36): TX_FIFO Control register (TXF_CSR)

Bit	Type	Function	Default
Bit 15	R/W	Unused	0
Bit 14	R/W	AUTO_INC_MODE	1
Bit 13	R/W	SW_WRITE_MODE	0
Bit 12	R/W	FRAME_MODE	0
Bit 11	R/W	BSWP_RWDATA	0
Bit 10	R/W	BSWP_RRDATA	0
Bit 9	R/W	WR_MCFRAME	0
Bit 8	R/W	RD_MCFRAME	0
Bit 7	R/W	EN_TXDONE_INTR	1
Bit 6	R/W	FIX_CRC	0
Bits 5:0	R/W	MAX_PKTS[5:0]	0x1E

AUTO_INC_MODE:

Puts theTX_FIFO into auto increment mode. This feature, if set (1), results in having various write and read pointers auto incremented for Switch Processor frame accesses to the data ram and MAC control frame rams (that is, if the FRAME_MODE bit is also set). This relieves firmware from having to keep track of the read and write pointers. Affected ram address pointers: wr_ptr and rd_ptr (for TX_FIFO data ram) and mcwptr (for MAC Ctrl frame ram).

SW_WRITE_MODE:

controls whether the TXF_DATAH and TXF_DATAL register is configured for read or write of either the TX_FIFO data ram or MAC Ctrl ram. If this bit is set (1) then data written to the TXF_DATAH/L registers will be written to the selected ram; if the bit is clear (0) then data read from the selected ram may be read by the Switch Processor in the TXF_DATAH/L register.

FRAME_MODE:

defines the type of access with respect to the data ram. If this bit is set (1) then write accesses to the TX_FIFO data ram by the Switch Processor are taken to be Ethernet Frame data and are formatted in the rams accordingly. If this bit is clear (0) then the Switch Processor may arbitrarily read and write bytes in either the data ram or the MAC control frame ram. In normal switch applications this bit is set (1) during the firmware initialization sequence.

BSWP_RWDATA:

if set (1), then a byte-swap is performed data being written to the TX_FIFO data ram by the Switch Processor.

BSWP_RRDATA:

if set (1), then a byte-swap is performed on data being read by the Switch Processor from the TX_FIFO data ram.

NOTE: these two BSWP bits are not intended to be used by the programmer and should be left in the default state, 0. Instead, if a byte-swap is desired to be performed on Ethernet frame data, use the EPBE mode bit. If EPBE mode bit is (1), the a byte-swap is performed on the Ethernet Frame data in the DPI module, which is prior to the TX_FIFO in the transmit datapath.

WR_MCFRAME:

Mode bit for selecting the ram that is the target of a Switch Processor write access. If set (1), then MAC CTRL frame ram is target of write; if clear (0), target is TX_FIFO data ram.

RD_MCFRAME:

Mode bit for selecting the ram that is the target of a Switch Processor read access. If set (1), then MAC CTRL frame ram is target of read; if clear (0), target is TX_FIFO data ram.

EN_TXDONE_INTR:

if set (1), enables the TX_DONE interrupt to the Switch Processor at end of all frames successfully transmit on link media. If clear (0) the TX_DONE interrupt will never be asserted.

FIX_CRC:

if set (1) enables appending of a correct 802.3 FrameCheckSequence (i.e. CRC) at the end of every frame written to either the TX_FIFO data ram or MAC Control frame ram if, and only if, the CRC for the frame as it was originally written had an incorrect FrameCheckSequence. If clear (0) no CRC will be appended.

NOTE- if a CRC is appended on the frame then the frame length will be increased by 4 bytes. However, no field in the Ethernet Frame (such as the LENGTH field of an 802.3 Frame, is modified). Example: if 60 bytes of frame data were written by the Switch Processor into the TX_FIFO data ram but without a CRC, and the FIX_CRC bit is set, then the TX_FIFO logic will append a 4 byte CRC and the final frame size will be 64-bytes.

MAX_PKTS[5:0]:

this is the maximum number of packets that are allowed in the disposition fifo (DSFIFO). The value in this register is compared to the number of valid entries in the DSFIFO ram and is used to set a full threshold flag that is used by internal logic to hold off the DPI from fetching additional frames.

This value default to decimal 30; the value in this register should not be written to be greater than 30.

Register 0x45 (cr37): TX_FIFO read address pointer (TXF_RADDR)

Bit	Type	Function	Default
Bit 15 to 11	R/W	MCRPTR[6:2]	0
Bit 10 to 0	R/W	RD_PTR[10:0]	0

MCRPTR[6:2]:

this is the read pointer used for read access to the 64-byte MAC Control frame ram. MCRPTR[6:0] is the byte address of the ram.

RD_PTR[10:0]:

this is the read pointer used for read access to the TX_FIFO data ram. RD_PTR[10:0] is the byte address of the data ram.

NOTE- in normal operation the MCRPTR and RD_PTR registers are not intended to be written. Writing these registers during switch operation can cause data transmit errors on the MAC transmit link interface.

Register 0x47 (cr39): Transmit Frame status register (TXF_DSIZE)

Bit	Type	Function	Default
Bit 15	R	TXLONG	0
Bit 14	R	DEFERRED_TX	0
Bit 13 to 11	R	Unused	0
Bit 10 to 0	R	TXSIZE	0

The Transmit Frame status register holds information on a frame that has been successfully transmit on the link media (that is, if in half-duplex there was no collision). The contents of this register are valid for access by the Switch Processor when the TX_DONE interrupt is asserted. The transmit datapath is pipelined to allow for full-throughput on the MAC link interface: TX_DONE interrupt can be asserted for transmit frame #N while the MAC link interface is transmitting data on the MII for frame #(N+1).

TXLONG:

this bit is set (1) if the frame that was read out of the TX_FIFO had a frame size greater thant the maximum frame size control register (see link constant register MAX_SIZE). The transmit frame will be truncated to be exactly MAX_SIZE number of bytes on the link media.

DEFERRED_TX:

this bit is set (1) if the frame transmission attempt was deferred.

NOTE – this bit is set by hardware regardless as to whether the deference occurred during the first or subsequent transmission attempt. For support of the Ethernet MIB statistic StatsDeferredTransmissions the Switch Processor must qualify this bit by first transmission attempt and not a collision.

TXSIZE:

this is the number of bytes that were successfully transmit on the MAC link media (the MII interface). The value of this register can be used for maintenance of Ethernet byte transmit statistics and is valid during assertion of the TX_DONE interrupt.

Register 0x48 (cr40): TX_FIFO Switch Processor read/write data register, low (TXF_DATAL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	SWDATA[15:0]	0

Register 0x49 (cr41): TX_FIFO Switch Processor read/write data register, high (TXF_DATAH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	SWDATA[31:16]	0

SWDATA[31:0]:

this register pair is used by the Switch Processor for access to the TX_FIFO data ram and the MAC Control frame ram. The mode (read versus write), is set via the SW_WRITE_MODE bit in the TXF_CSR register.

Register 0x4A (cr42): TX_FIFO Valid byte register (TXF_VALID)

Bit	Type	Function	Default
Bit 15 to 12	R	OSTATE[3:0]	0
Bit 11 to 8	R	ISTATE[3:0]	0
Bit 7 to 4	R	RVALID[3:0]	0
Bit 3 to 0	R/W	WVALID[3:0]	0

OSTATE[3:0]:

this read-only register gives the value of the OSTATE state machine in the TX_FIFO module.

ISTATE[3:0]:

this read-only register gives the value of the ISTATE state machine in the TX_FIFO module.

RVALID[3:0]:

this read-only register gives the value of the VALID bit field of the TX_FIFO data ram. The TX_FIFO data ram is 36-bits wide. During Switch Processor read of the TX_FIFO data ram the TXF_DATAL and TXF_DATAH control registers will return the low 32-bits of data (typically Ethernet frame data) while the upper 4-bits of the data ram can be read from RVALID[3:0]. The RVALID[3:0] bit-field codes are follows:

- an EndOfFrame delimiter is given by the code 4'h0
- a valid byte enable pattern corresponding to four (code 4'b1111), three (code 4'b1110), two (code 4'b1100), or one (code 4'b1000) byte. This is the number of valid bytes in the data-word that is being read from the TX_FIFO data ram.
- all other RVALID[3:0] code combinations are illegal.

WVALID[3:0]:

this register is written during Switch Processor writes to the TX_FIFO data ram with the **number** of bytes that are valid in the next data-word that is written to the data ram via writes to the TXF_DATAL and TXF_DATAH control register. The write to the data ram occurs on the cycle following the write to the TXF_DATAH register. Therefore the TXF_VALID register must be set with the correct number of bytes in the data-word before the TXF_DATAH register is written during Switch Processor frame writes. **Only the last data-word in a frame may have a WVALID[3:0] value that is less than or equal to 4'h4. Legal values of WVALID[3:0] are 4'h1, 4'h2, 4'h3, and 4'h4.**

Register 0x4B (cr43): Packet buffer size (in bytes)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	PBSIZE[7:0]	d80

PBSIZE[7:0]:

the value of PBSIZE[7:0] is interpreted by the hardware to be the size, in bytes, of the Packet Buffer in the local memory of the PM3351 device.

This number is a constant that may be set to a value other than the default value of decimal 80 during initialization. One restriction is that the size must be an integer mulitple of 4.

Register 0x4C (cr44): Packet buffer size (in bytes)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	RPBSIZE[7:0]	d80

RPBSIZE[7:0]:

the value of RPBSIZE[7:0] is interpreted by the hardware to be the size, in bytes, of the Packet Buffer that is to be read across the PCI expansion bus as part of the frame transfer protocol.

This number is a constant that may be set to a value other than the default value of decimal 80 during initialization. One restriction is that the size must be an integer mulitple of 4.

DPO funtional block (DMA transfer handshake channel)

The DPO functional block is the name of the hardware that supports the eight Transfer rings that are associated with each of the eight possible devices in an ELAN 1x100 or ELAN 8x10 system. The Switch Processor sets up the linked list of blank Data Descriptor structures that are used for the transfer rings at initialization and then the DPO channel is responsible for:

transfer ring management (read and write pointer)

generating the master write commands (all single word bus transactions) that **increment** the request and acknowledge counters on ELAN-compatible chips over the PCI expansion bus

the Switch Processor writesa template Data Descriptor to control registers in the DPO module. The DPO channel is responsible for replicating this Data Descriptor in local memory in the transfer ring for each chip to which the frame is to be switched.

de-allocation of the Data Descriptor associated with a frame that has been acknowledged.

The DPO functional block does not in any way modify or interpret the template Data Descriptor that is written by the Switch Processor to the control registers.

Register 0x4D (cr45): DPO: PCI increment counter write data, low (IPCIDATAL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IPCIDATA[15:0]	0

Register 0x4E (cr46): DPO: PCI increment counter write data, high (IPCIDATAH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IPCIDATA[31:16]	0

IPCIDATA[31:0]:

the contents of this register pair is used as the data-word during the PCI bus transaction that is used by the PM3351 for the single-word writes to the request and acknowledge counters on the PCI expansion bus.

The present implementation of the ELAN protocol does not make use of the actual data that is in the data-word during the PCI bus transaction that increments the request and acknowledge counters. This register is provided to allow for extensions to the basic protocol in special applications.

Register 0x4F (cr47): DPO: transfer handshake channel chip mask (ICHIP_MASK)

Bit	Type	Function	Default
Bit 15 to 8	R	DPO_AF[7:0]	0
Bit 7 to 0	R/W	ICHIP_MASK[7:0]	0

DPO_AF[7:0]:

almost-full flag for each of the eight transfer rings (with the bit index corresponding to the queue number). Almost full occurs when the number of valid elements in the transfer ring is either greater than or equal to 1 less than the size of the ring.

These register bits are provided for observability and should not be required for use by the Switch Processor during normal frame switching.

ICHIP_MASK[7:0]:

this chip mask is written by the Switch Processor to indicate to the DPO hardware the transfer rings on which the frame is to be appended. Each bit index of the mask corresponds to the transfer ring number. So a unicast frame will have one bit set (1) in ichip_mask[7:0] whereas a broadcast frame will have (N-1) bits set in ichip_mask[7:0], where N is the total number of ELAN-compatible chips attached to the PCI expansion bus.

Register 0x50 (cr48): DPO: transfer handshake channel Data Descriptor word 0, bits [15:0] (IDD0L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IDD0[15:0]	0

IDD0[15:0]:

template Data Descriptor word #0, bits [15:0], written by Switch Processor.

Register 0x51 (cr49): DPO: transfer handshake channel Data Descriptor word 0, bits [23:16] (IDD0H)

Bit	Type	Function	Default
Bit 15 to 12	R	S[3:0]	0
Bit 11 to 8		Unused	0
Bit 7 to 0	R/W	IDD0[23:16]	0

S[3:0]:

DPO functional block state.

IDD1[23:16]:

template Data Descriptor word #0, bits [23:16], written by Switch Processor.

Register 0x52 (cr50): DPO: transfer handshake channel Data Descriptor word 1, bits [15:0] (IDD1L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IDD1[15:0]	0

IDD1[15:0]:

template Data Descriptor word #1, bits [15:0], written by Switch Processor.

Register 0x53 (cr51): DPO: transfer handshake channel Data Descriptor word 1, bits [23:16] (IDD1H)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD1[23:16]	0

IDD1[23:16]:
template Data Descriptor word #1, bits [23:16], written by Switch Processor.

Register 0x54 (cr52): DPO: transfer handshake channel Data Descriptor word 2, bits [15:0] (IDD2L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IDD2[15:0]	0

IDD2[15:0]:
template Data Descriptor word #2, bits [15:0], written by Switch Processor.

Register 0x55 (cr53): DPO: transfer handshake channel Data Descriptor word 2, bits [23:16] (IDD2H)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD2[23:16]	0

IDD2[23:16]:
template Data Descriptor word #2, bits [23:16], written by Switch Processor.

Register 0x56 (cr54): DPO: transfer handshake channel Data Descriptor word 3, bits [15:0] (IDD3L)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IDD3[15:0]	0

IDD3[15:0]:
template Data Descriptor word #3, bits [15:0], written by Switch Processor.

Register 0x57 (cr55): DPO: transfer handshake channel Data Descriptor word 3, bits [23:16] (IDD3H)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD3[23:16]	0

IDD3[23:16]:
template Data Descriptor word #3, bits [23:16], written by Switch Processor.

Register 0x58(cr56): DPO: transfer handshake channel Data Descriptor word 0, bits [31:24] (IDD0FLAG)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD0[31:24]	0

IDD0[31:24]:
template Data Descriptor word #0, bits [31:24], written by Switch Processor.

Register 0x59(cr57): DPO: transfer handshake channel Data Descriptor word 1, bits [31:24] (IDD1FLAG)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD1[31:24]	0

IDD1[31:24]:
template Data Descriptor word #1, bits [31:24], written by Switch Processor.

Register 0x5A(cr58): DPO: transfer handshake channel Data Descriptor word 2, bits [31:24] (IDD2FLAG)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD2[31:24]	0

IDD2[31:24]:
template Data Descriptor word #2, bits [31:24], written by Switch Processor.

Register 0x5B(cr59): DPO: transfer handshake channel Data Descriptor word 3, bits [31:24] (IDD3FLAG)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	IDD3[31:24]	0

IDD3[31:24]:
template Data Descriptor word #3, bits [31:24], written by Switch Processor.

Register 0x5C(cr60): DPO: transfer handshake channel control/status register (IC_CTRL)

Bit	Type	Function	Default
Bit 15	R	INCR_REQ_OP	0
Bit 14	R	INCR_ACK_OP	0
Bit 13	R	DPI_INCR_ACK_ OP	0
Bit 12	R	PENDING_REQ	0
Bit 11	R	RING_FULL	0
Bit 10 to 9	R	Unused	0
Bit 8 to 7	R/W	IC_TEST_EN[1:0]	0
Bit 6 to 3	R/W	REGTYPE[3:0]	0
Bit 2 to 0	R/W	RINGNUM[2:0]	0

INCR_REQ_OP:
status bit set if DPO is actively performing an operation to increment the request counter across the PCI expansion bus.

INCR_ACK_OP:
status bit set if DPO is actively performing an operation to increment the request counter across the PCI expansion bus and it was the **Switch Processor** that initiated the operation.

DPI_INCR_ACK_OP:
status bit set if DPO is actively performing an operation to increment the request counter across the PCI expansion bus and it was the **DPI functional block** that initiated the operation.

In normal operation the firmware will usually only select one method for initiating the increment of the acknowledge counter. Having the DPI module initiate the acknowledge operation is more efficient.

PENDING_REQ:

this is an internal state bit that is set (1) if the Switch Processor wanted to initiate an increment request counter operations but the DPO state machine was not presently in state IDLE (which would occur if the DPO module was doing an increment acknowledge operation that was initiated by the DPI functional block). Present for observability.

RING_FULL:

this status bit is set (1) if any of the eight possible transfer rings are full.

IC_TEST_EN[1:0]:

these two register bits are for factory test and should not be set during normal operation.

REGTYPE[3:0]:

windowed address "type" field. This is the address that is used for accessing one of ten possible windowed registers in the DPO module.
4'd0: IC_MWEN[15:0]
4'd1: CHIPBASE[7:0] (one per transfer ring)
4'd2: RBASE[23:8] (one per transfer ring)
4'd3: RSIZE[3:0] (one per transfer ring)
4'd4: WPTR[15:0] (one per transfer ring)
4'd5: RPTR[15:0] (one per transfer ring)
4'd6: IRCOUNTL[15:0]
4'd7: IRCOUNTH[7:0]
4'd8: IACOUNTL[15:0]
4'd9: IACOUNTH[7:0]

RINGNUM[2:0]:

windowed register ring select. For windowed registers that are implemented on a per Transfer ring basis, the value of this register selects the ring to be accessed.

DPO windowed register description

There are a total of 45 windowed registers that can be accessed in the DPO module: 40 of those registers are implemented as a set of 5 registers per transfer ring while 5 of those registers are implemented in the top-level DPO functional block. The DPO functional block windowed register address is decoded from the REGTYPE[3:0] field of the IC_CTRL register while the ring number is decoded from the RINGNUM[2:0] field of the IC_CTRL register. The IC_DATA register is used as the data register for access to the DPO windowed registers.

For example, if RINGNUM is 3'h5 and REGTYPE[3:0] is 4'h3 then the RSIZE[3:0] register of ring #5 is can be accessed via the IC_DATA control register.

Following is the description of the windowed registers.

Bit	Type	Function	Default
Bit 15 to 0	R/W	IC_MWEN[15:0]	0

IC_MWEN[15:0 (DPO windowed)]:

write enable mask that is used for writing out the template Data Descriptor (which is in IDD0[31:0] to IDD3[31:0]. Each bit is used to qualify one byte in the Data Descriptor. Bit index 0 corresponds to IDD0[7:0] and bit index 15 corresponds to IDD3[31:24].

Bit	Type	Function	Default
Bit 15 to 8	R	Unused	0
Bit 7 to 0	R/W	CHIPBASE[7:0]	0

CHIPBASE[7:0] (DPO windowed: one per transfer ring):

this is the slave base address (that is, the value of AD[31:24] on the PCI bus during the address phase) of the device corresponding to this transfer ring. Set during initialization by the Switch Processor and then not modified.

Bit	Type	Function	Default
Bit 15 to 0	R/W	RBASE[23:8]	0

RBASE[23:8] (DPO windowed: one per transfer ring):

pointer in local memory to the base of the Transfer ring. The low 8 bits of the base address are hardwired to 8'h0.

Bit	Type	Function	Default
Bit 15 to 4	R	Unused	0
Bit 3 to 0	R/W	RSIZE[3:0]	0

RSIZE[3:0] (DPO windowed: one per transfer ring):
size of the Transfer ring. This register value must be static during switch operation and is initialized by the Switch Processor.

value	Size of Ring
4'd0	32
4'd1	64
4'd2	128
4'd3	256
4'd4	512
4'd5	1024 (1K)
4'd6	2048 (2K)
4'd7	4K
4'd8	8K
4'd9	16K
4'd10	32K
4'd11	64K
4'd12	64K
4'd13	64K
4'd14	64K
4'd15	64K

where size of the ring is given in terms of the number of Data Descriptors that are in the ring (which also corresponds to the maximum number of frames that can be queued up for transmit to the device to which this ring is assigned).

Bit	Type	Function	Default
Bit 15 to 0	R/W	WPTR[15:0]	0

WPTR[15:0] (DPO windowed: one per transfer ring):
this register holds the pointer in local memory that is used for writing Data Descriptors. In normal switch operation it is not accessed by the Switch Processor.

Bit	Type	Function	Default
Bit 15 to 0	R/W	RPTR[15:0]	0

RPTR[15:0] (DPO windowed: one per transfer ring):

this register holds the pointer in local memory that is used for reading Data Descriptors. In normal switch operation it is not accessed by the Switch Processor.

Bit	Type	Function	Default
Bit 15 to 0	R/W	IRCOUNT[15:0]	0

IRCOUNT[15:0] (DPO windowed):

Bit	Type	Function	Default
Bit 15 to 8	R	Unused	0
Bit 7 to 0	R/W	IRCOUNT[23:16]	0

IRCOUNT[23:16] (DPO windowed):

this register holds the low 24-bits of the PCI address that is to be used during an incrementing request counter operation. The high eight bits of the PCI address are taken from the rbase[7:0] register of the applicable transfer ring.

Bit	Type	Function	Default
Bit 15 to 0	R/W	IACOUNT[15:0]	0

IACOUNT[15:0] (DPO windowed):

Bit	Type	Function	Default
Bit 15 to 8	R	Unused	0
Bit 7 to 0	R/W	IACOUNT[23:16]	0

IACOUNT[23:16] (DPO windowed):

this register holds the low 24-bits of the PCI address that is to be used during an incrementing acknowledge counter operation. The high eight bits of the PCI address are taken from the rbase[7:0] register of the applicable transfer ring.

Register 0x5D (cr61): DPO: ring data access register, bits (IC_DATA)

Bit	Type	Function	Default
Bit 15 to 0	R/W	IC_DATA[15:0]	0

IC_DATA[15:0]:

this is the data register used for access to the windowed registers in the DPO functional block. The access to this register by the Switch Processor determines whether the register access is a read or a write. If the Switch Processor writes this register (via the LDI or MOVE instructions) the write will propagate to the selected windowed register in the following cycle.

Register 0x5E (cr62): TX_FIFO: disposition FIFO, tail register (DSTAIL)

Bit	Type	Function	Default
Bit 15 to 14	R	Unused	0
Bit 13 to 8	R	DSCNTR[5:0]	0
Bit 7 to 4	R	Unused	0
Bit 3 to 0	R/W	DSTAIL[3:0]	0

DSCNTR[5:0]:

number of valid entries in the TX_FIFO disposition FIFO.

DSTAIL[3:0]:

the Switch Processor writes this register with the disposition code of the frame that is being written into the TX_FIFO data ram.

Register 0x5F (cr63): TX_FIFO: disposition FIFO, head register (DSHEAD)

Bit	Type	Function	Default
Bit 15 to 14	R	Unused	0
Bit 13 to 8	R	FRAMECNTR[5:0]	0
Bit 7 to 4	R	Unused	0
Bit 3 to 0	R/W	DSHEAD[3:0]	0

FRAMECNTR[5:0]:

number of frames that are in the TX_FIFO data ram.

During normal switch operation, this counter is incremented when the DPI module finishes reading a frame across the PCI expansion bus and writing it out to the TX_FIFO data ram. The counter is decremented after the frame is completed at the head of the FIFO, either by being successfully transmit on

the MAC link media or by being dropped from the data ram (which is only done under control of the Switch Processor).

DSHEAD[3:0]:

this is the disposition code of the frame that is at the head of the TX_FIFO data ram (that is, it is the code of the frame that either has been or will be read out of the TX_FIFO data ram). This code is loaded by the TX_FIFO hardware from the internal disposition FIFO when it has been determined that a frame can be read out of the TX_FIFO data ram. If the code in the register is mapped to a STOP disposition code, an interrupt (TXF_STOP) will be issued to the Switch Processor. The Switch Processor can write this register with a code corresponding to TRANSMIT or DROP after it has completed its processing the frame.

NOTE- this register can be written by the Switch Processor only if the TXF_STOP interrupt line is asserted; writing to this register at other times may cause an unrecoverable transmit error.

Register 0x60 (cr64): Link constant register address (LC_SEL)

Bit	Type	Function	Default
Bit 15 to 4	R	Unused	0
Bit 3 to 0	R/W	LC_SEL[3:0]	0

LC_SEL[3:0]:

selects the address of the windowed register in the Link functional block that is to be accessed via LC_DATA.

Value	Register Accessed
4'd0	IFS1_TIME
4'd1	IFS2_TIME
4'd2	PRE_SIZE
4'd3	JAM_SIZE
4'd4	MAX_SIZE
4'd5	MIN_SIZE
4'd6	BLIND_TIME
4'd7	LBK_BYTE
4'd8	LNSTAT2
4'd9	ETYPE
4'd10	LATE_TIME
4'd11	TXCOUNT

Register 0x65 (cr65): Link constant register data (LC_DATA)

Bit	Type	Function	Default
Bit 15 to 0	R/W	LC_DATA[15:0]	0

LC_DATA[15:0]:
this is the data register used for access to the windowed registers in the Link functional block. The access to this register by the Switch Processor determines whether the register access is a read or a write. If the Switch Processor writes this register (via the LDI or MOVE instructions) the write will propagate to the selected windowed register in the following cycle.

Link_windowed register description

There are a total of 11 windowed registers that can be accessed in the Link functional block. Each of these 11 registers holds data that is expected to be static (that is constant) during switch operation; they are expected to be accessed by the Switch Processor only during initialization. The Link functional block windowed register address is decoded from the LC_SEL register while the LC_DATA register is used as the data register.

Following is the description of the windowed registers in the Link functional block.

IFS1_TIME (Link windowed register):

Bit	Type	Function	Default
Bit 15 to 8	R	Unused	0
Bit 7 to 0	R/W	IFS1_TIME_REG[7:0]	8'd13

IFS1_TIME_REG[7:0]:
the value of this register determines the Interframe Spacing, Part 1 (IFS1) that is used for transmission on the Link media. The units of this register is in terms of **nibble_times** (that is, TX_CLK cycles).

The actual value of IFS1, in bit times is given by:
IFS1 (bit times) = 4*(IFS1_TIME_REG + 2)

IFS2_TIME (Link windowed register):

Bit	Type	Function	Default
Bit 15 to 8	R	Unused	0
Bit 7 to 0	R/W	IFS2_TIME_REG[7:0]	8'd6

IFS2_TIME_REG[7:0]:

the value of this register determines the Interframe Spacing, Part 2 (IFS2) that is used for transmission on the Link media. The units of this register is in terms of **nibble-times** (that is, TX_CLK cycles).

The actual value of IFS2, in bit times is given by:

IFS2 (bit times) = 4*(IFS2_TIME_REG + 3)

PRE_SIZE (Link windowed register):

Bit	Type	Function	Default
Bit 15 to 8	R	Unused	0
Bit 7 to 0	R/W	PRE_SIZE[7:0]	8'd52

PRE_SIZE[7:0]:

this register gives the number of bits of preamble that is used during transmit on the Link media. The units are in **bit-times.** The value in this register must be set so that bits [1:0] are equal to 0.

The actual number of preamble bits during transmission excluding the StartFrameDelimiter is equal to:

num_preamble_bits = 4*(PRE_SIZE + 1)

NOTE- the transmit link state machine always is guaranteed to insert a valid one-byte StartFrameDelimiter.

JAM_SIZE (Link windowed register):

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**. So the default setting corresponds to a jam of 32 bits.

MAX_SIZE[11:0] (Link windowed register):

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.

For transmit, if a frame in the TX_FIFO data ram has more bytes than given by MAXSIZE the frame will be truncated to be exactly MAXSIZE bytes and the TXLONG status bit in the TXF_DSIZE register will be set (1).

For receive, frames that are received on the MAC link interface that are greater in size than MAXSIZE will be truncated to exactly MAXSIZE bytes (which will be written out to local memory in a linked-list of PacketBuffers as is done for non-errored receive frames), the RXLONG status bit in the DMSTAT1 register will be set (1), and the receive frame error interrupt line (RXERR_INT) will be asserted to the Switch Processor.

MIN_SIZE[7:0] (Link windowed register):

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.

This register is not used for transmit frames.

If a frame is received on the MAC link media with a frame size of less than MINSIZE number of bytes the RXSHORT bit in the DMSTAT1 register will be set (1), and the receive frame error interrupt line (RXERR_INT) will be asserted to the Switch Processor.

BLIND_TIME (Link windowed register):

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	BLIND[7:0]	0

BLIND[7:0]:

Blind time in **nibble-times**. This is the number of RX_CLK times for which the link receiver is “blinded” after a transmit.

The blind timer is not qualified by the duplex mode setting of the Link (so for full-duplex operation this register is expected to be set to 0 by the Switch Processor).

LBK_BYTE (Link windowed register):

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 the link as the transmission data when in loopback mode. This data byte is continuously transmit as the data after a valid preamble and StartFrameDelimiter has been sent.

LNSTAT2 (Link windowed register):

Bit	Type	Function	Default
Bit 15	R	STICKY_TXLATECOL	0
Bit 14	R	STICKY_RXFIFO_OVRUN	0
Bit 13	R	STICKY_LRFIFO_OVRUN	0
Bit 12 to 8	R	LINKRWORDS[4:0]	0
Bit 7 to 5	R	RX_IF_SM[2:0]	0
Bit 4 to 0	R	TX_IF_SM[4:0]	0

STICKY_TXLATECOL:

sticky error flag- late collision experienced during frame transmit.

A late collision is one in which the link is configured for half-duplex and the COL signal is asserted after more than a user-configurable number of bytes

of frame data have been framed on the MII interface pins (see LATE_TIME Link windowed register).

NOTE- three sticky error flags are implemented. The flags can be cleared by setting the CLR_STICKY bit in the LNCTRL register.

STICKY_RXFIFO_OVRUN:
sticky error flag- FIFO overrun occurred on the RXFIFO of the Link receiver functional block.

STICKY_LRFIFO_OVRUN:
sticky error flag- FIFO overrun occurred on the LRF FIFO of the Link receiver functional block.

LINKRWORDS[4:0]:
this is the number of valid words in the Link LRF FIFO. Present for observability.

RX_IF_SM[2:0]:
value of receive interface state machine (present for observability).

TX_IF_SM[4:0]:
value of transmit interface state machine (present for observability).

ETYPE (Link windowed register):

Bit	Type	Function	Default
Bit 15 to 0	R/W	ETYPE[15:0]	0x8808

ETYPE[15:0]
the value of this register is compared to the value of the Ethernet length/type field of a received frame. If the values match (that is, all bits are equal) then the receive frame status bit CTRL_FRAME is asserted: this status bit is mapped in the PM3351 to one of the GPI lines of the Switch Processor.

LATE_TIME (Link windowed register):

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 **bytes** after the start of a frame after which a collision is

considered late.

In full duplex operation the contents of this register have no effect.

TXCOUNT (Link windowed register):

Bit	Type	Function	Default
Bit 15 to 0	R	TXCOUNT[15:0]	0

TXCOUNT[15:0]:
transmit counter value. This counter is in the Link transmit interface functional block and is present for testing purposes only.

HASH functional block

The HASH functional block on the PM3351 is capable of storing the results of three separate MAC address hash searches. The result of each search and the pointers to the hash buckets can be read by the Switch Processor by accessing control registers. The MAC address keys are also readable and writeable by the Switch Processor (although in normal operation they will not be written by the Switch Processor but by other internal hardware blocks that delimit the MAC addresses from the Ethernet frame). Fail and busy status bits are supplied for each result and are mapped to separate GPI bits of the Switch Processor. On the PM3351 a hash search is done on the following three MAC addresses:

MAC source address on Link receive frames (search always done)

MAC destination address on Link receive frames (search always done)

MAC source address on frames read across the PCI expansion bus (search is initiated by the DPI logic if and only if the LEARN bit in the Data Descriptor flags field is set).

In normal switch operation a hash search occurs as follows. The DMA logic delimits the 48-bit MAC address in the Ethernet frame and writes it to the MAC address control register. The 48-bit MAC address is logically XOR'ed and AND'ed with a mask register to form a 16-bit hash index:

hash_index[15:0] = HMASK[15:0] & (mac_addr[47:32] ^ mac_addr[31:16] ^ mac_addr[15:0]);

The 16-bit hash index is used as an index into the hash array, where HMASK is a control register which is set according to the size of the hash table. The hash search begins after the full MAC address has been written to the control registers. During the hash search the linked-list of hash buckets and/or forwarding tags is scanned until

either the end-of-list is reached or a match is made on the MAC address. While the hash search is in progress a busy status bit is asserted.

After the hash search is complete, the busy signal is de-asserted and the pointer to the matching hash data structure is loaded to the applicable result control register. Flow control is provided to the Switch Processor via hardware so that for each of the three possible MAC address searches only one hash search will be active (for example, there will at most be one MAC source address hash search busy; the DMA will not attempt to initiate a second search until the previous search has completed and the Switch Processor has read the applicable control registers for the hash search).

The Switch Processor can read the result of the hash search by reading the applicable hash result register. The pass/fail status on the search is available for the Switch Processor by testing a GPI line. Each hash result register returns a different value depending upon whether the hash search passed or failed:

a pass corresponds to a match in the MAC address. The hash result register returns a pointer in local memory to the hash bucket or forwarding tag that has the 48-bit MAC address equal to the address in the MAC address key.

a fail corresponds to the situation that the entire linked-list of hash buckets and/or forwarding tags corresponding to the hash array was traversed and the end-of-list was reached. The hash result register returns the pointer in local memory to the first node in the hash array for this index to which a new address can be attached (that is, it is the end of the linked-list for this hash index).

Register 0x62 (cr66): HASH logic control/status register (HASHCTRL)

Bit	Type	Function	Default
Bit 15	R	IDLE	1
Bit 14	R/W	EN_IDLE	0
Bit 13 to 3		Unused	0
Bit 2	R/W	HCTRL_HPSEN	0
Bit	1	R/W	HDSEN
0			
Bit 0	R/W	HDDEN	0

IDLE:

HASH functional state machine in IDLE state.

EN_IDLE:

if this bit is set (1) then the HASH state machine will be prevented from leaving the IDLE state.

HCTRL_HPSEN:

enable bit for allowing the Switch Processor to initiate a hash search on the MAC source address for a frame read across the PCI expansion bus.

HDSEN:

enable bit for allowing a hash search on the MAC source address for a frame received on the Link interface.

HDDEN:

enable bit for allowing a hash search on the MAC destination address for a frame received on the Link interface.

In normal switch operation the firmware will leave the EN_IDLE and HCTRL_HPSEN bits in their default state (0), and will set the HDSEN and HDDEN bits (1).

Register 0x63 (cr67): HASH mask register (HMASK)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HMASK[15:0]	0

HMASK[15:0]:

the contents of this register is used as a mask in the generation of the 16-bit hash index.

Register 0x64 (cr68): HASH array base, address [15:0] (HBASEL)

Bit	Type	Function	Default
Bit 15 to 2	R/W	HBASE[15:2]	0
Bit 1 to 0	R	HBASE[1:0]	0

Register 0x65 (cr69): HASH array base, address [23:16] (HBASEH)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	HBASE[23:16]	0

HBASE[23:0]:

this is the base address of the hash pointer array in local memory. The low two bits, indices 1:0, are hardwired to 2'h0.

Register 0x66 (cr70): Hash result register: DPI source address look-up, low (HDPISRCRL)

Bit	Type	Function	Default
Bit 15 to 2	R/W	HDPISRCR[15:2]	0
Bit 1 to 0	R	HDPISRCR[1:0]	0

Register 0x67 (cr71): Hash result register for DPI source address look-up, high (HDPISRCRH)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	HDPISRCR[23:16]	0

HDPISRCR[23:0]:

hash result register for MAC source address on frames read across the PCI expansion bus (via DPI functional block).

Register 0x68 (cr72): Hash result register: Link RX frame destination address look-up, low (HDMADSTRL)

Bit	Type	Function	Default
Bit 15 to 2	R/W	HDMADSTR[15:2]	0
Bit 1 to 0	R	HDMADSTR[1:0]	0

Register 0x69 (cr73): Hash result register: Link RX frame destination address look-up, high (HDMADSTRH)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	HDMADSTR23:16]	0

HDMADSTR[23:0]:

hash result register for MAC destination address on Link receive frames.

Register 0x6A (cr74): Hash result register: Link RX frame source address look-up, low (HDMASRCRL)

Bit	Type	Function	Default
Bit 15 to 2	R/W	HDMASRCR[15:2]	0
Bit 1 to 0	R	HDMASRCR[1:0]	0

Register 0x6B (cr75): Hash result register: Link RX frame source address look-up, high (HDMASRCRH)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	HDMASRCR23:16]	0

HDMASRCR[23:0]:
hash result register for MAC source address on Link receive frames.

Register 0x6C (cr76): Hash DPI source MAC address, low (HDPISRCL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDPISRC[15:0]	0

Register 0x6D (cr77): Hash DPI source MAC address, middle (HDPISRCM)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDPISRC[31:16]	0

Register 0x6E (cr78): Hash DPI source MAC address, high (HDPISRCL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDPISRC[47:32]	0

HDPISRC[47:0]:
this is the 48-bit address key corresponding to the **MAC source address** for frames read across the PCI expansion bus.

In normal switch operation this register is written by the DPI functional block as the frame is read across the PCI expansion bus.

Register 0x6F (cr79): Hash Link RX destination MAC address, low (HDMADSTL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDMADST[15:0]	0

Register 0x70 (cr80): Hash Link RX destination MAC address, middle (HDMADSTM)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDMADST[31:16]	0

Register 0x71 (cr81): Hash Link RX destination MAC address, high (HDMADSTH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDMADST[47:32]	0

HDMADST[47:0]:

this is the 48-bit address key corresponding to the **MAC destination address** for frames received on the Link interface.

In normal switch operation this register is written by the DMA functional block as the frame is being written out to a Packet Buffer in local memory.

Register 0x72 (cr82): Hash Link RX source MAC address, low (HDMASRCL)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDMASRC[15:0]	0

Register 0x73 (cr83): Hash Link RX source MAC address, middle (HDMASRCM)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDMASRC[31:16]	0

Register 0x74 (cr84): Hash Link RX source MAC address, high (HDMASRCH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	HDMASRC[47:32]	0

HDMASRC[47:0]:

this is the 48-bit address key corresponding to the **MAC source address** for frames received on the Link interface.

In normal switch operation this register is written by the DMA functional block as the frame is being written out to a Packet Buffer in local memory.

Register 0x75 (cr85): Link control register (LWCTRL)

Bit	Type	Function	Default
Bit 15 to 11	R/W	Unused	0
Bit 10 to 9	R/W	TEST_BACKOFF[1:0]	0
Bit 8	R/W	Unused	0
Bit 7	R/W	FDPLX	1
Bit 6	R/W	CLR_STICKY	0
Bit 5	R/W	FORCE_JAM	0
Bit 4	R/W	TXDISABLE	1
Bit 3	R/W	RXDISABLE	1
Bit 2	R/W	RXFLUSH	0
Bit 1	R/W	Unused	0
Bit 0	R	TX_ENABLE	0

TEST_BACKOFF[1:0]:

this register is implemented for factory test. It should be left in its default state otherwise.

FDPLX:

full duplex mode bit. If set (1) the Link functional block is configured for full-duplex operation; if clear (0) the Link is configured for half-duplex.
If the Link is configured for full-duplex operation the MII signals COL and CRS are qualified internally in the PM3351 to be logically de-asserted.

When changing the duplex mode from full-duplex to half-duplex the Link functional block should be reset using the FRST register.

CLR_STICKY:

when set (1) this bit clears the three sticky link flags: STICKY_TXLATECOL, STICKY_RXFIFO_OVRUN, STICKY_LRFIFO_OVRUN.

FORCE_JAM:

when this bit is set (1) the transmit link interface will continuously transmit a nibble pattern of 4'h5 on the MII TXD[3:0] pins and have TX_EN asserted. The jam pattern is sent regardless as to the state of COL or CRS and the setting of the duplex mode. The jam pattern is stopped being sent when the FORCE_JAM bit is cleared (0).

TXDISABLE:

if this bit is set (1) then frame transmission will be halted on a frame boundary

(so if the Link is presently attempting to transmit a frame then frame transmission will be halted at the end of the present attempt, regardless as to whether a collision occurs). Frame transmission will be re-started only after the TXDISABLE bit is cleared.

RXDISABLE:

if this bit is set (1) then frame reception is halted on a frame boundary. Frame reception will be re-started only after the RXDISABLE bit is cleared.

Since a partial frame may have been received while the RXDISABLE bit was set, the RXFLUSH bit can be set to flush the partial frame from the link receive FIFO interface and RXDISABLE and RXFLUSH can then be simultaneously de-asserted.

RXFLUSH:

if this bit is set (1) then the Link receive interface is synchronously reset. This bit must be clear (0) to enable frame reception.

Register 0x76 (cr86): Link status register (LWSTAT)

Bit	Type	Function	Default
Bit 15	R	COLLISION	0
Bit 14	R	TXLATECOL	0
Bit 13	R	LT_TXLONG	0
Bit 12	R	TFRAME	0
Bit 11	R	TXLATEZONE	0
Bit 10	R	TX_ENABLE	0
Bit 9	R	TX_READY	0
Bit 8	R	LINKRFRAME	0
Bit 7	R	LRF_HAS_LAST WORD	0
Bit 6	R	RECEIVING	0
Bit 5	R	RX_READY_S	0
Bit 4	R	LR_CTRL_FRAM E	0
Bit 3	R	LR_FIFO_OVRUN	0
Bit 2	R	LR_RXLONG	0
Bit 1	R	LR_RXSHORT	0
Bit 0	R	LR_MISALIGN	0

COLLISION:

this is the value of the collision interrupt line.

TXLATECOL:
late collision occurred during transmit. Valid when the collision interrupt is asserted.

LT_TXLONG:
Link transmitting a frame having a size greater than the value of the MAX_SIZE Link control register. Valid if TFRAME is asserted. This bit is latched in the TXF_DSIZE register (which is valid when the TX_DONE interrupt is asserted).

TFRAME:
transmitting frame on the MII interface.

TXLATEZONE:
this bit is set (1) if a frame is being transmit and the frame is in the late collision zone (see Link control register LATE_COL).

TX_ENABLE:
transmit enable flow control bit (for observability).

TX_READY:
transmit enable flow control bit (for observability).

LINKRFRAME:
internal framing signal used by Link receiver to indicate to the DMA functional block that a receive frame is in the LRF FIFO.

LRF_HAS_LASTWORD:
internal status signal- set (1) when the LRF FIFO has the last word of a received frame.

RECEIVING:
the Link RX state machine is in a state corresponding to frame reception.

RX_READY_S:
the RX FIFO (on the MII interface) is non-empty.

LR_CTRL_FRAME:
receive frame status: a frame is being received having an Ethernet Length/Type field of the same value as the ETYPE Link control register. Status signal latched and held in DMSTAT control register.

LR_FIFO_OVRUN:
receive frame status: a FIFO over-run has occured in the Link receiver during reception of this frame. Status signal latched and held in DMSTAT control register.

LR_RXLONG:

receive frame status: a frame is being received with a length greater than that of the MAX_SIZE Link control register value. Status signal latched and held in DMSTAT control register.

LR_RXSHORT:

receive frame status: a frame is being received with a length less than that of the MIN_SIZE Link control register value. Status signal latched and held in DMSTAT control register.

LR_CRC_ERR:

receive frame status: a frame is being received having an incorrect 802.3 FrameCheckSequence field (that is, a CRC error).. Status signal latched and held in DMSTAT control register.

LR_MISALIGN:

receive frame status: this bit is set (1) if an odd number of framed data nibbles have been received on the MII receive interface and the the framing signal (that is, the MII signal RX_DV) has become deasserted. Status signal latched and held in DMSTAT control register.

Register 0x77 (cr87): Link backoff counter (LWBACK)

Bit	Type	Function	Default
Bit 15 to 10		Unused	0
Bit 9 to 0	R/W	BACKOFF[9:0]	0

BACKOFF[9:0]:

Link backoff register. This register is in units of slot times in which to backoff. This register is pre-loaded by the Switch Processor with the next backoff time to be used. The Link transmit logic loads the value of this register into the TXCOUNT counter after the collision jam sequence has been sent (so half-duplex mode only).

Register 0x78 (cr88): Number of PacketBuffers (LWNBUFS)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	LWNBUFFS[7:0]	0

LWNBUFFS[7:0]:

the number of PacketBuffers in the receive frame. Valid when either receive frame interrupt is asserted to the Switch Processor (RX_FRAME or

RX_ERR_FRAME). In normal operation, this register is written by the DMA and read by the Switch Processor.

Register 0x79 (cr89): First PB address low (LWFIRSTL)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	LWFIRST[23:16]	0

Register 0x7A (cr90): First PB address high (LWFIRSTH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	LWFIRST[15:0]	0

LWFIRST[23:0]:

pointer (that is, the local memory address) of the first PacketBuffer in the received frame. Valid during assertion of a receive frame interrupt.

Register 0x7B (cr91): Last PB address low (LWLASTL)

Bit	Type	Function	Default
Bit 15 to 8		Unused	0
Bit 7 to 0	R/W	LWLAST[23:16]	0

Register 0x7C (cr92): Last PB address high (LWLASTH)

Bit	Type	Function	Default
Bit 15 to 0	R/W	LWLAST[15:0]	0

LWLAST[23:0]:

pointer (that is, the local memory address) of the last PacketBuffer in the received frame. Valid during assertion of a receive frame interrupt.

Register 0x7D (cr93): Link receive frame size (LWDSIZE)

Bit	Type	Function	Default
Bit 15 to 12		Unused	0
Bit 11 to 0	R/W	RXSIZE[11:0]	0

RXSIZE:

receive frame byte counter. This is the number of framed data octets on the link media (the MII interface). Valid during assertion of a receive frame interrupt.

Register 0x7E (cr94): DPO: Acknowledge pointer for transfer ring (ACKRPTR)

Bit	Type	Function	Default
Bit 15 to 0	R	ACKRPTR[15:0]	0

ACKRPTR[15:0]:

Acknowledge pointer for transfer ring. The value of this register is used by the Switch Processor to compute the address in local memory of a Data Descriptor during an Acknowledge interrupt service routine. One of eight transfer rings is selected depending on the value of the ACKID_NUM[2:0] field in the ACKID control register. The ACKRPTR[15:0] register returns the read pointer of the selected transfer ring.

Register 0x7F (cr95): TX_FIFO read size register (TXF_RSIZE)

Bit	Type	Function	Default
Bit 15 to 11		Unused	0
Bit 10 to 0	R/W	TXF_RSIZE[10:0]	0.

TXF_RSIZE[10:0]:

this register holds the number of bytes that are being read out of either the TX FIFO data ram or the TX FIFO Mac control frame ram.

For data frames (that is, those being read out of the TX FIFO data ram) that are transmit on the Link media without collision the value of this register is latched and held in the TXF_DSIZE register (which is valid during assertion of the TX_DONE interrupt).

If a collision occurs during half-duplex transmission, then TXF_RSIZE holds the number of bytes of data read out before the Link transmit interface sensed the assertion of the MII COL signal.

If the TXF_STOP interrupt is asserted then this register holds the number of valid data bytes in the frame.

Interrupts to Swtich Processor

The interrupt lines that are mapped to the Switch Processor on the PM3351 are prioritized in terms of the interrupt number, with Int 0 (RESET_VECTOR) being the highest priority and Int 15 (ALARM2) the lowest priority.

Switch Processor Interrupt		Memory Location	INTR bus index (RTL code)
Int 0	RESET_VECTOR	0x00	-
Int 1	ALARM1	0x10	-
Int 2	RX_FRAME	0x20	intr[0]
Int 3	RX_ERR_FRAME	0x30	intr[1]
Int 4	TX_DONE	0x40	intr[2]
Int 5	COLLISION	0x50	intr[3]
Int 6	TXF_STOP	0x60	intr[4]
Int 7	RPCIM_DONE	0x70	intr[5]
Int 8	DPI_DONE	0x80	intr[6]
Int 9	DPI_HD_DONE	0x90	intr[7]
Int 10	ACK_INT	0xA0	intr[8]
Int 11	FLIST_ERR	0xB0	intr[9]
Int 12	REQ_INT	0xC0	intr[10]
Int 13	MINTR	0xD0	intr[11]
Int 14	TASK_INTR	0xE0	intr[12]
Int 15	ALARM2	0xF0	-

RESET_VECTOR:

This interrupt vector is taken after assertion of either a hardware (that is, assertion of the RST_ input pin) or a software reset (either by setting the GRES bit of the HCTRL register or because the Watchdog timer timed out).

ALARM1:

asserted when the value in the ALARM1 register equals the value in the CLOCK register. The interrupt is cleared in one of two ways:
* by writing the ALARM1 register to a value different than the current value of the CLOCK register.
* by testing GPI_13.

There is a second method for asserting the ALARM1 interrupt and that is by

setting the GPO line ASSERT_ALARM1 (see GPO description for additional details).

RX_FRAME:

this is one of two possible receive frame interrupts. The interrupt is asserted after a frame has been received on the MII interface, been written out to PacketBuffers in local memory by the DMA, the HASH search on the MAC source and MAC destination addresses has been completed, **and** the frame **did not assert** one of the GPI lines corresponding to:

* RX_CTRL_FRAME
* RX_FIFO_OVRUN
* RXLONG
* RXSHORT
* RX_CRC_ERR
* RX_MISALIGN

DMA functional block asserts this interrupt.
Flip THAW_RXLINK clears the interrupt.

RX_ERR_FRAME:

this is the second possible receive frame interrupt. The interrupt is asserted after a frame has been received on the MII interface, been written out to PacketBuffers in local memory by the DMA (if applicable), **and** the frame **asserted** one of the GPI lines corresponding to:

- RX_CTRL_FRAME
- RX_FIFO_OVRUN
- RXLONG
- RXSHORT
- RX_CRC_ERR
- RX_MISALIGN

If a sufficient number of bytes were received to include the MAC source and/or destination address then the assertion of this interrupt is held off until the HASH search is done on these addresses.

DMA functional block asserts this interrupt.
Flip THAW_RXLINK clears the interrupt.

TX_DONE:

asserted when a frame is done being transmitted on the Link media without collision. This interrupt is conditional on the EN_TX_DONE_INR bit being set (1) in the TXF_CSR control register.

Flip THAW_TXDONE clears the interrupt.

COLLISION:

asserted by the Link transmit interface when a collision is sensed on the MAC link interface (that is, the MII COL signal became asserted while a frame was being transmit by the PM3351). This interrupt will only be asserted if the Link is configured for half-duplex operation. This interrupt becomes asserted when the collision jam sequence begins on the MAC link interface.

Flip THAW_COLLISION clears the interrupt.

TXF_STOP:

asserted by the TX_FIFO functional block when the frame at the head of the TX_FIFO data ram has a disposition code corresponding to STOP.

This interrupt is cleared when the disposition code in the DSHEAD control register is written by the Switch Processor to have a disposition code corresponding either to TRANSMIT or DROP.

RPCIM_DONE:

the RPCIM functional block asserts this interrupt whenever a transaction has completed.
Flip CLR_RPCIM_DONE clears the interrupt.

DPI_DONE:

asserted by the DPI functional block (DMA transmit channel) to indicate that a Data Descriptor and its associated linked-list of PacketBuffers has been read across the PCI expansion bus and the frame data has been written to the TX_FIFO data ram. Assertion of this interrupt is conditional on having DPI_DONE_EN bit in DPI_CTRL register being set (1).

Flip THAW_DPI_DONE clears the interrupt.

DPI_HD_DONE:

asserted by the DPI functional block (DMA transmit channel) to indicate that the “head” of a frame has been read across the PCI expansion bus. The “head” refers to the Data Descriptor and the MAC destination and source address that is in the first PacketBuffer of the frame.

When this interrupt is asserted is conditional on the setting of the LEARN bit

of the Data Descriptor flags field:

- if LEARN bit is clear (0), then the interrupt is asserted when the last byte of the source address is read from the PCI_BIU and is simultaneously written out to the TX_FIFO data ram and the HASH source address register (HDPISRC).
- if the LEARN bit is set (1), then the interrupt is asserted after the HASH search is completed on the MAC source address HDPISRC.

Flip THAW_DPI_HD clears the interrupt.

ACK_INT:
asserted whenever one of the Acknowledge counters on the device is non-zero **and** the DISABLE_ACK_INT of the CTRSEL register is clear (0).

FLIST_ERR:
asserted by the DMA under the following conditions:

- the FLIST_LOCk state bit is clear (0) indicating that the Switch Processor is not accessing the free list
- either of these conditions occurs:
  the DMNFREE control register pair has a value of 0

  –OR-

  the DMNFREE register pair has the same value as the DMFLIST register pair.

After system initialization by the Switch Processor the FLIST_ERR interrupt should not become asserted. This interrupt condition will be treated by firmware as a fatal and unrecoverable error in the free pool list of PacketBuffers.

The FLIST_ERR interrupt will **not** be asserted if the free pool of PacketBuffers becomes empty. This condition, which occurs when DMFLIST becomes equal to 0 (but DMNFREE is non-zero), can be tested by using the FLIST_NULL coprocessor test condition bit.

REQ_INT:

asserted whenever one of the Request counters on the device is non-zero **and all of the following are true:**

***** the GPO line "REQ_LOCK" is clear (0)
* the DISABLE_REQ_INT bit in the CTRSEL register is clear (0)
* the 9-bit hold-off counter (HRICNTR) is non-zero.

The request interrupt is made conditional on REQ_LOCK, DISABLE_REQ_INT, and HRICNTR to allow for possible firmware control in determining the rate at which the request interrupt will occur.

The HRICNTR hold-off counter defaults to 0. It is loaded to a value of 511 when the flip HOLDOFF_REQ_INT occurs. After being loaded, the counter will decrement every SYSCLK cycle until reaching 0.

MINTR:

this interrupt is asserted whenever the MINTR_ pin is logically asserted (that is, has a TTL low logic level).

TASK_INTR:

this interrupt is asserted whenever the TASKCTR control register is non-zero.

ALARM2:

this interrupt is asserted when the value in the ALARM2 register equals the value in the CLOCK register. The interrupt is cleared in one of two ways:
* by writing the ALARM1 register to a value different than the current value of the CLOCK register.
* by testing GPI_14.

There is a second method for asserting the ALARM2 interrupt and that is by setting the GPO line ASSERT_ALARM2 (see GPO description for additional details).

GPI (Switch Processor General Purpose Inputs)

The Switch Processor has fifteen GPI lines that are used to read status provided by the various functional blocks on the PM3351 using the **TEST**, **MTEST, or MTESTB** instructions or by reading the **INCB** special purpose register.

GPI Table:	HW Name	Description
Receive Error Tests:		Used inside the receive frame error interrupt routine to determine what sort of error occurred. The software uses this information to update the appropriate error counters and dispose of the frame and it's associated linked-list of PacketBuffers.
GPI0:	RX_MISALIGN	set (1) if received frame is mis-aligned.
GPI1:	RX_CRC_ERR	set (1) if received frame has a CRC error (technically, that the 32-bit FrameCheckSequence field of the received frame did not comply with the 802.3 specification).
GPI2:	RXSHORT	set (1) if the received frame has a size less than the value of the MIN_SIZE Link control register.
GPI3:	RXLONG	set (1) if the received frame has a size greater than the value of the MAX_SIZE Link control register.
GPI4:	RX_FIFO_OVRUN	set (1) if during reception of the present frame either of the two FIFO's that are part of the Link receive functional block experienced an over-run condition. This will cause some of the data in the frame to be lost or corrupted. The RX_CRC_ERR status line will also be set if this error condition occurs.
GPI5:	RX_CTRL_FRAME	set (1) if the received frame had an Ethernet Length/Type field (the two bytes, in an 802.3 compliant frame, after the source address) equal in value to the ETYPE Link control register.
Request Pending Test:		
GPI6:	REQ_PRESENT	This GPI is essentially the same as the request interrupt. It tests positive if any of the request counters is non-zero.
Hash Lookup Tests:		**Used inside the receive frame and the transmit interrupts to determine the state of the HASH functional block.**
GPI7:	DMADSTFAIL	set (1) if the Hash search on the Link receive frame destination address failed
GPI8:	DMASRCFAIL	set (1) if the Hash search on the Link receive frame source address failed
GPI9:	DPISRCFAIL	set (1) if the Hash search on the source address of the frame read across PCI expansion bus (by the DPI functional block) failed
GPI10:	DMADSTBSY	Hash busy: hash search on Link receive destination address has not completed
GPI11:	DMASRCBSY	Hash busy: hash search on Link receive source address has not completed
GPI12:	DPISRCBSY	Hash busy: hash search on source address of frame read across the PCI expansion bus
Alarm Tests:		**Used in alarm polling to determine if an alarm triggered.**
GPI13:	—	Alarm 1 triggered. Side effect: when tested will de-assert the ALARM1 interrupt if presently set.
GPI14:	—	Alarm 2 triggered

	Side effect: when tested will de-assert the ALARM2 interrupt if presently set.

GPO (Switch Processor General Purpose Outputs)

The Switch Processor has thirty-two GPO lines that can be toggled by the use of the **FLIP** instruction. These outputs allow firmware to control various functional blocks on the PM3351. The hardware implementation of each GPO line can be either level-sensitive or pulsed for a single cycle. In either case the same Switch Processor **FLIP** instruction is used and whether the signal is used in the internal functional block as level-sensitive or edge-sensitive is a hardware implementation detail that does not concern the programmer.

As previously noted, the present status of any of the GPO lines is visible to the Switch Processor by reading the OUTCBL and OUTCBH special purpose registers (which map, respectively, to the GPO[15:0] and GPO[31:16] lines).

GPO Table:	HW Name	Level/ Pulse	Description
FreeLIST LOCK Controls:			Used to allow the Switch Processor and DMA hardware to have shared access to the DMFLIST and DMNFREE control registers. The Switch Processor is only allowed to modify the contents of the DMFLIST and DMNFREE registers if the internal **flist_lock** signal is set (1).
GPO0	FLIST_UNLOCK	Pulse	Free list lock off. A flip on this GPO line will unconditionally cause the internal **flist_lock** signal to become **clear (0).**
GPO1	FLIST_SET_LOCK	Pulse	Free list lock on. A flip on this GPO line will unconditionally cause the internal **flist_lock** signal to become **set (1).**
Hash :			**Control line for testing of HASH functional block by firmware.**
GPO2	RISC_HASH	level	When set (1), allows the Switch Processor to initiate a HASH search using writes to the HDPISRC, HDMADST, or HDMASRC control registers. Set only for testing purposes.
Request and ACKNOWLEDGE COUNTERS:			
GPO3	RISC_REQ_LOCK	level	Used to qualify the REQ_INT interrupt. When set (1), the REQ_INT interrupt will not be asserted. As a side note, the interrupt mask registers (EMASK, ESCH) can be used to perform the same function.

GPO4	REQ_DEC	Pulse	a flip causes one of the eight REQUEST counters to decrement. Decrements the selected request counter (see REQID_NUM field of the REQID control register). The counters get selected and prioritized by the counter functional block.
GPO5	ACK_DEC	Pulse	a flip causes one of the eight ACKNOWLEDGE counters to decrement. Decrements the selected acknowledge counter (see ACKID_NUM field of CTRSEL register). The counters get selected and prioritized by the counter functional block.
GPO6	TASK_INC	Pulse	a flip causes the TASKCTR control register to increment.
GPO7	TASK_DEC	pulse	a flip causes the TASKCTR control register to decrement.
GPO8	TEST_CNTR_INCR	pulse	used for factory testing only.
GPO9	HOLDOFF_REQ_INT	pulse	a flip causes the REQ_INT interrupt to be unconditionally de-asserted for the next 512 SYSCLK cycles.
GPO10	FDPLX_TXPAUSE	level	when set (1), inhibits reading of data frames from the TX_FIFO data ram. This GPO line is used by the Switch Processor, in conjunction with a software timer, to implement the PAUSE operation as per 802.3x, clause 31.
GPO11	-	level	Unused
GPO12	-	level	Unused
GPO13	-	level	unused
Thaw flips:			**this group of GPO lines is used to implement flow control between the various functional blocks and the firmware.**
GPO14	THAW_FLIST_INT	pulse	this flip will clear (0) the FLIST_ERR interrupt line. However, if the condition which caused the FLIST_ERR interrupt in the first place is still true, then the interrupt will remain asserted.
GPO15	THAW_RXLNK	pulse	this flip will clear (0) the receive frame interrupt condition (either RX_FRAME or RX_ERR_FRAME) and permits the DMA functional block to start reading the next frame from the Link receive functional block. As previously noted, the logical implementation of the DMA and link receivechannel is such that the DMA module can hold the status of the previously received frame while the current frame is being received. Consider a receive frame sequence of: frame #N, #N+1, #N+2 … A link receive FIFO over-run condition **will not** occur unless the servicing of the DMA interrupt for frame #N has not been cleared via a flip on THAW_RXLNK before the first 64 data bytes of frame #N+1 have been impressed on the MII receive interface pins.

GPO16	THAW_COLLISION	pulse	this flip will clear (0) the COLLISION interrupt line and: (1) the TXLATECOL transmit status bit is cleared (2) it allows the TX_FIFO output state machine to leave a collision holding state, thereby allowing either another attempt at transmitting the frame or for the frame to be dropped (see DSHEAD control register description).
GPO17	THAW_DPI_DONE	pulse	this flip will clear the DPI_DONE interrupt line, thereby allowing the DPI state machine to transition to IDLE. This allows the next frame to be fetched by reading across the PCI expansion bus.
GPO18	THAW_TXDONE	pulse	this flip will clear the TX_DONE interrupt line and allows the next transmit frame status to be loaded into the TXF_DSIZE control register. As previously noted, the logical implementation of the TX_FIFO and link transmit channel is such that the TX_FIFO module can hold the status of the previously transmitted frame while transmitting the current frame. Consider a transmit frame sequence of: frame #N, #N+1, #N+2 … Assume that there are no collisions on the link. The TX_FIFO **will not** enter a flow control state having to do with the servicing of the TX_DONE interrupt for frame #N+1 unless the TX_DONE interrupt corresponding to frame #N has not been cleared via a flip on THAW_TXDONE by the time that the last data byte of frame #N+1 has been transmit on the MII transmit interface pins.
GPO19	THAW_DPI_HD	pulse	this flip will clear the DPI_HD_DONE interrupt line. The DPI state machine will not return to IDLE after completing the read of the presently fetched frame across the PCI expansion bus unless the DPI_HD_DONE interrupt is clear.
GPO20	CLR_RPCIM_DONE	pulse	this flip will clear the RPCIM_DONE interrupt line. This allows the RPCIM state machine to return to IDLE and start the next PCI transaction.
GPO21	GPO_TXFI_RWFRAME	level	asserted while the Switch Processor is writing a frame into either the TX_FIFO data ram or MAC Ctrl ram. The first flip (leading edge assertion of GPO_TXFI_RWFRAME) is interpreted as the BeginningOfFrame. The second flip (de-assertion of GPO_TXFI_RWFRAME) is interpreted as the EndOfFrame and results in the completion of the frame write into the target FIFO. The Switch Processor writes the frame data as previously discussed in the section dealing with the TX_FIFO control registers.
Enable flips:			**These flips are used to trigger hardware procedures.**
GPO22	GO_REQ	pulse	this flip will initiate the transfer handshake channel to do a **request counter** increment

GPO23	GO_ACK	pulse	this flip will initiate the transfer handshake channel to do an **acknowledge counter** increment
GPO24	GO_DPI	pulse	this flip sets a valid status bit in the DPI functional block (DMA transfer channel) that indicates that the data descriptor pointer in the RemDD is to be read across the PCI expansion bus.
GPO25	GO_TXMCFRAME	pulse	this flip sets a valid status bit in the TX_FIFO module that indicates that a MAC Control frame should be transmit (the implicit assumption is that the Switch Processor will have written the correct frame contents to the Mac Control frame ram prior to doing the GO_TXMCFRAME flip).
MDIO outputs:			**these flips are used to control the MII management interface pins: MDC and MDIO.**
GPO26	O_MDC	level	the level of this GPO line corresponds to the TTL logic level on the MDC pin.
GPO27	O_MDIO	level	the level of this GPO line corresponds to the TTL logic level driven by the PM3351 on the MDIO pin if the output enable, OE_MDIO, is set (1).
GPO28	OE_MDIO	level	MDIO output enable. If set (1) then the MDIO tristate output buffer is enabled and the MDIO pin is driven to the value of the O_MDIO GPO line.
Switch Processor testing flips:			**these flips are used for test purposes.**
GPO29	RINT	level	RINT is a mask-able interrupt source for the INT_ pin on the PM3351. If GPO29 is set (1) and the RINTMSK bit is set in HCTRL, then the INT_ pin will be asserted (a TTL logic low). See description of RINT in Host Interface Status (HSTAT) and Control (HCTRL) registers for additional details.
GPO30	ASSERT_ALARM1	level	if set (1), the ALARM1 interrupt line will be asserted.
GPO31	ASSERT_ALARM2	level	if set (1), the ALARM2 interrupt line will be asserted.

Coprocessor Test Conditions

The Switch Processor has 16 coprocessor conditions. From a programmer's standpoint they are not much different than the GPI inputs except that they are tested by using different instructions (BCOP and BNCOP).

Coprocessor #	HW Name	Description
Cop0:	LINK_IDLE	set (1) if both the Link transmit and receive channels are idle. This test condition is not required for switching applications and is mapped to a cop_condition bit for observability.
Cop1:	ANY_RXFRAME_INT	set (1) if either link receive interrupt line is set.

Cop2:	RX_FRAME	this coprocessor test condition line is mapped to the RX_FRAME interrupt line. The combination of Cop1 and Cop2 allows an efficient means (branch-and-test) for the Switch Processor to test for an outstanding receive frame interrupt while it is servicing a lower priority, but possible lengthy, interrupt service routine.
Cop3:	DPO_IDLE	Transfer handshake channel (i.e. DPO functional block) is in IDLE state. The transfer handshake channel is in the IDLE state if there are no outstanding increment request counter or increment acknowledge counter operations. The Switch Processor should test for this bit being set (1) before writing to the template DD registers in the transfer handshake channel.
Cop4:	FDPLX	this line is logically the same as the LNCTRL register FDPLX bit (set to 1 if the Link is configured for full duplex).
Cop5:	RPCIM_IDLE	set (1) if the RPCIM module is in an IDLE state.
Cop6:	I_MDIO_R	this coprocessor condition bit is mapped to the MDIO input buffer. It is tested during MII management reads to determine if the MDIO pin being driving by the external PHY device is a 0 or a 1.
Cop7:	ANY_DPO_AF	set (1) if any of the eight Transfer rings are full or almost-full (that is, within 1 element of being full).
Cop8:	FLIST_NULL	set (1) if the DMFLIST control register pair is equal to 0.
Cop9:	TXF_AF	set (1) if the TX_FIFO data ram is almost full. The almost full condition is set when the data ram has less free space than that corresponding to the PBSIZE control register. TXF_AF is tested by the Switch Processor during the write of frame data into the TX_FIFO data ram.
Cop10:	DPI_TXFI_IDLE	entire PCI frame read expansion port is in IDLE state up to the logical tail of the TX_FIFO. This coprocessor condition is expected to be used for testing before firmware starts to write a Transmit management frame (that is a frame created by the Switch Processor) to the TX_FIFO data ram.
Cop11:	TXFO_IDLE	TX_FIFO OSTATE state machine is idle
Cop12:	TXFOMC_IDLE	TX_FIFO OMCSTATE state machine is idle
Cop13:	TXMCFRAME_EMPTY	TX_FIFO MAC Control frame data ram is empty
Cop14:	Cop_regs[0]	this is mapped to the cop_regs[0] bit in the DMCTRL register. Implemented to allow software set-able branch-test condition.
Cop15:	Cop_regs[1]	this is mapped to the cop_regs[1] bit in the DMCTRL register. Implemented to allow software set-able branch-test condition.

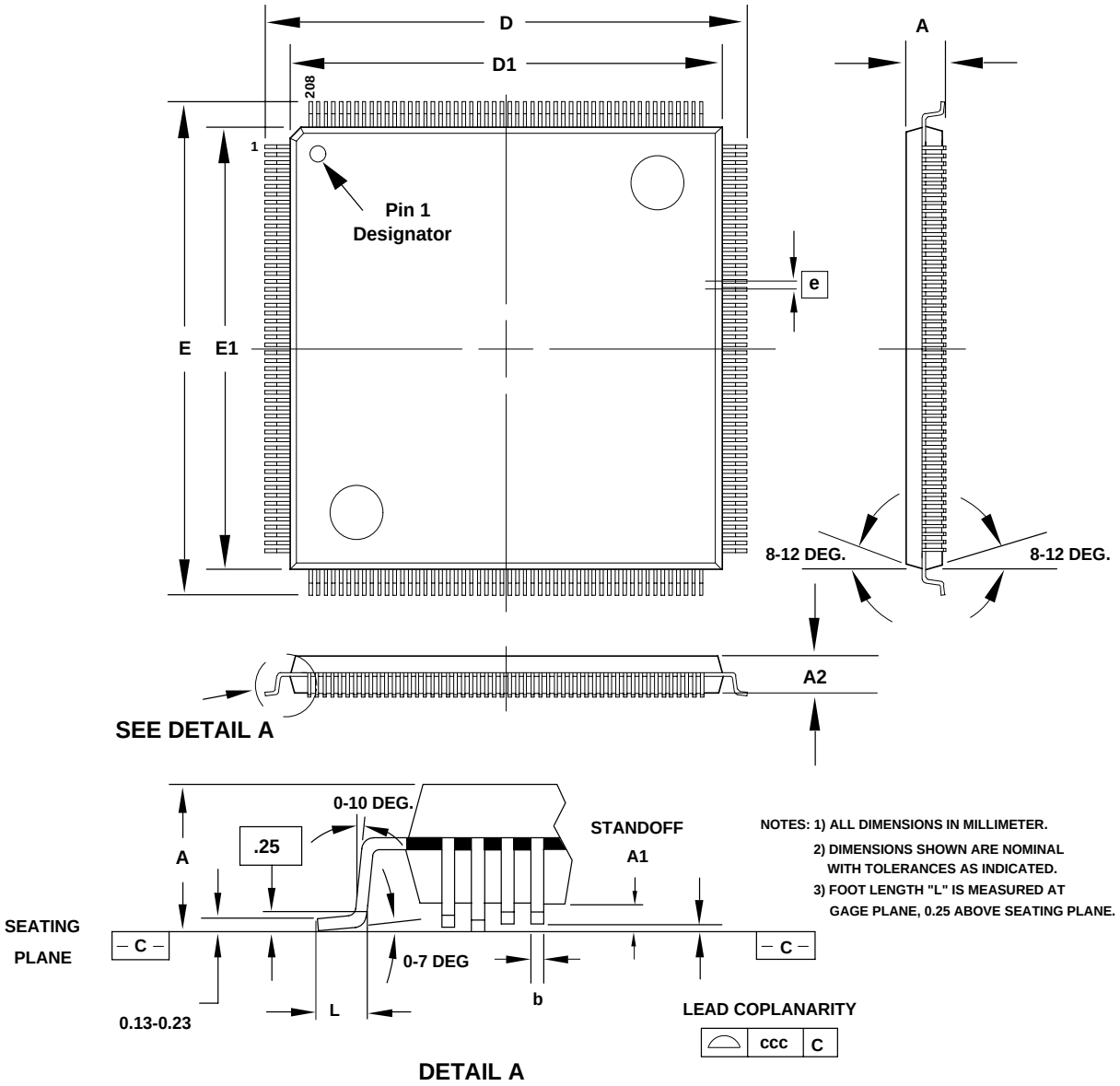
OPERATION

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. VBIAS power must be supplied either before VDDI and VDDO or simultaneously with VDDI and VDDO to prevent current flow through the ESD protection devices which exist between VBIAS and VDDI/VDDO power supplies.
2. VDDI power must be supplied either before VDDO or simultaneously with VDDO to prevent current flow through the ESD protection devices which exist between VDDI and VDDO power supplies. Connection to a common VDD power plane is the recommended standard practice for customer applications.
3. 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 VDDI 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 (20 mA).
4. Power down the device in the reverse sequence.

MECHANICAL INFORMATION



DETAIL A

PACKAGE TYPE: 208 PIN METRIC PLASTIC QUAD FLATPACK-MQFP											
BODY SIZE: 28 x 28 x 3.49 MM											
Dim.	A	A1	A2	D	D1	E	E1	L	e	b	ccc
Min.	3.64	0.25	3.39	30.40	27.90	30.40	27.90	0.50		0.17	
Nom.			3.49	30.60	28.00	30.60	28.00	0.60	0.50	0.22	
Max.	4.07	0.48	3.59	30.80	28.10	30.80	28.10	0.75		0.27	0.10

ORDERING AND THERMAL INFORMATION

PART NO.	DESCRIPTION
PM3351-RC	208 Plastic Quad Flat Pack (PQFP)
PM3351-SW	Switching Code Firmware

PART NO.	CASE TEMPERATURE	Theta Ja
PM3351-RC	-0°C to 70°C	28 °C/W

NOTES

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

PM-970113 (R3) ref PMC-961052 (R3) Issue date: February 1998