

***TMS320C645x DSP  
Ethernet Media Access Controller (EMAC)/  
Management Data Input/Output (MDIO)***

***User's Guide***

Literature Number: SPRU975B  
August 2006





<b>Preface</b> .....	<b>10</b>
<b>1 Introduction</b> .....	<b>11</b>
1.1 Purpose of the Peripheral .....	11
1.2 Features .....	11
1.3 Functional Block Diagram .....	12
1.4 Industry Standard(s) Compliance Statement .....	13
<b>2 EMAC Functional Architecture</b> .....	<b>14</b>
2.1 Clock Control.....	14
2.2 Memory Map .....	15
2.3 System Level Connections .....	16
2.4 Ethernet Protocol Overview .....	24
2.5 Programming Interface.....	26
2.6 EMAC Control Module .....	37
2.7 Management Data Input/Output (MDIO) Module.....	38
2.8 EMAC Module.....	43
2.9 Media Independent Interfaces .....	46
2.10 Packet Receive Operation.....	50
2.11 Packet Transmit Operation .....	55
2.12 Receive and Transmit Latency .....	55
2.13 Transfer Node Priority.....	56
2.14 Reset Considerations .....	56
2.15 Initialization .....	57
2.16 Interrupt Support.....	60
2.17 Power Management.....	63
2.18 Emulation Considerations .....	63
<b>3 EMAC Control Module Registers</b> .....	<b>64</b>
3.1 Introduction .....	64
3.2 EMAC Control Module Interrupt Control Register (EWCTL).....	64
3.3 EMAC Control Module Interrupt Timer Count Register (EWINTTCNT) .....	65
<b>4 MDIO Registers</b> .....	<b>66</b>
4.1 Introduction .....	66
4.2 MDIO Version Register (VERSION) .....	67
4.3 MDIO Control Register (CONTROL).....	68
4.4 PHY Acknowledge Status Register (ALIVE) .....	69
4.5 PHY Link Status Register (LINK).....	70
4.6 MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRAW) .....	71
4.7 MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED) .....	72
4.8 MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW) .....	73
4.9 MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED) .....	74
4.10 MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET).....	75
4.11 MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR).....	76
4.12 MDIO User Access Register 0 (USERACCESS0) .....	77
4.13 MDIO User PHY Select Register 0 (USERPHYSEL0) .....	78
4.14 MDIO User Access Register 1 (USERACCESS1) .....	79

4.15	MDIO User PHY Select Register 1 (USERPHYSEL1) .....	80
<b>5</b>	<b>EMAC Port Registers .....</b>	<b>81</b>
5.1	Introduction .....	81
5.2	Transmit Identification and Version Register (TXIDVER) .....	85
5.3	Transmit Control Register (TXCONTROL) .....	86
5.4	Transmit Teardown Register (TXTEARDOWN) .....	87
5.5	Receive Identification and Version Register (RXIDVER).....	88
5.6	Receive Control Register (RXCONTROL) .....	89
5.7	Receive Teardown Register (RXTEARDOWN).....	90
5.8	Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW) .....	91
5.9	Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED).....	92
5.10	Transmit Interrupt Mask Set Register (TXINTMASKSET) .....	93
5.11	Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR) .....	94
5.12	MAC Input Vector Register (MACINVECTOR) .....	95
5.13	Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW).....	96
5.14	Receive Interrupt Status (Masked) Register (RXINTSTATMASKED) .....	97
5.15	Receive Interrupt Mask Set Register (RXINTMASKSET).....	98
5.16	Receive Interrupt Mask Clear Register (RXINTMASKCLEAR).....	99
5.17	MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW).....	100
5.18	MAC Interrupt Status (Masked) Register (MACINTSTATMASKED) .....	101
5.19	MAC Interrupt Mask Set Register (MACINTMASKSET) .....	102
5.20	MAC Interrupt Mask Clear Register (MACINTMASKCLEAR).....	103
5.21	Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) .....	104
5.22	Receive Unicast Enable Set Register (RXUNICASTSET).....	106
5.23	Receive Unicast Clear Register (RXUNICASTCLEAR).....	107
5.24	Receive Maximum Length Register (RXMAXLEN) .....	108
5.25	Receive Buffer Offset Register (RXBUFFEROFFSET) .....	109
5.26	Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH) .....	110
5.27	Receive Channel 0-7 Flow Control Threshold Register (RXnFLOWTHRESH).....	111
5.28	Receive Channel 0-7 Free Buffer Count Register (RXnFREEBUFFER) .....	112
5.29	MAC Control Register (MACCONTROL) .....	113
5.30	MAC Status Register (MACSTATUS) .....	115
5.31	Emulation Control Register (EMCONTROL) .....	117
5.32	FIFO Control Register (FIFOCONTROL) .....	118
5.33	MAC Configuration Register (MACCONFIG).....	119
5.34	Soft Reset Register (SOFTRESET) .....	120
5.35	MAC Source Address Low Bytes Register (MACSRCADDRLO).....	121
5.36	MAC Source Address High Bytes Register (MACSRCADDRHI) .....	122
5.37	MAC Hash Address Register 1 (MACHASH1) .....	123
5.38	MAC Hash Address Register 2 (MACHASH2) .....	124
5.39	Back Off Test Register (BOFFTEST).....	125
5.40	Transmit Pacing Algorithm Test Register (TPACETEST) .....	126
5.41	Receive Pause Timer Register (RXPAUSE) .....	127
5.42	Transmit Pause Timer Register (TXPAUSE).....	128
5.43	MAC Address Low Bytes Register (MACADDRLO).....	129
5.44	MAC Address High Bytes Register (MACADDRHI) .....	130
5.45	MAC Index Register (MACINDEX) .....	131
5.46	Transmit Channel 0-7 DMA Head Descriptor Pointer Register (TXnHDP).....	132
5.47	Receive Channel 0-7 DMA Head Descriptor Pointer Register (RXnHDP) .....	133

---

5.48	Transmit Channel 0-7 Completion Pointer Register (TXnCP).....	134
5.49	Receive Channel 0-7 Completion Pointer Register (RXnCP) .....	135
5.50	Network Statistics Registers .....	136
<b>Appendix A Glossary .....</b>		<b>145</b>
<b>Appendix B Revision History .....</b>		<b>147</b>

## List of Figures

1	EMAC and MDIO Block Diagram .....	12
2	Ethernet Configuration with MII Interface .....	16
3	Ethernet Configuration with RMI Interface .....	18
4	Ethernet Configuration with GMII Interface .....	20
5	Ethernet Configuration with RGMII Interface .....	22
6	Ethernet Frame .....	24
7	Basic Descriptor Format .....	26
8	Typical Descriptor Linked List .....	27
9	Transmit Descriptor Format .....	30
10	Receive Descriptor Format .....	33
11	EMAC Control Module Block Diagram .....	37
12	MDIO Module Block Diagram .....	39
13	EMAC Module Block Diagram .....	43
14	EMAC Control Module Interrupt Control Register (EWCTL) .....	64
15	EMAC Control Module Interrupt Timer Count Register (EWINTCNT) .....	65
16	MDIO Version Register (VERSION) .....	67
17	MDIO Control Register (CONTROL) .....	68
18	PHY Acknowledge Status Register (ALIVE) .....	69
19	PHY Link Status Register (LINK) .....	70
20	MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRAW) .....	71
21	MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED) .....	72
22	MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW) .....	73
23	MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED) .....	74
24	MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET) .....	75
25	MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR) .....	76
26	MDIO User Access Register 0 (USERACCESS0) .....	77
27	MDIO User PHY Select Register 0 (USERPHYSEL0) .....	78
28	MDIO User Access Register 1 (USERACCESS1) .....	79
29	MDIO User PHY Select Register 1 (USERPHYSEL1) .....	80
30	Transmit Identification and Version Register (TXIDVER) .....	85
31	Transmit Control Register (TXCONTROL) .....	86
32	Transmit Teardown Register (TXTEARDOWN) .....	87
33	Receive Identification and Version Register (RXIDVER) .....	88
34	Receive Control Register (RXCONTROL) .....	89
35	Receive Teardown Register (RXTEARDOWN) .....	90
36	Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW) .....	91
37	Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED) .....	92
38	Transmit Interrupt Mask Set Register (TXINTMASKSET) .....	93
39	Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR) .....	94
40	MAC Input Vector Register (MACINVECTOR) .....	95
41	Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW) .....	96
42	Receive Interrupt Status (Masked) Register (RXINTSTATMASKED) .....	97
43	Receive Interrupt Mask Set Register (RXINTMASKSET) .....	98
44	Receive Interrupt Mask Clear Register (RXINTMASKCLEAR) .....	99
45	MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW) .....	100
46	MAC Interrupt Status (Masked) Register (MACINTSTATMASKED) .....	101
47	MAC Interrupt Mask Set Register (MACINTMASKSET) .....	102
48	MAC Interrupt Mask Clear Register (MACINTMASKCLEAR) .....	103
49	Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) .....	104
50	Receive Unicast Enable Set Register (RXUNICASTSET) .....	106
51	Receive Unicast Clear Register (RXUNICASTCLEAR) .....	107
52	Receive Maximum Length Register (RXMAXLEN) .....	108

---

53	Receive Buffer Offset Register (RXBUFFEROFFSET) .....	109
54	Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH).....	110
55	Receive Channel <i>n</i> Flow Control Threshold Register (RX <i>n</i> FLOWTHRESH) .....	111
56	Receive Channel <i>n</i> Free Buffer Count Register (RX <i>n</i> FREEBUFFER) .....	112
57	MAC Control Register (MACCONTROL) .....	113
58	MAC Status Register (MACSTATUS).....	115
59	Emulation Control Register (EMCONTROL) .....	117
60	FIFO Control Register (FIFOCONTROL).....	118
61	MAC Configuration Register (MACCONFIG) .....	119
62	Soft Reset Register (SOFTRESET) .....	120
63	MAC Source Address Low Bytes Register (MACSRCADDRLO).....	121
64	MAC Source Address High Bytes Register (MACSRCADDRHI).....	122
65	MAC Hash Address Register 1 (MACHASH1).....	123
66	MAC Hash Address Register 2 (MACHASH2).....	124
67	Back Off Random Number Generator Test Register (BOFFTEST) .....	125
68	Transmit Pacing Algorithm Test Register (TPACETEST) .....	126
69	Receive Pause Timer Register (RXPAUSE) .....	127
70	Transmit Pause Timer Register (TXPAUSE).....	128
71	MAC Address Low Bytes Register (MACADDRLO).....	129
72	MAC Address High Bytes Register (MACADDRHI) .....	130
73	MAC Index Register (MACINDEX) .....	131
74	Transmit Channel <i>n</i> DMA Head Descriptor Pointer Register (TX <i>n</i> HDP) .....	132
75	Receive Channel <i>n</i> DMA Head Descriptor Pointer Register (RX <i>n</i> HDP) .....	133
76	Transmit Channel <i>n</i> Completion Pointer Register (TX <i>n</i> CP).....	134
77	Receive Channel <i>n</i> Completion Pointer Register (RX <i>n</i> CP) .....	135
78	Statistics Register.....	136

## List of Tables

1	Interface Selection Pins .....	16
2	EMAC and MDIO Signals for MII Interface .....	17
3	EMAC and MDIO Signals for RMII Interface .....	19
4	EMAC and MDIO Signals for GMII Interface .....	21
5	EMAC and MDIO Signals for RGMII Interface .....	23
6	Ethernet Frame Description.....	24
7	Basic Descriptors.....	26
8	Receive Frame Treatment Summary .....	53
9	Middle of Frame Overrun Treatment .....	54
10	Emulation Control .....	63
11	EMAC Control Module Registers.....	64
12	EMAC Control Module Interrupt Control Register (EWCTL) Field Descriptions .....	64
13	EMAC Control Module Interrupt Timer Count Register (EWINTCNT) Field Descriptions.....	65
14	Management Data Input/Output (MDIO) Registers .....	66
15	MDIO Version Register (VERSION) Field Descriptions .....	67
16	MDIO Control Register (CONTROL) Field Descriptions .....	68
17	PHY Acknowledge Status Register (ALIVE) Field Descriptions .....	69
18	PHY Link Status Register (LINK) Field Descriptions .....	70
19	MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRAW) Field Descriptions .....	71
20	MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED) Field Descriptions.....	72
21	MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW) Field Descriptions .....	73
22	MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED) Field Descriptions.....	74
23	MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET) Field Descriptions .....	75
24	MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR) Field Descriptions .....	76
25	MDIO User Access Register 0 (USERACCESS0) Field Descriptions.....	77
26	MDIO User PHY Select Register 0 (USERPHYSEL0) Field Descriptions .....	78
27	MDIO User Access Register 1 (USERACCESS1) Field Descriptions.....	79
28	MDIO User PHY Select Register 1 (USERPHYSEL1) Field Descriptions .....	80
29	Ethernet Media Access Controller (EMAC) Registers.....	81
30	Transmit Identification and Version Register (TXIDVER) Field Descriptions.....	85
31	Transmit Control Register (TXCONTROL) Field Descriptions .....	86
32	Transmit Teardown Register (TXTEARDOWN) Field Descriptions.....	87
33	Receive Identification and Version Register (RXIDVER) Field Descriptions .....	88
34	Receive Control Register (RXCONTROL) Field Descriptions .....	89
35	Receive Teardown Register (RXTEARDOWN) Field Descriptions .....	90
36	Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW) Field Descriptions.....	91
37	Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED) Field Descriptions .....	92
38	Transmit Interrupt Mask Set Register (TXINTMASKSET) Field Descriptions .....	93
39	Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR) Field Descriptions.....	94
40	MAC Input Vector Register (MACINVECTOR) Field Descriptions.....	95
41	Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW) Field Descriptions .....	96
42	Receive Interrupt Status (Masked) Register (RXINTSTATMASKED) Field Descriptions.....	97
43	Receive Interrupt Mask Set Register (RXINTMASKSET) Field Descriptions .....	98
44	Receive Interrupt Mask Clear Register (RXINTMASKCLEAR) Field Descriptions .....	99
45	MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW) Field Descriptions .....	100
46	MAC Interrupt Status (Masked) Register (MACINTSTATMASKED) Field Descriptions.....	101
47	MAC Interrupt Mask Set Register (MACINTMASKSET) Field Descriptions .....	102
48	MAC Interrupt Mask Clear Register (MACINTMASKCLEAR) Field Descriptions .....	103
49	Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions .....	104



---

50	Receive Unicast Enable Set Register (RXUNICASTSET) Field Descriptions .....	106
51	Receive Unicast Clear Register (RXUNICASTCLEAR) Field Descriptions .....	107
52	Receive Maximum Length Register (RXMAXLEN) Field Descriptions .....	108
53	Receive Buffer Offset Register (RXBUFFEROFFSET) Field Descriptions.....	109
54	Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH) Field Descriptions .....	110
55	Receive Channel <i>n</i> Flow Control Threshold Register (RX $n$ FLOWTHRESH) Field Descriptions.....	111
56	Receive Channel <i>n</i> Free Buffer Count Register (RX $n$ FREEBUFFER) Field Descriptions .....	112
57	MAC Control Register (MACCONTROL) Field Descriptions .....	113
58	MAC Status Register (MACSTATUS) Field Descriptions .....	115
59	Emulation Control Register (EMCONTROL) Field Descriptions.....	117
60	FIFO Control Register (FIFOCONTROL) Field Descriptions.....	118
61	MAC Configuration Register (MACCONFIG) Field Descriptions .....	119
62	Soft Reset Register (SOFTRESET) Field Descriptions.....	120
63	MAC Source Address Low Bytes Register (MACSRCADDRLO) Field Descriptions .....	121
64	MAC Source Address High Bytes Register (MACSRCADDRHI) Field Descriptions.....	122
65	MAC Hash Address Register 1 (MACHASH1) Field Descriptions .....	123
66	MAC Hash Address Register 2 (MACHASH2) Field Descriptions .....	124
67	Back Off Test Register (BOFFTEST) Field Descriptions .....	125
68	Transmit Pacing Algorithm Test Register (TPACETEST) Field Descriptions.....	126
69	Receive Pause Timer Register (RXPAUSE) Field Descriptions.....	127
70	Transmit Pause Timer Register (TXPAUSE) Field Descriptions .....	128
71	MAC Address Low Bytes Register (MACADDRLO) Field Descriptions .....	129
72	MAC Address High Bytes Register (MACADDRHI) Field Descriptions.....	130
73	MAC Index Register (MACINDEX) Field Descriptions .....	131
74	Transmit Channel <i>n</i> DMA Head Descriptor Pointer Register (TX $n$ HDP) Field Descriptions.....	132
75	Receive Channel <i>n</i> DMA Head Descriptor Pointer Register (RX $n$ HDP) Field Descriptions .....	133
76	Transmit Channel <i>n</i> Completion Pointer Register (TX $n$ CP) Field Descriptions .....	134
77	Receive Channel <i>n</i> Completion Pointer Register (RX $n$ CP) Field Descriptions.....	135
78	Statistics Register Field Descriptions .....	136
A-1	Physical Layer Definitions .....	146
B-1	Document Revision History.....	147

## Read This First

---

---

---

### About This Manual

This document provides a functional description of the Ethernet Media Access Controller (EMAC) and Physical layer (PHY) device Management Data Input/Output (MDIO) module integrated with TMS320C645x devices.

### Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

### Related Documentation From Texas Instruments

The following documents describe the C6000™ devices and related support tools. Copies of these documents are available on the Internet at [www.ti.com](http://www.ti.com). *Tip:* Enter the literature number in the search box provided at [www.ti.com](http://www.ti.com).

**TMS320C6000 CPU and Instruction Set Reference Guide** (literature number [SPRU732B](#)) describes the CPU architecture, pipeline, instruction set, and interrupts of the C64x and C64x+ DSPs.

**TMS320C6455 Technical Reference** (literature number [SPRU965](#)) gives an introduction to the TMS320C6455™ DSP and discusses the application areas that are enhanced.

**TMS320C6000 Programmer's Guide** (literature number [SPRU198](#)) describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

**TMS320C6000 Code Composer Studio Tutorial** (literature number [SPRU301](#)) introduces the Code Composer Studio™ integrated development environment and software tools.

**Code Composer Studio Application Programming Interface Reference Guide** (literature number [SPRU321](#)) describes the Code Composer Studio™ application programming interface (API), which allows you to program custom plug-ins for Code Composer.

**TMS320C64x+ Megamodule Reference Guide** (literature number [SPRU871](#)) describes the TMS320C64x+ digital signal processor (DSP) megamodule. Included is a discussion on the internal direct memory access (IDMA) controller, the interrupt controller, the power-down controller, memory protection, bandwidth management, and the memory and cache.

**TMS320C645x DSP Peripherals Overview Reference Guide** (literature number [SPRUE52](#)) provides a brief description of the peripherals available on the TMS320C645x digital signal processors (DSPs).

**TMS320C6455 Chip Support Libraries (CSL)** (literature number [SPRC234](#)) is a download with the latest chip support libraries.

### Trademarks

C6000, TMS320C6455, TMS320C6000, Code Composer Studio are trademarks of Texas Instruments.

# ***Ethernet Media Access Controller (EMAC)/Management Data Input/Output (MDIO)***

---

---

---

## **1 Introduction**

This document provides a functional description of the Ethernet Media Access Controller (EMAC) and Physical layer (PHY) device Management Data Input/Output (MDIO) module integrated with TMS320C645x (C645x) devices. Included are the features of the EMAC and MDIO modules, a discussion of their architecture and operation, how these modules connect to the outside world, and the registers description for each module.

The EMAC controls the flow of packet data from the processor to the PHY. The MDIO module controls PHY configuration and status monitoring.

Both the EMAC and the MDIO modules interface to the DSP through a custom interface that allows efficient data transmission and reception. This custom interface is referred to as the EMAC control module, and is considered integral to the EMAC/MDIO peripheral.

### **1.1 Purpose of the Peripheral**

The EMAC module is used on TMS320C645x devices to move data between the device and another host connected to the same network, in compliance with the Ethernet protocol.

### **1.2 Features**

The basic feature set of the EMAC module integrated with C645x is:

- Synchronous 10/100/1000 Mbps operation
- Full duplex Gigabit operation (half duplex gigabit is not supported)
- Little endian and big endian support
- Supports four types of interfaces to the physical layer device (PHY): standard media independent interface (MII), reduced pin count media independent interface (RMII), standard gigabit media independent interface (GMII) and reduced pin count gigabit media independent interface (RGMII)
- EMAC acts as DMA master to either internal or external device memory space
- Eight receive channels with VLAN tag discrimination for receive quality of service (QOS) support
- Eight transmit channels with round-robin or fixed priority for transmit quality of service (QOS) support
- Ether-stats and 802.3-stats statistics gathering
- Transmit CRC generation selectable on a per-channel basis
- Broadcast frames selection for reception on a single channel
- Multicast frames selection for reception on a single channel
- Promiscuous receive mode frames selection for reception on a single channel (all frames, all good frames, short frames, error frames)
- Hardware flow control
- 8K byte local EMAC descriptor memory that allows the peripheral to operate on descriptors without affecting the CPU. The descriptor memory holds enough information to transfer up to 512 ethernet packets without CPU intervention.
- Programmable interrupt logic permits the software driver to restrict the generation of back-to-back interrupts, thus allowing more work to be performed in a single call to the interrupt service routine

### 1.3 Functional Block Diagram

Figure 1 shows the three main functional modules of the EMAC/MDIO peripheral:

- EMAC control module
- EMAC module
- MDIO module

The EMAC control module is the main interface between the device core processor and the EMAC module and MDIO module. The EMAC control module contains the necessary components to allow the EMAC to make efficient use of device memory, plus it controls device interrupts. The EMAC control module incorporates 8K byte internal RAM to hold EMAC buffer descriptors.

The management data input / output (MDIO) module implements the 802.3 serial management interface to interrogate and control up to 32 ethernet PHY(s) connected to the device, using a shared two-wire bus. Application software uses the MDIO module to configure the auto-negotiation parameters of each PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC module for correct operation. The module is designed to allow almost transparent operation of the MDIO interface, with very little maintenance from the core processor.

The ethernet media access controller (EMAC) module provides an efficient interface between the C645x core processor and the networked community. The EMAC supports 10Base-T (10 Mbits / sec), and 100BaseTX (100 Mbits / sec), in either half or full duplex mode, and 1000BaseT (1000 Mbits / sec) in full duplex mode, with hardware flow control and quality-of-service (QoS) support.

**Figure 1. EMAC and MDIO Block Diagram**

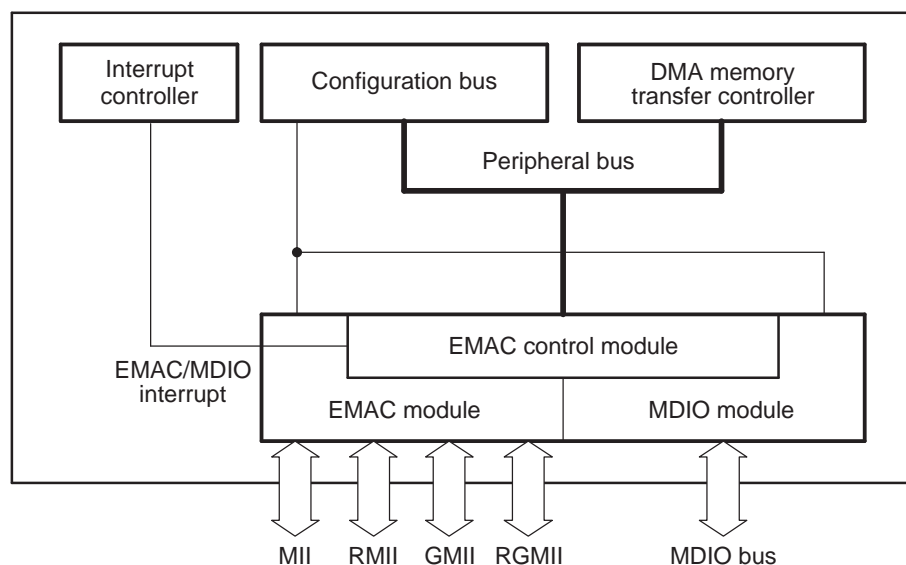


Figure 1 also shows the main interface between the EMAC control module and the CPU.

The following connections are made to the device core:

- The peripheral bus connection from the EMAC control module allows the EMAC module to read and write both internal and external memory through the switch fabric interface.
- The EMAC control, EMAC, and MDIO modules all have control registers. These registers are memory-mapped into device memory space via the device configuration bus. The control module's internal RAM maps to this same range along with these registers.
- The EMAC and MDIO interrupts are combined into a single interrupt within the control module. The interrupt from the control module then goes to the device's interrupt controller.

The EMAC and MDIO interrupts are combined within the control module, so only the control module interrupt needs to be monitored by the application software or device driver. The interrupt is mapped to a specific CPU interrupt via the enhanced interrupt selector within the C64x+ core. The combined EMAC / MDIO interrupt maps to the interrupt controller input as system event 17.

#### **1.4 Industry Standard(s) Compliance Statement**

The EMAC peripheral conforms to the IEEE 802.3 standard, describing the “Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer” specifications. ISO / IEC has also adopted the IEEE 802.3 standard and re-designated it as ISO/IEC 8802-3:2000(E).

In difference from this standard, the EMAC peripheral integrated with the C645x devices does not use the transmit coding error signal MTXER. Instead of driving the error pin when an underflow condition occurs on a transmitted frame, the EMAC intentionally generates an incorrect check sum by inverting the frame CRC so that the network will detect the transmitted frame as an error.

## 2 EMAC Functional Architecture

This chapter discusses the architecture and basic function of the EMAC peripheral.

### 2.1 Clock Control

The frequencies for the transmit and receive clocks are fixed by the IEEE 802.3 specification as shown below:

- 2.5 Mhz at 10 Mbps
- 25 Mhz at 100 Mbps
- 125 MHz at 1000 Mbps

The C645x device uses two PLL controllers to generate all of the clocks that the DSP needs. The primary PLL controller generates a peripheral clock (SYSCLK2) that several peripherals use, including the EMAC. SYSCLK2 runs at a rate equal to 1/6th of the CPU clock frequency (CPUclk/6), and is not programmable.

The secondary PLL controller conveniently generates a reference clock (SYSCLK1). The EMAC uses the SYSCLK1 to generate the transmit and receive clocks for the GMII and RGMII interfaces. The frequency of SYSCLK1 is equal to the secondary PLL controller's input clock (25 MHz) multiplied by 10 and divided by either 5 or 2. You must program SYSCLK1 to the correct frequency based on the interface that you are using (as determined by the configuration pins MACSEL[1:0]). See [Section 2.1.1](#), [Section 2.1.3](#), and [Section 2.1.4](#) for the configuration of SYSCLK1 that is required on each interface.

---

**Note:** The RGMII interface does not require an interface clock. However, a reference clock that is derived from SYSCLK1 is provided for your convenience. The reference clock is not a free running clock. The reference clock stops while the DSP is in reset; therefore, you must be careful if you use this clock elsewhere in the system.

---

The MDIO clock is based on a divide-down of the peripheral clock (SYSCLK2) and is specified to run up to 2.5 MHz, although typical operation is 1.0 MHz. Since the peripheral clock frequency is variable, the application software or driver controls the divide-down amount.

#### 2.1.1 MII Clocking

When you select the MII clocking interface by setting MACSEL to the default value (00b), the transmit and receive clock sources are provided from an external PHY via the MTCLK and MRCLK pins. These clocks are inputs to the EMAC module and operate at 2.5 MHz in 10 Mbps mode, and at 25 MHz in 100 MHz mode. The MII clocking interface is not used in 1000 Mbps mode. For timing purposes, data is transmitted and received with reference to MTCLK and MRCLK, respectively.

#### 2.1.2 RMII Clocking

When you select the RMII interface by setting MACSEL to 01b, you must provide a 50MHz reference clock through the MREFCLK pin. The EMAC clocks the transmit and receive operations from the reference clock. The MTCLK and MRCLK device pins are not used for this interface. The RMII protocol turns one data phase of an MII transfer into two data phases at double the clock frequency. This is the driving factor for the 50 MHz reference clock. Data at the IO pins are running at 5 MHz in 10 Mbps mode, and at 50 MHz in 100 Mbps mode.

#### 2.1.3 GMII Clocking

When you set MACSEL to 02b to select GMII, the transmit and receive clock sources for 10/100 Mbps modes are provided from an external PHY via the MTCLK and MRCLK pins. For 1000 Mbps mode, the receive clock is provided by an external PHY via the MRCLK pin. For transmit in 1000 Mbps mode, the clock is sourced synchronous with the data, and is provided by the EMAC to be output on the GMTCLK pin.

The SYSCLK1 of the secondary PLL controller sources the GMTCLK. The divider generating SYSCLK1 needs to be programmed to /2 (the default value) for this interface to provide a 125 MHz clock to the EMAC.

For timing purposes, data in 10/100 mode is transmitted and received with reference to MTCLK and MRCLK, respectively. For 1000 Mbps mode, receive timing is the same, but transmit is relative to GMTCLK.

#### 2.1.4 RGMII Clocking

When the RGMII interface is selected by setting MACSEL to 11b, you must configure the internal clock (SYSCLK1) to a 125 MHz frequency by setting the divider for the secondary PLL controller to /5. This provides a reference clock that you can use to source the clock on the RGMII PHY as necessary.

---

**Note:** This reference clock is not a free running clock. An external device should only use this clock if it does not expect a valid clock during device reset.

---

The EMAC drives the transmit clock, while an external PHY generates the receive clock. The reference clock drives the device pin that gives the 125 MHz clock to the PHY; this enables the PHY to generate the receive clock that is sent to EMAC.

The RGMII protocol takes a GMII data stream and turns it into an interface with half of the data bus width and sends the same amount of data with a reduced pinout. The RGMII protocol also allows for dynamic switching of the mode between 10/100/1000 Mbps modes. This negotiation data is embedded in the incoming data stream from the external PHY. For timing purposes, data is transmitted and received with respect to MTCLK and MRCLK respectively.

The RGMII interface has separate I/O pins from the other EMAC pins because the interface voltage is different from the other interfaces.

## 2.2 Memory Map

The EMAC includes an internal memory that holds information about the ethernet packets that are received or transmitted. This internal RAM is 2K x 32 bits in size. You can write data to and read data from the EMAC internal memory via either the EMAC or the CPU. It is used to store buffer descriptors that are 4 words (16 bytes) deep. This 8K local memory can hold enough information to transfer up to 512 ethernet packets without CPU intervention.

You can put the packet buffer descriptors in internal processor memory (L2) on the C645x devices. There are some trade-offs in terms of cache performance and throughput when you put descriptors in L2 versus when you put them in EMAC's internal memory. Cache performance improves when you put the buffer descriptors in internal memory. However, the EMAC throughput is better when you put the descriptors in the local EMAC RAM.

## 2.3 System Level Connections

The C645x device supports four different interfaces to a physical layer device. You can only transfer data on one interface at a given time. Each of these interfaces is selected in hardware via the configuration pins (MACSEL[1:0]).

Table 1 shows the possible settings for these configuration pins.

**Table 1. Interface Selection Pins**

MACSEL [1:0]	Interface
00	MII
01	RMII
10	GMII
11	RGMII

### 2.3.1 Media Independent Interface (MII) Connections

Figure 2 shows a device with integrated EMAC and MDIO interfaced via a MII connection. This interface is only available in 10 Mbps and 100 Mbps modes.

**Figure 2. Ethernet Configuration with MII Interface**

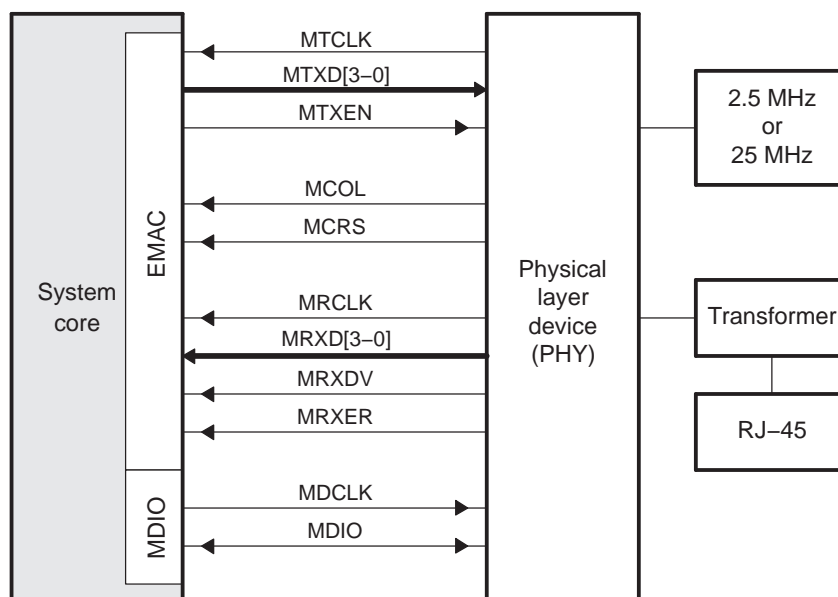




Table 2 summarizes the individual EMAC and MDIO signals for the MII interface. For more information, refer to either the IEEE 802.3 standard or ISO/IEC 8802-3:2000(E).

The EMAC module does not include a transmit error (MTXER) pin. If a transmit error occurs, CRC inversion is used to negate the validity of the transmitted frame.

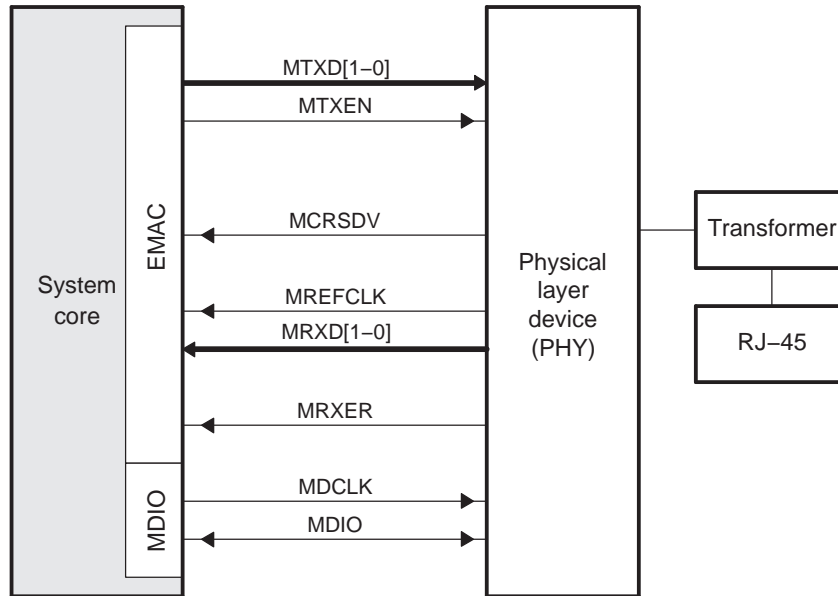
**Table 2. EMAC and MDIO Signals for MII Interface**

Signal Name	I/O	Description
MTCLK	I	Transmit clock (MTCLK). The transmit clock is a continuous clock that provides the timing reference for transmit operations. The MTXD and MTXEN signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation and 25 MHz at 100 Mbps operation.
MTXD[3:0]	O	Transmit data (MTXD). The transmit data pins are a collection of 4 data signals comprising 4 bits of data. MTDX0 is the least-significant bit (LSB). The signals are synchronized by MTCLK and valid only when MTXEN is asserted.
MTXEN	O	Transmit enable (MTXEN). The transmit enable signal indicates that the MTXD pins are generating nibble data for use by the PHY. It is driven synchronously to MTCLK.
MCOL	I	Collision detected (MCOL). The MCOL pin is asserted by the PHY when it detects a collision on the network. It remains asserted while the collision condition persists. This signal is not necessarily synchronous to MTCLK nor MRCLK. This pin is used in half-duplex operation only.
MCRS	I	Carrier sense (MCRS). The MCRS pin is asserted by the PHY when the network is not idle in either transmit or receive. The pin is de-asserted when both transmit and receive are idle. This signal is not necessarily synchronous to MTCLK or MRCLK. This pin is used in half-duplex operation only.
MRCLK	I	Receive clock (MRCLK). The receive clock is a continuous clock that provides the timing reference for receive operations. The MRXD, MRXDV, and MRXER signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation and 25 MHz at 100 Mbps operation.
MRXD[3:0]	I	Receive data (MRXD). The receive data pins are a collection of 4 data signals comprising 4 bits of data. MRDX0 is the least-significant bit (LSB). The signals are synchronized by MRCLK and valid only when MRXDV is asserted.
MRXDV	I	Receive data valid (MRXDV). The receive data valid signal indicates that the MRXD pins are generating nibble data for use by the EMAC. It is driven synchronously to MRCLK.
MRXER	I	Receive error (MRXER). The receive error signal is asserted for one or more MRCLK periods to indicate that an error was detected in the received frame. This is meaningful only during data reception when MRXDV is active.
MDCLK	O	Management data clock (MDCLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO pin. The frequency of this clock is controlled by the CLKDIV bits in the MDIO control register (CONTROL).
MDIO	I/O	Management data input output (MDIO). The MDIO pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO pin acts as an output for everything except the data bit cycles, when the pin acts as an input for read operations.

### 2.3.2 Reduced Media Independent Interface (RMII) Connections

Figure 3 shows a device with integrated EMAC and MDIO interfaced via a RMII connection. This interface is only available in 10 Mbps and 100 Mbps modes. The RMII interface is only supported in full-duplex mode for the C645x family of devices.

**Figure 3. Ethernet Configuration with RMII Interface**



The RMII interface has the same functionality as the MII, but it does so with a reduced number of pins, thus lowering the total cost for an application. In devices incorporating many PHY interfaces such as switches, the number of pins can add significant cost as the port counts increase. [Table 3](#) summarizes the individual EMAC and MDIO signals for the RMII interface.

The RMII interface does not include an MCOL signal. A collision is detected from the receive and transmit data delimiters. The data signals are 2 bits wide, and a single reference clock must be provided to the MAC, operating at 50MHz to sustain the same data rate as MII.

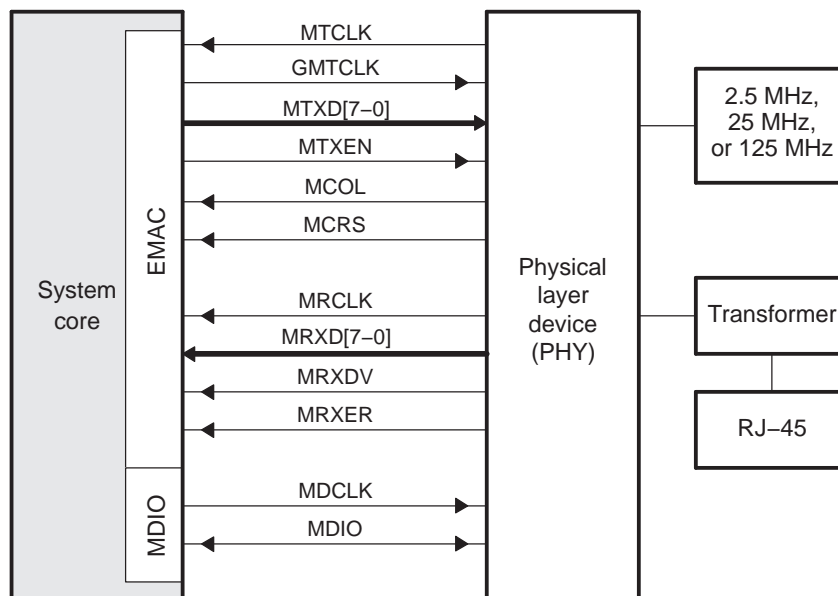
**Table 3. EMAC and MDIO Signals for RMII Interface**

Signal Name	I/O	Description
MTXD[1-0]	O	Transmit data (MTXD). The transmit data pins are a collection of 2 data signals comprising 2 bits of data. MTDX0 is the least-significant bit (LSB). The signals are synchronized to the RMII reference clock and valid only when MTXEN is asserted.
MTXEN	O	Transmit enable (MTXEN). The transmit enable signal indicates that the MTXD pins are generating nibble data for use by the PHY. It is driven synchronously to the RMII reference clock.
MCRSDV	I	Carrier sense/receive data valid (MCRSDV). The MCRSDV pin is asserted by the PHY when the network is not idle in either transmit or receive. The data on MRXD is considered valid once the MCRSDV signal is asserted. The pin is de-asserted when both transmit and receive are idle. The assertion of this signal is asynchronous to the RMII reference clock. This pin is used in half-duplex operation only.
MREFCLK	I	Reference clock (MREFCLK). A 50MHz clock must be provided through this pin for RMII operation.
MRXD[1-0]	I	Receive data (MRXD). The receive data pins are a collection of 2 data signals comprising 2 bits of data. MRDX0 is the least-significant bit (LSB). The signals are synchronized to the RMII reference clock and valid only when MCRSDV is asserted. In 10 Mbps operation, MRXD is sampled every 10th cycle of the RMII reference clock.
MRXER	I	Receive error (MRXER). The receive error signal is asserted for one or more reference clock periods to indicate that an error was detected in the received frame. This is meaningful only during data reception when MCRSDV is active. It is driven synchronously to the RMII reference clock.
MDCLK	O	Management data clock (MDCLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO pin. The frequency of this clock is controlled by the CLKDIV bits in the MDIO control register (CONTROL).
MDIO	I/O	Management data input output (MDIO). The MDIO pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO pin acts as an output for everything except the data bit cycles, when the pin acts as an input for read operations.

### 2.3.3 Gigabit Media Independent Interface (GMII) Connections

Figure 4 shows a device with integrated EMAC and MDIO interfaced via a GMII connection. This interface is available in 10 Mbps, 100 Mbps, and 1000 Mbps modes.

**Figure 4. Ethernet Configuration with GMII Interface**



The GMII interface supports 10/100/1000 Mbps modes. Only full-duplex mode is available in 1000 Mbps mode. In 10/100 Mbps modes, the GMII interface acts like an MII interface, and only the lower 4 bits of data are transferred for each of the data buses.

Table 4 summarizes the individual EMAC and MDIO signals for the GMII interface.

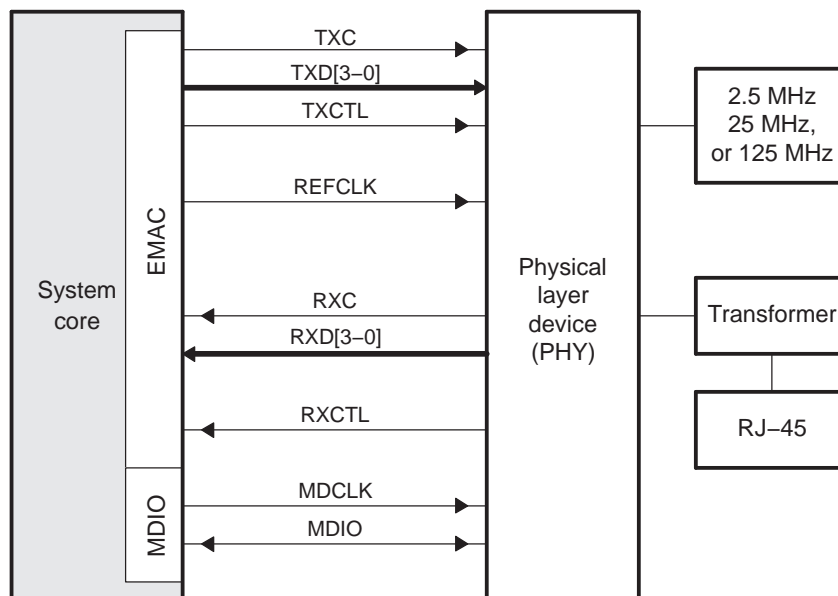
**Table 4. EMAC and MDIO Signals for GMII Interface**

Signal Name	I/O	Description
MTCLK	I	Transmit clock (MTCLK). The transmit clock is a continuous clock that provides the timing reference for transmit operations in 10/100 Mbps mode. The MTXD and MTXEN signals are tied to this clock when in 10/100 Mbps mode. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation, and 25 MHz at 100 Mbps operation.
GMTCLK	O	GMII source synchronous transmit clock (GMTCLK). This clock is used in 1000 Mbps mode only, providing a continuous 125 MHz frequency for transmit operations. The MTXD and MTXEN signals are tied to this clock when in Gigabit mode. The clock is generated by the EMAC and is 125 MHz.
MTXD[7-0]	O	Transmit data (MTXD). The transmit data pins are a collection of 8 data signals comprising 8 bits of data. MTDX0 is the least-significant bit (LSB). The signals are synchronized by MTCLK in 10/100 Mbps mode, and by GMTCLK in Gigabit mode, and valid only when MTXEN is asserted.
MTXEN	O	Transmit enable (MTXEN). The transmit enable signal indicates that the MTXD pins are generating nibble data for use by the PHY. It is driven synchronously to MTCLK in 10/100 Mbps mode, and to GMTCLK in Gigabit mode.
MCOL	I	Collision detected (MCOL). The MCOL pin is asserted by the PHY when it detects a collision on the network. It remains asserted while the collision condition persists. This signal is not necessarily synchronous to MTCLK nor MRCLK. This pin is used in half-duplex operation only.
MCRS	I	Carrier sense (MCRS). The MCRS pin is asserted by the PHY when the network is not idle in either transmit or receive. The pin is de-asserted when both transmit and receive are idle. This signal is not necessarily synchronous to MTCLK nor MRCLK. This pin is used in half-duplex operation only.
MRCLK	I	Receive clock (MRCLK). The receive clock is a continuous clock that provides the timing reference for receive operations. The MRXD, MRXDV, and MRXER signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation, 25 MHz at 100 Mbps operation and 125 MHz at 1000 Mbps operation.
MRXD[7-0]	I	Receive data (MRXD). The receive data pins are a collection of 8 data signals comprising 8 bits of data. MRDX0 is the least-significant bit (LSB). The signals are synchronized by MRCLK and valid only when MRXDV is asserted.
MRXDV	I	Receive data valid (MRXDV). The receive data valid signal indicates that the MRXD pins are generating nibble data for use by the EMAC. It is driven synchronously to MRCLK.
MRXER	I	Receive error (MRXER). The receive error signal is asserted for one or more MRCLK periods to indicate that an error was detected in the received frame. This is meaningful only during data reception when MRXDV is active.
MDCLK	O	Management data clock (MDCLK). The MDIO data clock is sourced by the MDIO module on the system. It is used to synchronize MDIO data access operations done on the MDIO pin. The frequency of this clock is controlled by the CLKDIV bits in the MDIO control register (CONTROL).
MDIO	I/O	Management data input output (MDIO). The MDIO pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO pin acts as an output for everything except the data bit cycles, when the pin acts as an input for read operations.

### 2.3.4 Reduced Gigabit Media Independent Interface (RGMII) Connections

Figure 5 shows a device with integrated EMAC and MDIO interfaced via a RGMII connection. This interface is available in 10 Mbps, 100 Mbps, and 1000 Mbps modes.

**Figure 5. Ethernet Configuration with RGMII Interface**



The RGMII interface is a reduced pin alternative to the GMII interface. The data paths are reduced, control signals are multiplexed together, and both edges of the clock are used.

The RGMII interface does not include a MCOL and a MCRS signal for half-duplex mode (only available in 10/100 Mbps mode).

Carrier sense (MCRS) is indicated by one of the following cases instead:

- MRXDV signal (multiplexed in the RXCTL signal) is true
- MRXDV is false, MRXERR (multiplexed in the RXCTL signal) is true, and a value of FFh exists on the RXD[7:0] simultaneously

Table 5 summarizes the individual EMAC and MDIO signals for the RGMII interface.

**Table 5. EMAC and MDIO Signals for RGMII Interface**

Signal Name	I/O	Description
TXC	O	Transmit clock (TXC). The transmit clock is a continuous clock that provides the timing reference for transmit operations. The TXD and TXCTL signals are tied to this clock. The clock is driven by the EMAC and is 2.5 MHz at 10 Mbps operation, 25 MHz at 100 Mbps operation, and 125 MHz at 1000 Mbps operation.
TXD[3-0]	O	Transmit data (TXD). The transmit data pins are a collection of 4 data signals comprising 4 bits of data. TDX0 is the least-significant bit (LSB). The signals are synchronized by TXC and valid only when TXCTL is asserted. The lower 4 bits of data are transmitted on the rising edge of the clock, and the higher 4 bits of data are transmitted on the falling edge of the TXC.
TXCTL	O	Transmit enable (TXCTL). The transmit enable signal indicates that the TXD pins are generating nibble data for use by the PHY. It is driven synchronously to TXC.
REFCLK	O	Reference clock (REFCLK). This 125 MHz reference clock is provided as a convenience. It can be used as a clock source to the PHY, so that the PHY may generate the RXC clock to be sent to EMAC. This clock is stopped while the device is in reset.
RXC	I	Receive clock (RXC). The receive clock is a continuous clock that provides the timing reference for receive operations. The RXD, and RXCTL signals are tied to this clock. The clock is generated by the PHY and is 2.5 MHz at 10 Mbps operation, 25 MHz at 100 Mbps operation, and 125 MHz at 1000 Mbps operation.
RXD[3-0]	I	Receive data (RXD). The receive data pins are a collection of 4 data signals comprising 4 bits of data. RDX0 is the least-significant bit (LSB). The signals are synchronized by RXC and valid only when RXCTL is asserted. The lower 4 bits of data are received on the rising edge of the clock, and the higher 4 bits of data are received on the falling edge of the RXC.
RXCTL	I	Receive control (RXCTL). The receive control data has the receive data valid (MRXDV) signal on the rising edge of the receive clock, and a derivative of receive data valid and receive error (MRXER) on the falling edge of RXC. When receiving a valid frame with no errors, MRXDV = TRUE is generated as a logic high on the rising edge on RXC and MRXER = FALSE is generated as a logic high on the falling edge of RXC. When no frame is being received, MRXDV = FALSE is generated as a logic low on the rising edge of RXC and MRXER = FALSE is generated as a logic low on the falling edge of RXC. When receiving a valid frame with errors, MRXDV = TRUE is generated as a logic high on the rising edge of RXC and MRXER = TRUE is generated as a logic low on the falling edge of RXC.
MDCLK	O	Management data clock (MDCLK). The MDIO data clock is sourced by the MDIO module. It synchronizes MDIO data access operations done on the MDIO pin. The frequency of this clock is controlled by the CLKDIV bits in the MDIO control register (CONTROL).
MDIO	I/O	Management data input output (MDIO). The MDIO pin drives PHY management data into and out of the PHY by way of an access frame consisting of start of frame, read/write indication, PHY address, register address, and data bit cycles. The MDIO pin acts as an output for everything except the data bit cycles, when the pin acts as an input for read operations.

## 2.4 Ethernet Protocol Overview

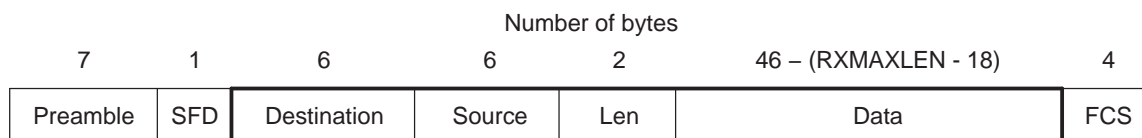
Ethernet provides an unreliable, connectionless service to a networking application. A brief overview of the ethernet protocol follows. For more information on the carrier sense multiple access with collision detection (CSMA/CD) access method (ethernet’s multiple access protocol), see the IEEE 802.3 standard document.

### 2.4.1 Ethernet Frame Format

All the ethernet technologies use the same frame structure. The format of an ethernet frame is shown in [Figure 6](#) and described in [Table 6](#). The ethernet packet is the collection of bytes representing the data portion of a single ethernet frame on the wire (shown outlined in bold in [Figure 6](#)).

The ethernet frames are of variable lengths, with no frame smaller than 64 bytes or larger than RXMAXLEN bytes (header, data, and CRC).

**Figure 6. Ethernet Frame**



Legend: SFD=Start Frame Delimiter; FCS=Frame Check Sequence (CRC)

**Table 6. Ethernet Frame Description**

Field	Bytes	Description
Preamble	7	These 7 bytes have a fixed value of 55h. They wake up the receiving EMAC ports and synchronize their clocks to that of the sender’s clock.
Start of Frame Delimiter	1	This field with a value of 5Dh immediately follows the preamble pattern and indicates the start of important data.
Destination address	6	This field contains the Ethernet MAC address of the intended EMAC port for the frame. It may be an individual or multicast (including broadcast) address. If the destination EMAC port receives an Ethernet frame with a destination address that does not match any of its MAC physical addresses, and no promiscuous, multicast or broadcast channel is enabled, it discards the frame.
Source address	6	This field contains the MAC address of the Ethernet port that transmits the frame to the Local Area Network.
Length/Type	2	The length field indicates the number of EMAC client data bytes contained in the subsequent data field of the frame. This field can also be used to identify the data type carried by the frame.
Data	46 to (RXMAXLEN - 18)	This field carries the datagram containing the upper layer protocol frame (the IP layer datagram). The maximum transfer unit (MTU) of Ethernet is (RXMAXLEN - 18) bytes. Therefore, if the upper layer protocol datagram exceeds (RXMAXLEN - 18) bytes, the host must fragment the datagram and send it in multiple Ethernet packets. The minimum size of the data field is 46 bytes. Thus, if the upper layer datagram is less than 46 bytes, the data field must be extended to 46 bytes by appending extra bits after the data field, but prior to calculating and appending the FCS.
Frame Check Sequence	4	A cyclic redundancy check (CRC) is used by the transmit and receive algorithms to generate a CRC value for the FCS field. The frame check sequence covers the 60 to (RXMAXLEN - 4) bytes of the packet data. Note that the 4-byte FCS field may not be included as part of the packet data, depending on the EMAC configuration.



### 2.4.2 Multiple Access Protocol

Nodes in an ethernet local area network are interconnected by a broadcast channel. As a result, when an EMAC port transmits a frame, all of the adapters on the local network receive the frame. Carrier sense multiple access with collision detection (CSMA/CD) algorithms are used when the EMAC operates in half-duplex mode. When operating in full-duplex mode, there is no contention for use of a shared medium, because there are exactly two ports on the local network.

Each port runs the CSMA/CD protocol without explicit coordination with the other ports on the ethernet network.

Within a specific port, the CSMA/CD protocol is as follows:

1. The port obtains data from upper layer protocols at its node, prepares an ethernet frame, and puts the frame in a buffer.
2. If the port senses that the medium is idle, it starts to transmit the frame. If the port senses that the transmission medium is busy, it waits until it senses no signal energy (plus an inter-packet gap time) and then starts to transmit the frame.
3. While transmitting, the port monitors for the presence of signal energy coming from other ports. If the port transmits the entire frame without detecting signal energy from other ethernet devices, the port is finished with the frame.
4. If the port detects signal energy from other ports while transmitting, it stops transmitting its frame and instead transmits a 48-bit jam signal.
5. After transmitting the jam signal, the port enters an exponential back off phase. Specifically, when transmitting a given frame, after experiencing a number of collisions in a row for the frame, the port chooses a random value that is dependent on the number of collisions. The port then waits an amount of time which is multiple of this random value, and returns to Step 1.

## 2.5 Programming Interface

### 2.5.1 Packet Buffer Descriptors

The buffer descriptor is a central part of the EMAC module. It determines how the application software describes ethernet packets to be sent and empty buffers to be filled with incoming packet data.

The basic descriptor format is shown in [Figure 7](#) and described in [Table 7](#).

**Figure 7. Basic Descriptor Format**

Word Offset	Bit Fields	
	31	16   15   0
0	Next Descriptor Pointer	
1	Buffer Pointer	
2	Buffer Offset	Buffer Length
3	Flags	Packet Length

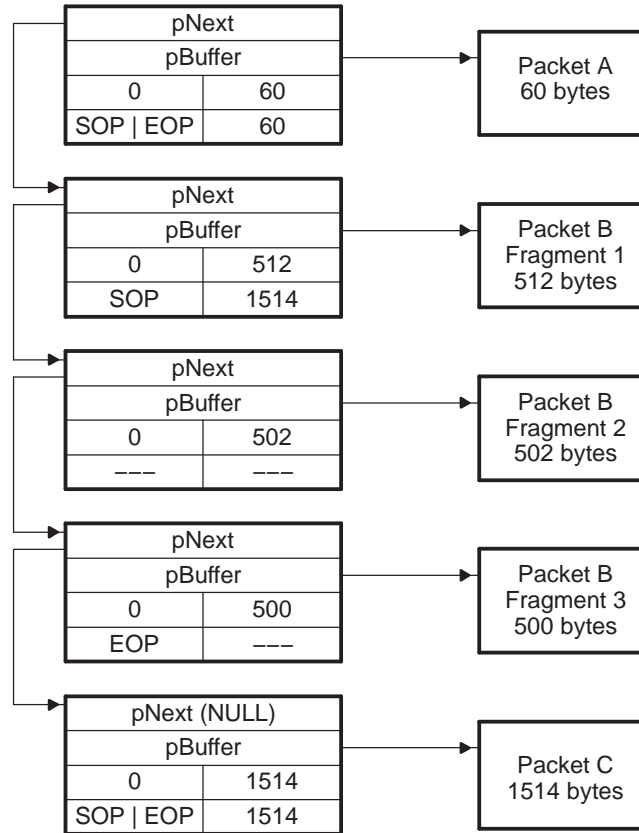
**Table 7. Basic Descriptors**

Word Offset	Field	Field Description
0	Next Descriptor Pointer	The next descriptor pointer creates a single linked list of descriptors. Each descriptor describes a packet or a packet fragment. When a descriptor points to a single buffer packet or the first fragment of a packet, the start of packet (SOP) flag is set in the flags field. When a descriptor points to a single buffer packet or the last fragment of a packet, the end of packet (EOP) flag is set. When a packet is fragmented, each fragment must have its own descriptor and appear sequentially in the descriptor linked list.
1	Buffer Pointer	The buffer pointer refers to the memory buffer that either contains packet data during transmit operations, or is an empty buffer ready to receive packet data during receive operations.
2	Buffer Offset	The buffer offset is the offset from the start of the packet buffer to the first byte of valid data. This field only has meaning when the buffer descriptor points to a buffer that contains data.
2	Buffer Length	The buffer length is the number of valid packet data bytes stored in the buffer. If the buffer is empty and waiting to receive data, this field represents the size of the empty buffer.
3	Flags	The flags field contains more information about the buffer, such as whether it is the first fragment in a packet (SOP), the last fragment in a packet (EOP), or contains an entire contiguous Ethernet packet (both SOP and EOP). <a href="#">Section 2.5.4</a> and <a href="#">Section 2.5.5</a> describe the flags.
3	Packet Length	The packet length only has meaning for buffers that both contain data and are the start of a new packet (SOP). For SOP descriptors, the packet length field contains the length of the entire Ethernet packet, even if it is contained in a single buffer or fragmented over several buffers.

For example, consider three packets to be transmitted, Packet A is a single fragment (60 bytes), Packet B is fragmented over three buffers (1514 bytes total), and Packet C is a single fragment (1514 bytes).

Figure 8 shows the linked list of descriptors to describe these three packets.

**Figure 8. Typical Descriptor Linked List**



## 2.5.2 Transmit and Receive Descriptor Queues

The EMAC module processes descriptors in linked list chains (Section 2.5.1). The lists controlled by the EMAC are maintained by the application software via the head descriptor pointer (HDP) registers. Since the EMAC supports eight channels for both transmit and receive, there are eight head descriptor pointer registers for both.

They are designated as follows:

- TX $n$ HDP: Transmit Channel  $n$  DMA Head Descriptor Pointer Register
- RX $n$ HDP: Receive Channel  $n$  DMA Head Descriptor Pointer Register

After an EMAC reset, and before enabling the EMAC for send or receive, you must initialize all 16 head descriptor pointer registers to zero.

The EMAC uses a simple system to determine ownership of a descriptor (either the EMAC or the application software). There is a flag in the descriptor flags field called OWNER. When this flag is set, the EMAC owns the referenced packet.

---

**Note:** Ownership is assigned on a packet-based granularity, not on descriptor granularity. Thus, only SOP descriptors make use of the OWNER flag. The EMAC patches the SOP descriptor of the corresponding packet and clears the OWNER flag as packets are processed. This means that the EMAC is finished processing all descriptors up to and including the first with the EOP flag set. This indicates that you have reached the end of the packet. This may only be one descriptor with both the SOP and EOP flags set.

---

To add a descriptor or a linked list of descriptors to an EMAC descriptor queue for the first time, the software application writes the pointer to the descriptor or first descriptor of a list to the corresponding HDP register. Note that the last descriptor in the list must have its *next* pointer cleared so that the EMAC can detect the end of the list. If only a single descriptor is added, its *next descriptor* pointer must be initialized to zero.

The HDP register must never be written to a second time while a previous list is active. To add additional descriptors to a descriptor list already owned by the EMAC, the NULL *next* pointer of the last descriptor of the previous list is patched with a pointer to the first descriptor in the new list. The list of new descriptors to be appended to the existing list must itself be NULL terminated before the pointer patch is performed.

If the EMAC reads the *next* pointer of a descriptor as NULL in the instant before an application appends additional descriptors to the list by patching the pointer, this may result in a race condition. Thus, the software application must always examine the Flags field of all EOP packets, looking for a special flag called end of queue (EOQ). The EOQ flag is set by the EMAC on the last descriptor of a packet when the descriptor's *next* pointer is NULL, allowing the EMAC to indicate to the software application that it has reached the end of the list. When the software application sees the EOQ flag set, and there are more descriptors to process, the application may then submit the new list or missed list portion by writing the new list pointer to the same HDP register that started the process.

This process applies when adding packets to a transmit list, and empty buffers to a receive list.

### 2.5.3 Transmit and Receive EMAC Interrupts

The EMAC processes descriptors in linked list chains ([Section 2.5.1](#)), using the linked list queue mechanism ([Section 2.5.2](#)).

The EMAC synchronizes the descriptor list processing by using interrupts to the software application. The interrupts are controlled by the application by using the interrupt masks, global interrupt enable, and the completion pointer register (CP). This register is also called interrupt acknowledge register.

As the EMAC supports eight channels for both transmit and receive, there are eight CP registers for both. They are designated as:

- TX $n$ CP: Transmit Channel  $n$  Completion Pointer (Interrupt Acknowledge) Register
- RX $n$ CP: Receive Channel  $n$  Completion Pointer (Interrupt Acknowledge) Register

These registers serve two purposes. When read, they return the pointer to the last descriptor that the EMAC has processed. When written by the software application, the value represents the last descriptor processed by the software application. If these two values do not match, the interrupt is active.

The system configuration determines whether an active interrupt can interrupt the CPU. In general, the global interrupt for EMAC and MDIO must be enabled in the EMAC control module, and it also must be mapped in the DSP interrupt controller and enabled as a CPU interrupt. If the system is configured properly, the interrupt for a specific receive or transmit channel executes under these conditions when the corresponding interrupt is enabled in the EMAC using the RXINTMASKSET or TXINTMASKSET registers.

The current state of the receive or transmit channel interrupt can be examined directly by the software application by reading the RXINTSTATRAW and TXINTSTATRAW registers, whether or not the interrupt is enabled.

Interrupts are acknowledged when the application software updates the value of TX $n$ CP or RX $n$ CP with a value that matches the internal value kept by the EMAC.

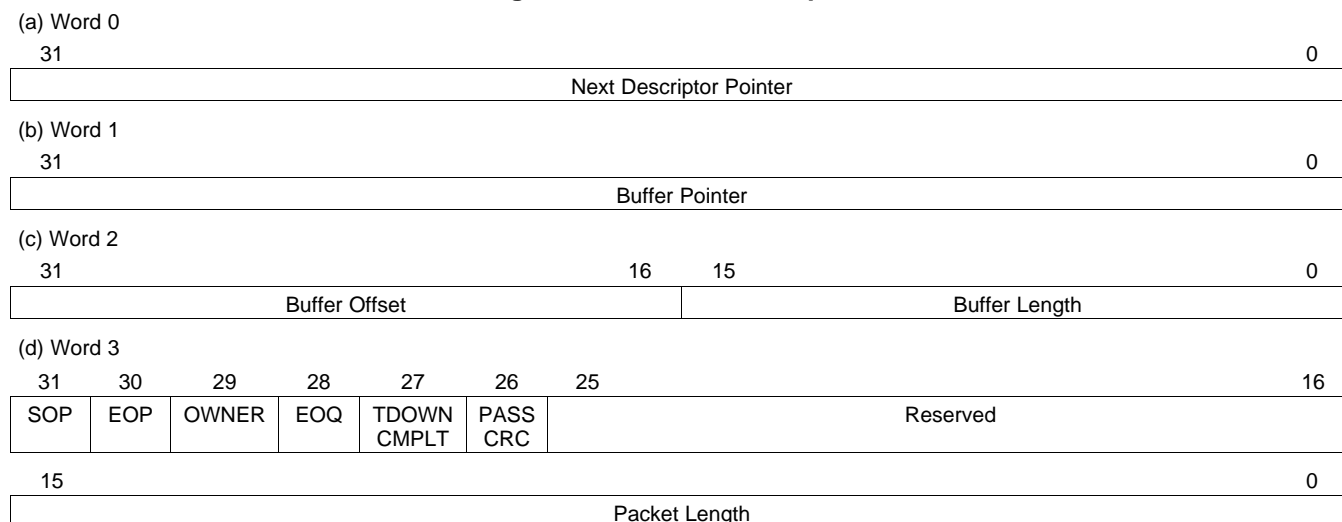
This mechanism ensures that the application software never misses an EMAC interrupt, as the interrupt and its acknowledgment are tied directly to the actual buffer descriptors processing.

## 2.5.4 Transmit Buffer Descriptor Format

A transmit (TX) buffer descriptor (Figure 9) is a contiguous block of four 32-bit data words aligned on a 32-bit boundary that describes a packet or a packet fragment.

Example 1 shows the transmit buffer descriptor described by a C structure.

**Figure 9. Transmit Descriptor Format**



### Example 1. Transmit Descriptor in C Structure Format

```

/*
// EMAC Descriptor
//
// The following is the format of a single buffer descriptor
// on the EMAC.
*/

typedef struct _EMAC_Desc {
    struct _EMAC_Desc *pNext; /* Pointer to next descriptor in chain */
    Uint8 *pBuffer; /* Pointer to data buffer */
    Uint32 BufOffLen; /* Buffer Offset(MSW) and Length(LSW) */
    Uint32 PktFlgLen; /* Packet Flags(MSW) and Length(LSW) */
} EMAC_Desc;

/* Packet Flags */
#define EMAC_DSC_FLAG_SOP 0x80000000u
#define EMAC_DSC_FLAG_EOP 0x40000000u
#define EMAC_DSC_FLAG_OWNER 0x20000000u
#define EMAC_DSC_FLAG_EOQ 0x10000000u
#define EMAC_DSC_FLAG_TDOWNCMPLT 0x08000000u
#define EMAC_DSC_FLAG_PASSCRC 0x04000000u

```

#### 2.5.4.1 Next Descriptor Pointer

The next descriptor pointer indicates the 32-bit word aligned memory address of the next buffer descriptor in the transmit queue. The pointer creates a linked list of buffer descriptors. If the value of this pointer is zero, then the current buffer is the last buffer in the queue. The software application must set this value prior to adding the descriptor to the active transmit list. The pointer is not altered by the EMAC.

The value of pNext should never be altered once the descriptor is in an active transmit queue, unless its current value is NULL. If the pNext pointer is initially NULL, and more packets need to be queued for transmit, the software application may alter this pointer to point to a newly appended descriptor. The EMAC will use the new pointer value and proceed to the next descriptor unless the pNext value has already been read. If the pNext value has already been read, the transmitter will halt on the specified transmit channel, and the software application may restart it then. The software can detect this issue by searching for an end of queue (EOQ) condition flag on the updated packet descriptor when it is returned by the EMAC.

#### 2.5.4.2 Buffer Pointer

The buffer pointer is the byte-aligned memory address of the memory buffer associated with the buffer descriptor. The software application must set this value prior to adding the descriptor to the active transmit list. This pointer is not altered by the EMAC.

#### 2.5.4.3 Buffer Offset

This 16-bit field indicates how many unused bytes are at the start of the buffer. For example, a value of 0000h indicates that no unused bytes are at the start of the buffer and that valid data begins on the first byte of the buffer. A value of 000Fh indicates that the first 15 bytes of the buffer are to be ignored by the EMAC and that valid buffer data starts on byte 16 of the buffer. The software application must set this value prior to adding the descriptor to the active transmit list. This field is not altered by the EMAC.

Note that this value is only checked on the first descriptor of a given packet (where the SOP flag is set). It cannot specify the offset of subsequent packet fragments. Also, as the buffer pointer may point to any byte-aligned address, this field may be unnecessary, depending on the device driver architecture.

The range of legal values for this field is 0 to (Buffer Length – 1).

#### 2.5.4.4 Buffer Length

This 16-bit field indicates how many valid data bytes are in the buffer. On single fragment packets, this value is also the total length of the packet data to be transmitted. If the buffer offset field is used, the offset bytes are not counted as part of this length. This length counts only valid data bytes. The software application must set this value prior to adding the descriptor to the active transmit list. This field is not altered by the EMAC.

#### 2.5.4.5 Packet Length

This 16-bit field specifies the number of data bytes in the entire packet. Any leading buffer offset bytes are not included. The sum of the buffer length fields of each of the packet's fragments (if more than one) must be equal to the packet length. The software application must set this value prior to adding the descriptor to the active transmit list. This field is not altered by the EMAC. This value is only checked on the first descriptor of a given packet, where the SOP flag is set.

#### 2.5.4.6 Start of Packet (SOP) Flag

When set, this flag indicates that the descriptor points to a packet buffer that is the start of a new packet. For a single fragment packet, both the SOP and end of packet (EOP) flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet sets the EOP flag. This bit is set by the software application and is not altered by the EMAC.

#### **2.5.4.7 End of Packet (EOP) Flag**

When set, this flag indicates that the descriptor points to the last packet buffer for a given packet. For a single fragment packet, both the start of packet (SOP) and EOP flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet sets the EOP flag. This bit is set by the software application and is not altered by the EMAC.

#### **2.5.4.8 Ownership (OWNER) Flag**

When set, this flag indicates that all the descriptors for the given packet (from SOP to EOP) are currently owned by the EMAC. This flag is set by the software application on the SOP packet descriptor before adding the descriptor to the transmit descriptor queue. For a single fragment packet, the SOP, EOP, and OWNER flags are all set. The OWNER flag is cleared by the EMAC once it is finished with all the descriptors for the given packet. Note that this flag is valid on SOP descriptors only.

#### **2.5.4.9 End of Queue (EOQ) Flag**

When set, this flag indicates that the descriptor in question was the last descriptor in the transmit queue for a given transmit channel, and that the transmitter has halted. This flag is initially cleared by the software application prior to adding the descriptor to the transmit queue. This bit is set by the EMAC when the EMAC identifies that a descriptor is the last for a given packet (the EOP flag is set), and there are no more descriptors in the transmit list (next descriptor pointer is NULL).

The software application can use this bit to detect when the EMAC transmitter for the corresponding channel has halted. This is useful when the application appends additional packet descriptors to a transmit queue list that is already owned by the EMAC. Note that this flag is valid on EOP descriptors only.

#### **2.5.4.10 Teardown Complete (TDOWNCMPLT) Flag**

This flag is used when a transmit queue is being torn down, or aborted, instead of allowing transmission, such as during device driver reset or shutdown conditions. The EMAC sets this bit in the SOP descriptor of each packet as it is aborted from transmission.

Note that this flag is valid on SOP descriptors only. Also note that only the first packet in an unsent list has the TDOWNCMPLT flag set. The EMAC does not process subsequent descriptors.

#### **2.5.4.11 Pass CRC (PASSCRC) Flag**

The software application sets this flag in the SOP packet descriptor before it adds the descriptor to the transmit queue. Setting this bit indicates to the EMAC that the 4-byte Ethernet CRC is already present in the packet data, and that the EMAC should not generate its own version of the CRC.

When the CRC flag is cleared, the EMAC generates and appends the 4-byte CRC. The buffer length and packet length fields do not include the CRC bytes. When the CRC flag is set, the 4-byte CRC is supplied by the software application and is appended to the end of the packet data. The buffer length and packet length fields include the CRC bytes, as they are part of the valid packet data. Note that this flag is valid on SOP descriptors only.

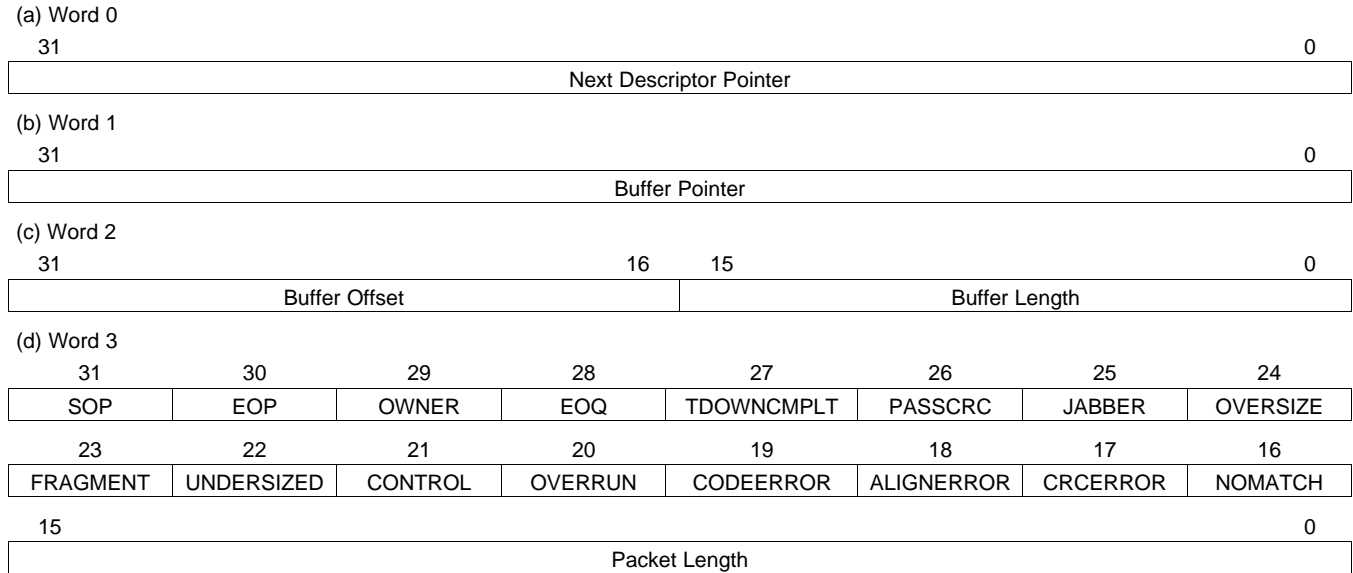


## 2.5.5 Receive Buffer Descriptor Format

A receive (RX) buffer descriptor (Figure 10) is a contiguous block of four 32-bit data words aligned on a 32-bit boundary that describes a packet or a packet fragment.

Example 2 shows the receive descriptor described by a C structure.

**Figure 10. Receive Descriptor Format**



**Example 2. Receive Descriptor in C Structure Format**

```

/*
// EMAC Descriptor
//
// The following is the format of a single buffer descriptor
// on the EMAC.
*/

typedef struct _EMAC_Desc {
    struct _EMAC_Desc *pNext; /* Pointer to next descriptor in chain */
    Uint8 *pBuffer; /* Pointer to data buffer */
    Uint32 BufOffLen; /* Buffer Offset(MSW) and Length(LSW) */
    Uint32 PktFlgLen; /* Packet Flags(MSW) and Length(LSW) */
} EMAC_Desc;

/* Packet Flags */
#define EMAC_DSC_FLAG_SOP 0x80000000u
#define EMAC_DSC_FLAG_EOP 0x40000000u
#define EMAC_DSC_FLAG_OWNER 0x20000000u
#define EMAC_DSC_FLAG_EOQ 0x10000000u
#define EMAC_DSC_FLAG_TDOWNCMLPT 0x08000000u
#define EMAC_DSC_FLAG_PASSCRC 0x04000000u
#define EMAC_DSC_FLAG_JABBER 0x02000000u
#define EMAC_DSC_FLAG_OVERSIZE 0x01000000u
#define EMAC_DSC_FLAG_FRAGMENT 0x00800000u
#define EMAC_DSC_FLAG_UNDERSIZED 0x00400000u
#define EMAC_DSC_FLAG_CONTROL 0x00200000u
#define EMAC_DSC_FLAG_OVERRUN 0x00100000u
#define EMAC_DSC_FLAG_CODEERROR 0x00080000u
#define EMAC_DSC_FLAG_ALIGNERROR 0x00040000u
#define EMAC_DSC_FLAG_CRCERROR 0x00020000u
#define EMAC_DSC_FLAG_NOMATCH 0x00010000u
  
```

### 2.5.5.1 Next Descriptor Pointer

The next descriptor pointer indicates the 32-bit word aligned memory address of the next buffer descriptor in the receive queue. The pointer creates a linked list of buffer descriptors. If the value of the pointer is zero, then the current buffer is the last buffer in the queue. The software application must set this value prior to adding the descriptor to the active receive list. This pointer is not altered by the EMAC.

The value of pNext should never be altered once the descriptor is in an active receive queue, unless its current value is NULL. If the pNext pointer is initially NULL, and more empty buffers can be added to the pool, the software application may alter this pointer to indicate a newly appended descriptor. The EMAC will use the new pointer value and proceed to the next descriptor unless the pNext value has already been read. If the pNext value has already been read, the receiver will halt the receive channel in question, and the software application may restart it at that time. The software can detect this case by searching for an end of queue (EOQ) condition flag on the updated packet descriptor when it is returned by the EMAC.

### 2.5.5.2 Buffer Pointer

The buffer pointer is the byte-aligned memory address of the memory buffer associated with the buffer descriptor. The software application must set this value prior to adding the descriptor to the active receive list. This pointer is not altered by the EMAC.

### 2.5.5.3 Buffer Offset

This 16-bit field must be initialized to zero by the software application before adding the descriptor to a receive queue.

This field will be updated depending on the RXBUFFEROFFSET register setting. When the offset register is set to a non-zero value, the received packet is written to the packet buffer at an offset given by the value of the register, and this value is also written to the buffer offset field of the descriptor.

When a packet is fragmented over multiple buffers because it does not fit in the first buffer supplied, the buffer offset only applies to the first buffer in the list, which is where the start of packet (SOP) flag is set in the corresponding buffer descriptor. In other words, the buffer offset field is only updated by the EMAC on SOP descriptors.

The range of legal values for the BUFFEROFFSET register is 0 to (Buffer Length – 1) for the smallest value of buffer length for all descriptors in the list.

### 2.5.5.4 Buffer Length

This 16-bit field has two functions:

- Before the descriptor is first placed on the receive queue by the application software, the software initializes the buffer length field with the physical size of the empty data buffer specified by the buffer pointer field.
- After the empty buffer has been processed by the EMAC and filled with received data bytes, the EMAC updates the buffer length field to reflect the actual number of valid data bytes written to the buffer.

### 2.5.5.5 Packet Length

This 16-bit field specifies the number of data bytes in the entire packet. The software application initializes this value to zero for empty packet buffers. The EMAC fills in the value on the first buffer used for a given packet, as signified by the EMAC setting a start of packet (SOP) flag. The EMAC sets the packet length on all SOP buffer descriptors.

#### **2.5.5.6 Start of Packet (SOP) Flag**

When set, this flag indicates that the descriptor points to the starting packet buffer of a new packet. For a single fragment packet, both the SOP and end of packet (EOP) flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet has the EOP flag set. The software application initially clears this flag before adding the descriptor to the receive queue. The EMAC sets this bit on SOP descriptors.

#### **2.5.5.7 End of Packet (EOP) Flag**

When set, this flag indicates that the descriptor points to the last packet buffer for a given packet. For a single fragment packet, both the start of packet (SOP) and EOP flags are set. Otherwise, the descriptor pointing to the last packet buffer for the packet has the EOP flag set. The software application initially clears this flag before adding the descriptor to the receive queue. The EMAC sets this bit on EOP descriptors.

#### **2.5.5.8 Ownership (OWNER) Flag**

When set, this flag indicates that the descriptor is currently owned by the EMAC. The software application sets this flag before adding the descriptor to the receive descriptor queue. The EMAC clears this flag once it is finished with a given set of descriptors associated with a received packet. The EMAC updates the flag on SOP descriptor only. If the application identifies that the OWNER flag is cleared on an SOP descriptor, it may assume that the EMAC has released all descriptors up to and including the first with the EOP flag set. Note that for single buffer packets, the same descriptor will have both the SOP and EOP flags set.

#### **2.5.5.9 End of Queue (EOQ) Flag**

When set, this flag indicates that the specified descriptor was the last descriptor in the receive queue for a given receive channel, and that the corresponding receiver channel has halted. The software application initially clears this flag prior to adding the descriptor to the receive queue. The EMAC sets this bit when the EMAC identifies that a descriptor is the last for a given packet received (it also sets the EOP flag), and there are no more descriptors in the receive list (the next descriptor pointer is NULL).

The software application uses this bit to detect when the EMAC receiver for the corresponding channel has halted. This is useful when the application appends additional free buffer descriptors to an active receive queue. Note that this flag is valid on EOP descriptors only.

#### **2.5.5.10 Teardown Complete (TDOWNCMPLT) Flag**

This flag is used when a receive queue is being torn down, or aborted, instead of being filled with received data, such as during device driver reset or shutdown conditions. The EMAC sets this bit in the descriptor of the first free buffer when the teardown occurs. No additional queue processing is performed.

#### **2.5.5.11 Pass CRC (PASSCRC) Flag**

The EMAC sets this flag in the SOP buffer descriptor, if the received packet includes the 4-byte CRC. The software application must clear this flag before submitting the descriptor to the receive queue.

#### **2.5.5.12 Jabber Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet is a jabber frame and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE register.

#### **2.5.5.13 Oversize Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet is an oversized frame and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE register.

**2.5.5.14 Fragment Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet is only a packet fragment and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE register.

**2.5.5.15 Undersized Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet is undersized and was not discarded because the RXCSFEN bit was set in the RXMBPENABLE register.

**2.5.5.16 Control Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet is an EMAC control frame and was not discarded because the RXCMFEN bit was set in the RXMBPENABLE register.

**2.5.5.17 Overrun Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet was aborted due to a receive overrun.

**2.5.5.18 Code Error (CODEERROR) Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet contained a code error and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE register.

**2.5.5.19 Alignment Error (ALIGNERROR) Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet contained an alignment error and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE register.

**2.5.5.20 CRC Error (CRCERROR) Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet contained a CRC error and was not discarded because the RXCEFEN bit was set in the RXMBPENABLE register.

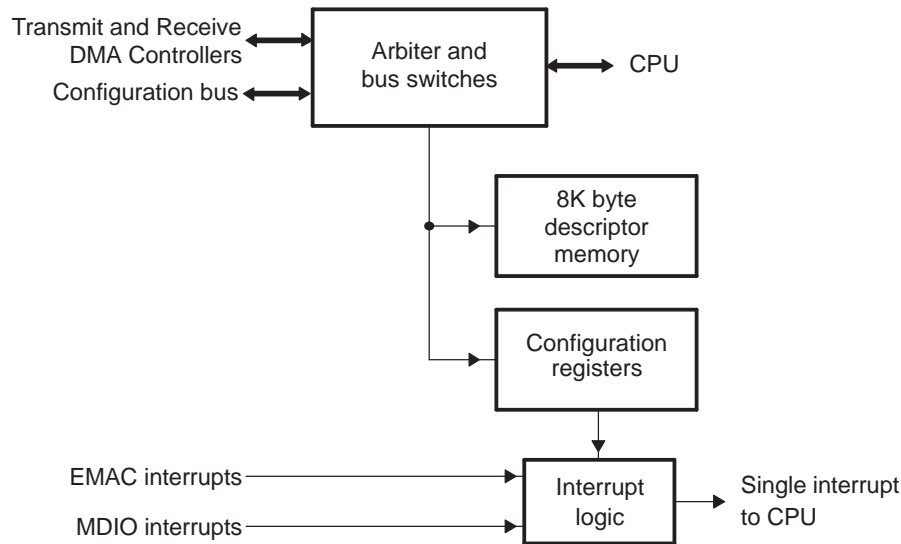
**2.5.5.21 No Match (NOMATCH) Flag**

The EMAC sets this flag in the SOP buffer descriptor if the received packet did not pass any of the EMAC's address match criteria and was not discarded because the RXCAFEN bit was set in the RXMBPENABLE register. Although the packet is a valid Ethernet data packet, it is only received because the EMAC is in promiscuous mode.

## 2.6 EMAC Control Module

The EMAC control module (Figure 11) interfaces the EMAC and MDIO modules to the rest of the system, and provides a local memory space to hold EMAC packet buffer descriptors. Local memory is used to avoid contention to device memory spaces. Other functions include the bus arbiter, and interrupt logic control.

**Figure 11. EMAC Control Module Block Diagram**



### 2.6.1 Internal Memory

The control module includes 8K bytes of internal memory. The internal memory block allows the EMAC to operate more independently of the CPU. It also prevents memory underflow conditions when the EMAC issues read or write requests to descriptor memory. (The EMAC's internal FIFOs protect memory accesses to read or write actual Ethernet packet data.)

A descriptor is a 16-byte memory structure that holds information about a single Ethernet packet buffer, which may contain a full or partial Ethernet packet. Thus, with the 8K memory block provided for descriptor storage, the EMAC module can send and receive up to a combined 512 packets before it must be serviced by application or driver software.

### 2.6.2 Bus Arbiter

The control module's bus arbiter operates transparently to the rest of the system. It arbitrates between the device core and EMAC buses for access to internal descriptor memory, and arbitrates between internal EMAC buses for access to system memory.

### 2.6.3 Interrupt Control

The EMAC control module combines the multiple interrupt conditions generated by the EMAC and MDIO modules into a single interrupt signal that is mapped to a CPU interrupt via the CPU interrupt controller.

The control module uses two registers to control the interrupt signal to the CPU. First, the INTEN bit in the EWCTL register globally enables and disables the interrupt signal to the CPU. The INTEN bit drives the interrupt line low during interrupt processing. Upon re-enabling the bit, the interrupt signal will rise if another interrupt condition exists, creating a rising edge detectable by the CPU.

The EMAC control module interrupt timer count register (EWINTTCNT) is programmed with a value that counts down once the EMAC/MDIO interrupts are enabled using EWCTL. The CPU interrupt signal is prevented from rising again until this count reaches zero.

The EWINTTCNT has no effect on interrupts once the count reaches zero, so there is no induced interrupt latency on random sporadic interrupts. However, the count delays the issuing of a second interrupt immediately after a first. This protects the system from entering interrupt thrashing mode, in which the software interrupt service routine (ISR) completes processing just in time to receive another interrupt. By postponing subsequent interrupts in a back-to-back condition, the software application or driver can perform more work in its ISR. The EWINTTCNT reset value can be adjusted from within the ISR according to current system load, or set to a fixed value that assures a maximum number of interrupts per second.

The counter counts at the peripheral clock frequency of CPU clock/6. The default reset count is 0 (inactive), the maximum value is 1 FFFFh (131 071).

## 2.7 Management Data Input/Output (MDIO) Module

The Management Data Input/Output (MDIO) module manages up to 32 physical layer (PHY) devices connected to the Ethernet Media Access Controller (EMAC). The MDIO module allows almost transparent operation of the MDIO interface with little maintenance from the CPU.

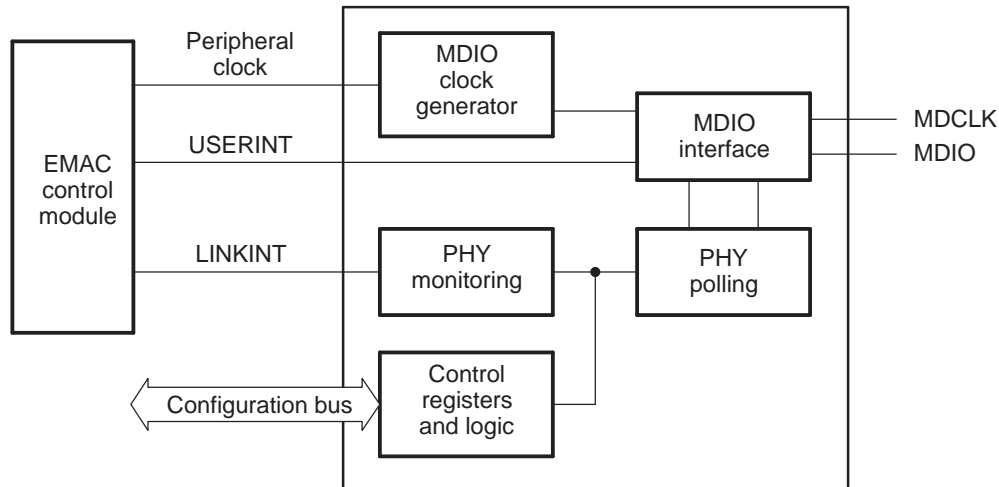
The MDIO module enumerates all PHY devices in the system by continuously polling 32 MDIO addresses. Once it detects a PHY device, the MDIO module reads the PHY status register to monitor the PHY link state. The MDIO module stores link change events that can interrupt the CPU. The event storage allows the CPU to poll the link status of the PHY device without continuously performing MDIO module accesses. However, when the system must access the MDIO module for configuration and negotiation, the MDIO module performs the MDIO read or write operation independent of the CPU. This independent operation allows the DSP to poll for completion or interrupt the CPU once the operation has completed.

### 2.7.1 MDIO Module Components

The MDIO module ([Figure 12](#)) interfaces to PHY components through two MDIO pins (MDCLK and MDIO), and to the DSP core through the EMAC control module and the configuration bus. The MDIO module consists of the following logical components:

- MDIO clock generator
- Global PHY detection and link state monitoring
- Active PHY monitoring
- PHY register user access

**Figure 12. MDIO Module Block Diagram**



### 2.7.1.1 MDIO Clock Generator

The MDIO clock generator controls the MDIO clock based on a divide-down of the peripheral clock (CPUclk/6) in the EMAC control module. The MDIO clock is specified to run up to 2.5 MHz, although typical operation would be 1.0 MHz. As the peripheral clock frequency is variable (CPUclk/6), the application software or driver controls the divide-down amount.

### 2.7.1.2 Global PHY Detection and Link State Monitoring

The MDIO module enumerates all PHY devices in the system by continuously polling all 32 MDIO addresses. The module tracks whether a PHY on a particular address has responded, and whether the PHY currently has a link. This information allows the software application to quickly determine which MDIO address the PHY is using, and if the system is using more than one PHY. The software application can then quickly switch between PHYs based on their current link status.

### 2.7.1.3 Active PHY Monitoring

Once a PHY candidate has been selected for use, the MDIO module transparently monitors its link state by reading the PHY status register. The MDIO device stores link change events that may optionally interrupt the CPU. Thus, the system can poll the link status of the PHY device without continuously performing MDIO accesses. Up to two PHY devices can be actively monitored at any given time.

### 2.7.1.4 PHY Register User Access

When the DSP must access the MDIO for configuration and negotiation, the PHY access module performs the actual MDIO read or write operation independent of the CPU. Thus, the CPU can poll for completion or receive an interrupt when the read or write operation has been performed. There are two user access registers (USERACCESS0 and USERACCESS1), allowing the software to submit up to two access requests simultaneously. The requests are processed sequentially.

## 2.7.2 MDIO Module Operational Overview

The MDIO module implements the 802.3 serial management interface to simultaneously interrogate and control up to two Ethernet PHYs, using a shared two-wired bus. It separately performs auto-detection and records the current link status of up to 32 PHYs, polling all 32 MDIO addresses.

Application software uses the MDIO module to configure the auto-negotiation parameters of the primary PHY attached to the EMAC, retrieve the negotiation results, and configure required parameters in the EMAC. Up to two Ethernet PHYs can be directly controlled and queried. The Media Independent Interface addresses of these two PHY devices are specified in the PHYADRMON fields of the USERPHYSEL $n$  register. The module can be programmed to trigger a CPU interrupt on a PHY link change event by setting the LINKINTENB bit in USERPHYSEL $n$ . Reads and writes to registers in these PHY devices are performed using the USERACCESS $n$  register.

The MDIO module powers up in an idle state until it is enabled by setting the ENABLE bit in the CONTROL register. This also configures the MDIO clock divider and preamble mode selection. The MDIO preamble is enabled by default, but it can be disabled if none of the connected PHYs require it.

Once the MDIO module is enabled, the MDIO interface state machine continuously polls the PHY link status (by reading the generic PHY Status register) of all possible 32 PHY addresses and records the results in the ALIVE and LINK registers. The corresponding bit for each PHY (0-31) is set in the ALIVE register if the PHY responded to the read request. The corresponding bit is set in the LINK register if the PHY responded and also is currently linked. In addition, any PHY register read transactions initiated by the application software using the USERACCESS $n$  register cause the ALIVE register to be updated.

The USERPHYSEL $n$  register is used to track the link status of any two of the 32 possible PHY addresses. Changes in the link status of the two monitored PHYs sets the appropriate bit in the LINKINTRAW and LINKINTMASKED registers, if they are enabled by the LINKINTENB bit in USERPHYSEL $n$ .

While the MDIO module is enabled, the host can issue a read or write transaction over the management interface using the DATA, PHYADR, REGADR, and WRITE bits in the USERACCESS $n$  register. When the application sets the GO bit in USERACCESS $n$ , the MDIO module begins the transaction without any further intervention from the CPU. Upon completion, the MDIO module clears the GO bit and sets the USERINTRAW[0-1] bit in the USERINTRAW register corresponding to the USERACCESS $n$  used. The corresponding USERINTMASKED bit in the USERINTMASKED register may also be set, depending on the mask setting configured in the USERINTMASKSET and USERINTMASKCLEAR registers.

A round-robin arbitration scheme schedules transactions that may be queued using both USERACCESS0 and USERACCESS1. The application software must verify the status of the GO bit in USERACCESS $n$  before initiating a new transaction to ensure that the previous transaction has completed. The application software can use the ACK bit in USERACCESS $n$  to determine the status of a read transaction.

### 2.7.2.1 Initializing the MDIO Module

To have the application software or device driver initialize the MDIO device, perform the following:

1. Configure the PREAMBLE and CLKDIV bits in the CONTROL register.
2. Enable the MDIO module by setting the ENABLE bit in the CONTROL register.
3. The ALIVE register can be read after a delay to determine which PHYs responded, and the LINK register can determine which of those (if any) already have a link.
4. Set up the appropriate PHY addresses in the USERPHYSEL $n$  register, and set the LINKINTENB bit to enable a link change event interrupt if desirable.
5. If an interrupt on a general MDIO register access is desired, set the corresponding bit in the USERINTMASKSET register to use the USERACCESS $n$  register. If only one PHY is to be used, the application software can set up one of the USERACCESS $n$  registers to trigger a completion interrupt. The other register is not set up.



### 2.7.2.2 Writing Data to a PHY Register

The MDIO module includes a user access register (USERACCESS $n$ ) to directly access a specified PHY device. To write a PHY register, perform the following:

1. Ensure that the GO bit in the USERACCESS $n$  register is cleared.
2. Write to the GO, WRITE, REGADR, PHYADR, and DATA bits in USERACCESS $n$  corresponding to the desired PHY and PHY register.
3. The write operation to the PHY is scheduled and completed by the MDIO module. Completion of the write operation can be determined by polling the GO bit in USERACCESS $n$  for a 0.
4. Completion of the operation sets the corresponding bit in the USERINTRAW register for the USERACCESS $n$  used. If interrupts have been enabled on this bit using the USERINTMASKSET register, then the bit is also set in the USERINTMASKED register and an interrupt is triggered on the DSP.

### 2.7.2.3 Reading Data From a PHY Register

The MDIO module includes a user access register (USERACCESS $n$ ) to directly access a specified PHY device. To read a PHY register, perform the following:

1. Ensure that the GO bit in the USERACCESS $n$  register is cleared.
2. Write to the GO, REGADR, and PHYADR bits in USERACCESS $n$  corresponding to the desired PHY and PHY register.
3. The read data value is available in the DATA bits of USERACCESS $n$  after the module completes the read operation on the serial bus. Completion of the read operation can be determined by polling the GO and ACK bits in USERACCESS $n$ . Once the GO bit has cleared, the ACK bit is set on a successful read.
4. Completion of the operation sets the corresponding bit in the USERINTRAW register for the USERACCESS $n$  used. If interrupts have been enabled on this bit using the USERINTMASKSET register, then the bit is also set in the USERINTMASKED register and an interrupt is triggered on the DSP.

### 2.7.2.4 Example of MDIO Register Access Code

The MDIO module uses the USERACCESS $n$  register to access the PHY control registers. Software functions that implement the access process include the following four macros:

PHYREG_read (regadr, phyadr )	Start the process of reading a PHY register
PHYREG_write(regadr, phyadr, data )	Start the process of writing a PHY register
PHYREG_wait ( )	Synchronize operation (make sure read/write is idle)
PHYREG_wait Results (results )	Wait for read to complete and return data read

It is not necessary to wait after a write operation, as long as the status is checked before every operation to make sure the MDIO hardware is idle. An alternative approach is to call PHYREG\_wait ( ) after every write, and PHYREG\_wait Results ( ) after every read, then the hardware can be assumed to be idle when starting a new operation.

The implementation of these macros using the register layer Chip Support Library (CSL) is shown in [Example 3](#) (USERACCESS0 is assumed).

Note that this implementation does not check the ACK bit on PHY register reads; in other words, it does not follow the procedure outlined in [Section 2.7.2.3](#). As the ALIVE register initially selects a PHY, it is assumed that the PHY is acknowledging read operations. It is possible that a PHY could become inactive at a future point in time. For example, a PHY can have its MDIO addresses changed while the system is running, although it is not a common occurrence. This condition can be tested by periodically checking the PHY state in the ALIVE register.

### Example 3. MDIO Register Access Macros

```

#define PHYREG_read(regadr, phyadr)                                     \
    MDIO_REGS->USERACCESS0 =                                         \
        CSL_FMK(MDIO_USERACCESS0_GO,1u)                             | \
        CSL_FMK(MDIO_USERACCESS0_REGADR,regadr)                     | \
        CSL_FMK(MDIO_USERACCESS0_PHYADR,phyadr)                     \

#define PHYREG_write(regadr, phyadr, data)                             \
    MDIO_REGS->USERACCESS0 =                                         \
        CSL_FMK(MDIO_USERACCESS0_GO,1u)                             | \
        CSL_FMK(MDIO_USERACCESS0_WRITE,1)                           | \
        CSL_FMK(MDIO_USERACCESS0_REGADR,regadr)                     | \
        CSL_FMK(MDIO_USERACCESS0_PHYADR,phyadr)                     | \
        CSL_FMK(MDIO_USERACCESS0_DATA, data)                        \

#define PHYREG_wait()                                                \
    while (CSL_FEXT(MDIO_REGS->USERACCESS0,MDIO_USERACCESS0_GO) )

#define PHYREG_wait Results(results ) {                               \
    while (CSL_FEXT(MDIO_REGS->USERACCESS0,MDIO_USERACCESS0_GO) ); \
    results = CSL_FEXT(MDIO_REGS->USERACCESS0, MDIO_USERACCESS0_DATA); }

```

## 2.8 EMAC Module

Section 2.8 discusses the architecture and basic functions of the EMAC module.

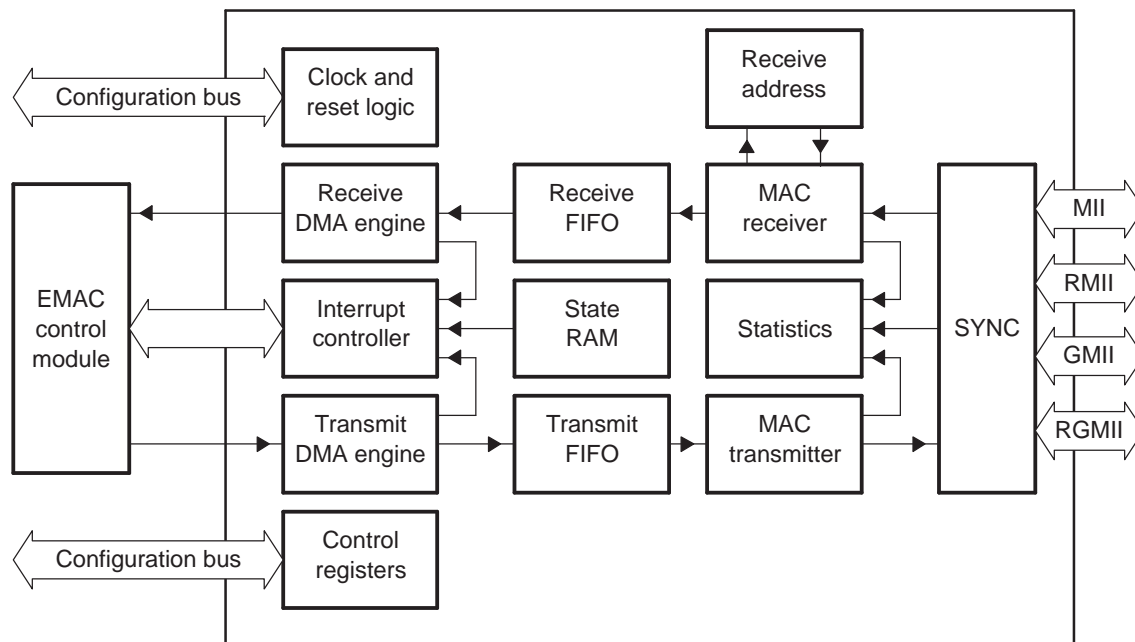
### 2.8.1 EMAC Module Components

The EMAC module (Figure 13) interfaces to PHY components through one of the four Media Independent Interfaces (MII, RMI, GMII, or RGMII), and interfaces to the system core through the EMAC control module.

The EMAC module consists of the following logical components:

- The receive path includes: receive DMA engine, receive FIFO, MAC receiver, and receive address sub-module
- The transmit path includes: transmit DMA engine, transmit FIFO, and MAC transmitter
- Statistics logic
- State RAM
- Interrupt controller
- Control registers and logic
- Clock and reset logic

Figure 13. EMAC Module Block Diagram



#### 2.8.1.1 Receive DMA Engine

The receive DMA engine performs the data transfer between the receive FIFO and the device internal or external memory. It interfaces to the processor through the bus arbiter in the EMAC control module. This DMA engine is totally independent of the C645x DSP EDMA.

#### 2.8.1.2 Receive FIFO

The receive FIFO consists of sixty-eight cells of 64 bytes each and associated control logic. The FIFO buffers receive data in preparation for writing into packet buffers in device memory, and also enable receive FIFO flow control.

### **2.8.1.3 MAC Receiver**

The MAC receiver detects and processes incoming network frames, de-frames them, and places them into the receive FIFO. The MAC receiver also detects errors and passes statistics to the statistics RAM.

### **2.8.1.4 Receive Address**

This sub-module performs address matching and address filtering based on the incoming packet's destination address. It contains a 32 by 53 bit two-port RAM in which up to 32 addresses can be stored to be either matched or filtered by the EMAC.

The RAM may contain multicast packet addresses, but the associated channel must have the unicast enable bit set, even though it is a multicast address. The unicast enable bits are used with multicast addresses in the receive address RAM (not the multicast hash enable bits). Therefore, hash matches can be disabled, but specific multicast addresses can be matched (or filtered) in the RAM. If a multicast packet hash matches, the packet may still be filtered in the RAM. Each packet can be sent to only a single channel.

### **2.8.1.5 Transmit DMA Engine**

The transmit DMA engine performs the data transfer between the device internal or external memory and the transmit FIFO. It interfaces to the processor through the bus arbiter in the EMAC control module. This DMA engine is totally independent of the C645x DSP EDMA.

### **2.8.1.6 Transmit FIFO**

The transmit FIFO consists of twenty-four cells of 64 bytes each and associated control logic. This enables a packet of 1518 bytes (standard Ethernet packet size) to be sent without the possibility of under-run. The FIFO buffers data in preparation for transmission.

### **2.8.1.7 MAC Transmitter**

The MAC transmitter formats frame data from the transmit FIFO and transmits the data using the CSMA/CD access protocol. The frame CRC can be automatically appended, if required. The MAC transmitter also detects transmission errors and passes statistics to the statistics registers.

### **2.8.1.8 Statistics Logic**

The statistics logic RAM counts and stores the Ethernet statistics, keeping track of 36 different Ethernet packet statistics.

### **2.8.1.9 State RAM**

The state RAM contains the head descriptor pointers and completion pointers registers for both transmit and receive channels.

### **2.8.1.10 EMAC Interrupt Controller**

The interrupt controller contains the interrupt related registers and logic. The 18 raw EMAC interrupts are input to this sub-module and masked module interrupts are output.

### **2.8.1.11 Control Registers and Logic**

The EMAC is controlled by a set of memory-mapped registers. The control logic also signals transmit, receive, and status related interrupts to the CPU through the EMAC control module.

### 2.8.1.12 Clock and Reset Logic

The clock and reset sub-module generates all the clocks and resets for the EMAC peripheral.

## 2.8.2 EMAC Module Operational Overview

After reset, initialization, and configuration of the EMAC, the application software running on the host may initiate transmit operations. Transmit operations are initiated by host writes to the appropriate transmit channel head descriptor pointer contained in the state RAM block. The transmit DMA controller then fetches the first packet in the packet chain from memory. The DMA controller writes the packet into the transmit FIFO in bursts of 64-byte cells. The MAC transmitter initiates the packet transmission when either the threshold number of cells (configurable via TXCELLTHRESH in the FIFOCONTROL register) have been written to the transmit FIFO, or a complete packet has been written, whichever is smaller. The SYNC block transmits the packet over one of the MII interfaces in accordance with the 802.3 protocol. The statistics block counts transmit statistics.

Receive operations are initiated by host writes to the appropriate receive channel head descriptor pointer after host initialization and configuration. The SYNC sub-module receives packets and strips off the Ethernet related protocol. The packet data is input to the MAC receiver, which checks for address match (in conjunction with the receive address block) and processes errors. Accepted packets are written to the receive FIFO in bursts of 64-byte cells. The receive DMA controller then writes the packet data to memory. The statistics block counts receive statistics.

The EMAC module operates independently of the CPU. It is configured and controlled by its register set mapped into device memory. Information about data packets is communicated using 16-byte descriptors that are placed in an 8K-byte block of RAM in the EMAC control module.

For transmit operations, each 16-byte descriptor describes a packet or packet fragment in the system's internal or external memory. For receive operations, each 16-byte descriptor represents a free packet buffer or buffer fragment. On both transmit and receive, an Ethernet packet is allowed to span one or more memory fragments, represented by one 16-byte descriptor per fragment. In typical operation, there is only one descriptor per receive buffer, but transmit packets may be fragmented, depending on the software architecture.

An interrupt is issued to the CPU whenever a transmit or receive operation has completed. However, it is not necessary for the CPU to service the interrupt while there are additional resources available. In other words, the EMAC continues to receive Ethernet packets until its receive descriptor list has been exhausted. On transmit operations, the transmit descriptors need only be serviced to recover their associated memory buffer. Thus, it is possible to delay servicing of the EMAC interrupt if there are real time tasks to perform.

Eight channels are supplied for both transmit and receive operations. On transmit, the eight channels represent eight independent transmit queues. The EMAC can be configured to treat these channels as an equal priority round-robin queue, or as a set of eight fixed-priority queues. On receive, the eight channels represent eight independent receive queues with packet classification. Packets are classified based on the destination MAC address. Each of the eight channels is assigned its own MAC address, enabling the EMAC module to act like eight virtual MAC adapters. Also, specific types of frames can be sent to specific channels. For example, multicast, broadcast, or other (promiscuous, error, etc.) frames can each be received on a specific receive channel queue.

The EMAC tracks 36 different statistics, as well as recording the status of each individual packet in its corresponding packet descriptor.

## 2.9 Media Independent Interfaces

The EMAC supports four physical interfaces to external devices: Media Independent Interface (MII), Reduced Media Independent Interface (RMII), Gigabit Media Independent Interface (GMII), and Reduced Gigabit Media Independent Interface (RGMII). The physical interface used depends on the MACSEL pins. The basic operation of all four interfaces is the same, with some minor differences.

The following sections discuss the operation of these interfaces in 10/100 Mbps mode (MII, RMII, GMII and RGMII), and 1000 Mbps mode (GMII and RGMII). An IEEE 802.3 compliant Ethernet MAC controls these interfaces.

### 2.9.1 Data Reception

#### 2.9.1.1 Receive Control

Data received from the PHY is interpreted and output to the EMAC receive FIFO. Interpretation involves detection and removal of the preamble and start of frame delimiter, extraction of the address and frame length, data handling, error checking and reporting, cyclic redundancy checking (CRC), and statistics control signal generation. Receive address detection and frame filtering of the frames that do not address-match is performed outside the Media Independent interface.

#### 2.9.1.2 Receive Inter-Frame Interval

The 802.3 required inter-packet gap (IPG) is 24 receive data clocks (96 bit times). However, the EMAC can tolerate a reduced IPG (2 receive clocks in 10/100 Mbps mode and 5 receive clocks in 1000 Mbps mode) with a correct preamble and start frame delimiter. This interval between frames must comprise (in the following order):

1. An Inter-Packet Gap (IPG).
2. A seven bytes preamble (all bytes 55h).
3. A one byte start of frame delimiter (5Dh).

#### 2.9.1.3 Receive Flow Control

When enabled and triggered, receive flow control is initiated to limit the EMAC from further frame reception. Two forms of receive flow control are implemented on the C645x device:

- Receive buffer flow control
- Receive FIFO flow control

When enabled and triggered, receive buffer flow control prevents further frame reception based on the number of free buffers available. Receive buffer flow control issues flow control collisions in half-duplex mode and IEEE 802.3X pause frames for full-duplex mode.

Receive buffer flow control is triggered when the number of free buffers in any enabled receive channel (RX $n$ FREEBUFFER) is less than or equal to the channel flow control threshold register (RX $n$ FLOWTHRESH) value. Receive flow control is independent of receive QoS, except that both use the free buffer values.

When enabled and triggered, receive FIFO flow control prevents further frame reception based on the number of cells currently in the receive FIFO. Receive FIFO flow control may be enabled only in full-duplex mode (FULLDUPLEX bit is set in the MACCONTROL register). Receive flow control prevents reception of frames on the port until all of the triggering conditions clear, at which time frames may again be received by the port.

Receive FIFO flow control is triggered when the occupancy of the FIFO is greater than or equal to the RXFIFOFLOWTHRESH value in the FIFOCONTROL register. The RXFIFOFLOWTHRESH value must be greater than or equal to 1h and less than or equal to 42h (decimal 66). The RXFIFOFLOWTHRESH reset value is 2h.

Receive flow control is enabled by the RXBUFFERFLOWEN bit and the RXFIFOFLOWEN bit in the MACCONTROL register. The FULLDUPLEX bit in the MACCONTROL register configures the EMAC for collision or IEEE 802.3X flow control.

#### 2.9.1.4 Collision-Based Receive Buffer Flow Control

Collision-based receive buffer flow control provides a means of preventing frame reception when the EMAC is operating in half-duplex mode (FULLDUPLEX bit is cleared in MACCONTROL register). When receive flow control is enabled and triggered, the EMAC generates collisions for received frames. The jam sequence transmitted is the twelve byte sequence C3.C3.C3.C3.C3.C3.C3.C3.C3.C3.C3.C3 in hexadecimal. The jam sequence begins approximately when the source address starts to be received. Note that these forced collisions are not limited to a maximum of 16 consecutive collisions, and are independent of the normal back-off algorithm.

Receive flow control does not depend on the value of the incoming frame destination address. A collision is generated for any incoming packet, regardless of the destination address, if any EMAC enabled channel's free buffer register value is less than or equal to the channel's flow threshold value.

#### 2.9.1.5 IEEE 802.3X Based Receive Buffer Flow Control

IEEE 802.3x based receive buffer flow control provides a means of preventing frame reception when the EMAC is operating in full-duplex mode (the FULLDUPLEX bit is set in the MACCONTROL register). When receive flow control is enabled and triggered, the EMAC transmits a pause frame to request that the sending station stop transmitting for the period indicated within the transmitted pause frame.

The EMAC transmits a pause frame to the reserved multicast address at the first available opportunity (immediately if currently idle, or following the completion of the frame currently being transmitted). The pause frame contains the maximum possible value for the pause time (FFFFh). The EMAC counts the receive pause frame time (decrements FF00h to 0) and retransmits an outgoing pause frame, if the count reaches zero. When the flow control request is removed, the EMAC transmits a pause frame with a zero pause time to cancel the pause request.

Note that transmitted pause frames are only a request to the other end station to stop transmitting. Frames that are received during the pause interval are received normally (provided the receive FIFO is not full).

Pause frames are transmitted if enabled and triggered, regardless of whether or not the EMAC is observing the pause time period from an incoming pause frame.

The EMAC transmits pause frames as described below:

- The 48-bit reserved multicast destination address 01.80.C2.00.00.01h.
- The 48-bit source address (set via the MACSRCADDRLO and MACSRCADDRHI registers).
- The 16-bit length/type field containing the value 88.08h.
- The 16-bit pause opcode equal to 00.01h.
- The 16-bit pause time value of FF.FFh. A pause-quantum is 512 bit-times. Pause frames sent to cancel a pause request have a pause time value of 00.00h.
- Zero padding to 64-byte data length (EMAC transmits only 64-byte pause frames).
- The 32-bit frame-check sequence (CRC word).

All quantities are hexadecimal and are transmitted most-significant-byte first. The least-significant-bit (LSB) is transferred first in each byte.

If the RXBUFFERFLOWEN bit in the MACCONTROL register is cleared while the pause time is nonzero, then the pause time is cleared and a zero count pause frame is sent.

## 2.9.2 Data Transmission

The EMAC passes data to the PHY from the transmit FIFO (when enabled). Data is synchronized to the transmit clock rate. Transmission begins when there are TXCELLTHRESH cells of 64 bytes each, or a complete packet, in the FIFO.

### 2.9.2.1 Transmit Control

A jam sequence is output if a collision is detected on a transmit packet. If the collision was late (after the first 64 bytes have been transmitted), the collision is ignored. If the collision is not late, the controller will back off before retrying the frame transmission. When operating in full-duplex mode, the carrier sense (MCRS) and collision-sensing (MCOL) modes are disabled.

### 2.9.2.2 CRC Insertion

If the SOP buffer descriptor PASSCRC flag is cleared, the EMAC generates and appends a 32-bit Ethernet CRC onto the transmitted data. For the EMAC-generated CRC case, a CRC (or placeholder) at the end of the data is allowed but not required. The buffer byte count value should not include the CRC bytes, if they are present.

If the SOP buffer descriptor PASSCRC flag is set, then the last four bytes of the transmit data are transmitted as the frame CRC. The four CRC data bytes should be the last four bytes of the frame and should be included in the buffer byte count value. The MAC performs no error checking on the outgoing CRC.

### 2.9.2.3 Adaptive Performance Optimization (APO)

The EMAC incorporates adaptive performance optimization (APO) logic that may be enabled by setting the TXPACE bit in the MACCONTROL register. Transmission pacing to enhance performance is enabled when the TXPACE bit is set. Adaptive performance pacing introduces delays into the normal transmission of frames, delaying transmission attempts between stations, and reducing the probability of collisions occurring during heavy traffic (as indicated by frame deferrals and collisions). These actions increase the chance of a successful transmission.

When a frame is deferred, suffers a single collision, multiple collisions, or excessive collisions, the pacing counter is loaded with an initial value of 31. When a frame is transmitted successfully (without experiencing a deferral, single collision, multiple collision, or excessive collision), the pacing counter is decremented by 1 down to 0.

If the pacing counter is zero, this allows a new frame to immediately attempt transmission (after one IPG). If the pacing counter is nonzero, the frame is delayed by a pacing delay of approximately four inter-packet gap delays. APO only affects the IPG preceding the first attempt at transmitting a frame; APO does not affect the back-off algorithm for re-transmitted frames.

### 2.9.2.4 Interpacket-Gap (IPG) Enforcement

The measurement reference for the IPG of 96 bit times is changed depending on frame traffic conditions. If a frame is successfully transmitted without collision and MCRS is de-asserted within approximately 48 bit times of MTXEN being de-asserted, then 96 bit times is measured from MTXEN. If the frame suffered a collision or MCRS is not de-asserted until more than approximately 48 bit times after MTXEN is de-asserted, then 96 bit times (approximately, but not less) is measured from MCRS.

### 2.9.2.5 Back Off

The EMAC implements the 802.3 binary exponential back-off algorithm.



### 2.9.2.6 Transmit Flow Control

When enabled, incoming pause frames are acted upon to prevent the EMAC from transmitting any further frames. Incoming pause frames are only acted upon when the FULLDUPLEX and TXFLOWEN bits in the MACCONTROL register are set. Pause frames are not acted upon in half-duplex mode. Pause frame action is taken if enabled, but normally the frame is filtered and not transferred to memory. MAC control frames are transferred to memory, if the RXCMFEN bit in the RXMBPENABLE register is set. The TXFLOWEN and FULLDUPLEX bits affect whether MAC control frames are acted upon, but they have no effect upon whether MAC control frames are transferred to memory or filtered.

Pause frames are a subset of MAC control frames with an opcode field of 0001h. Incoming pause frames are only acted upon by the EMAC if the following conditions occur:

- The TXFLOWEN bit is set in the MACCONTROL register.
- The frame's length is between 64 bytes and RXMAXLEN bytes inclusive.
- The frame contains no CRC error or align/code errors.

The pause time value from valid frames is extracted from the two bytes following the opcode. The pause time is loaded into the EMAC transmit pause timer and the transmit pause time period begins.

If a valid pause frame is received during the transmit pause time period of a previous transmit pause frame, then either the destination address is not equal to the reserved multicast address or any enabled or disabled unicast address, and the transmit pause timer immediately expires; or the new pause time value is 0, and the transmit pause timer immediately expires. Otherwise, the EMAC transmit pause timer is set immediately to the new pause frame pause time value. (Any remaining pause time from the previous pause frame is discarded.)

If the TXFLOWEN bit in MACCONTROL is cleared, then the pause timer immediately expires.

The EMAC does not start the transmission of a new data frame any sooner than 512-bit times after a pause frame with a non-zero pause time has finished being received (MRXDV going inactive). No transmission begins until the pause timer has expired (the EMAC may transmit pause frames to initiate outgoing flow control). Any frame already in transmission when a pause frame is received is completed and unaffected.

Incoming pause frames consist of:

- A 48-bit destination address equal to one of the following:
  - The reserved multicast destination address 01.80.C2.00.00.01h
  - Any EMAC 48-bit unicast address. Pause frames are accepted, regardless of whether the channel is enabled.
- The 48-bit source address of the transmitting device
- The 16-bit length/type field containing the value 88.08h
- The 16-bit pause opcode equal to 00.01h
- The 16-bit pause time. A pause-quantum is 512 bit-times
- Padding to 64-byte data length
- The 32-bit frame-check sequence (CRC word)

All quantities are hexadecimal and are transmitted most-significant-byte first. The least-significant-bit (LSB) is transferred first in each byte.

The padding is required to make up the frame to a minimum of 64 bytes. The standard allows pause frames longer than 64 bytes to be discarded or interpreted as valid pause frames. The EMAC recognizes any pause frame between 64 bytes and RXMAXLEN bytes in length.

### 2.9.2.7 Speed, Duplex, and Pause Frame Support

The MAC can operate in half-duplex or full-duplex mode at 10 Mbps or 100 Mbps, and can operate in full duplex only in 1000 Mbps. Pause frame support is included in 10/100/1000 Mbps modes as configured by the host.

## 2.10 Packet Receive Operation

### 2.10.1 Receive DMA Host Configuration

To configure the receive DMA for operation, the host must perform the following actions:

- Initialize the receive addresses
- Initialize the RX $n$ HDP registers to zero
- Write the MACHASH1 and MACHASH2 registers, if hash matching multicast addressing is desired
- Initialize the RX $n$ FREEBUFFER, RX $n$ FLOWTHRESH, and RXFILTERLOWTHRESH registers, if flow control is to be enabled
- Enable the desired receive interrupts using the RXINTMASKSET and RXINTMASKCLEAR registers
- Set the appropriate configuration bits in the MACCONTROL register
- Write the RXBUFFEROFFSET register value (typically zero)
- Set up the receive channel(s) buffer descriptors and initialize the RX $n$ HDP registers
- Enable the receive DMA controller by setting the RXEN bit in the RXCONTROL register
- Configure and enable the receive operation, as desired, in the RXMBPENABLE register and by using the RXUNICASTSET and RXUNICASTCLEAR registers

### 2.10.2 Receive Channel Enabling

Each of the eight receive channels has an enable bit (RXCH $n$ EN) in the RXUNICASTSET register that is controlled using the RXUNICASTSET and RXUNICASTCLEAR registers. The RXCH $n$ EN bits determine whether the given channel is enabled (when set to 1) to receive frames with a matching unicast or multicast destination address.

The RXBROADEN bit in the RXMBPENABLE register determines if broadcast frames are enabled or filtered. If broadcast frames are enabled, then they are copied to only a single channel selected by the RXBROADCH field of RXMBPENABLE register.

The RXMULTEN bit in the RXMBPENABLE register determines if hash matching multicast frames are enabled or filtered. Incoming multicast addresses (group addresses) are hashed into an index in the hash table. If the indexed bit is set, the frame hash will match and it will be transferred to the channel selected by the RXMULTCH field when multicast frames are enabled. The multicast hash bits are set in the MACHASH1 and MACHASH2 registers.

The RXPRMCH bits in the RXMBPENABLE register select the promiscuous channel to receive frames selected by the RXCMFEN, RXCSFEN, RXCEFEN, and RXCAFEN bits. These four bits allow reception of MAC control frames, short frames, error frames, and all frames (promiscuous), respectively.

The address RAM can be configured to set multiple unicast and/or multicast addresses to a given channel (if the match bit is set in the RAM). Multicast addresses in the RAM are enabled by the RXUNICASTSET register and not by the RXMULTEN bit in the RXMBPENABLE register. The RXMULTEN bit enables the hash multicast match only. The address RAM takes precedence over the hash match.

If a multicast packet is received that hash matches (multicast packets enabled), but is filtered in the RAM, then the packet is filtered. If a multicast packet does not hash match, regardless of whether or not hash matching is enabled, but matches an enabled multicast address in the RAM, then the packet will be transferred to the associated channel.

### 2.10.3 Receive Channel Addressing

The receive address block can store up to 32 addresses to be filtered or matched. Before enabling packet reception, all the address RAM locations should be initialized, including locations to be unused. The system software is responsible for adding and removing addresses from the RAM.

A MAC address location in RAM is 53 bits wide and consists of:

- 48 bits of the MAC address
- 3 bits for the channel to which a valid address match will be transferred. The channel is a don't care if MATCHFILT bit is cleared.
- A valid bit
- A match or filter bit

First, write the index into the address RAM in the MACINDEX register to start writing a MAC address. Then write the upper 32 bits of the MAC address (MACADDRHI register), and then the lower 16 bits of MAC address with the VALID and MATCHFILT control bits (MACADDRLO). The valid bit should be cleared for the unused locations in the receive address RAM.

The most common uses for the receive address sub-module are:

- Set EMAC in promiscuous mode, using RXCAFEN and RXPROMCH bits in the RXMBPENABLE register. Then filter up to 32 individual addresses, which can be both unicast and/or multicast.
- Disable the promiscuous mode (RXCAFEN = 0) and match up to 32 individual addresses, multicast and/or unicast.

### 2.10.4 Hardware Receive QOS Support

Hardware receive quality of service (QOS) is supported, when enabled, by the Tag Protocol Identifier format and the associated Tag Control Information (TCI) format priority field. When the incoming frame length/type value is equal to 81.00h, the EMAC recognizes the frame as an Ethernet Encoded Tag Protocol Type. The two octets immediately following the protocol type contain the 16-bit TCI field. Bits 15-13 of the TCI field contain the received frames priority (0 to 7). The received frame is a low-priority frame if the priority value is 0 to 3. The received frame is a high-priority frame if the priority value is 4 to 7. All frames that have a length/type field value not equal to 81.00h are low-priority frames.

Received frames that contain priority information are determined by the EMAC as:

- A 48-bit (6 bytes) destination address equal to:
  - The destination station's individual unicast address
  - The destination station's multicast address (MACHASH1 and MACHASH2 registers)
  - The broadcast address of all ones
- A 48-byte (6 bytes) source address
- The 16-bit (2 bytes) length/type field containing the value 81.00h
- The 16-bit (2 bytes) TCI field with the priority field in the upper 3 bits
- Data bytes
- The 4-bytes CRC

The RXFILTERLOWTHRESH and the RX $n$ FREEBUFFER registers are used in conjunction with the priority information to implement receive hardware QOS. Low-priority frames are filtered if the number of free buffers (RX $n$ FREEBUFFER) for the frame channel is less than or equal to the filter low threshold (RXFILTERLOWTHRESH) value. Hardware QOS is enabled by the RXQOSEN bit in the RXMBPENABLE register.

### 2.10.5 Host Free Buffer Tracking

The host must track free buffers for each enabled channel (including unicast, multicast, broadcast, and promiscuous) if receive QOS or receive flow control is used. Disabled channel free buffer values are don't cares. During initialization, the host should write the number of free buffers for each enabled channel to the appropriate  $RX_n$ FREEBUFFER register. The EMAC decrements the appropriate channel's free buffer value for each buffer used. When the host reclaims the frame buffers, the host should write the channel free buffer register with the number of reclaimed buffers (write to increment). There are a maximum of 65 535 free buffers available. The  $RX_n$ FREEBUFFER registers only need to be updated by the host if receive QOS or flow control is used.

### 2.10.6 Receive Channel Teardown

The host commands a receive channel teardown by writing the channel number to the RXTEARDOWN register. When a teardown command is issued to an enabled receive channel, the following occurs:

- Any current frame in reception completes normally.
- The TDOWNCMPLT flag is set in the next buffer descriptor in the chain, if there is one.
- The channel head descriptor pointer is cleared.
- A receive interrupt for the channel is issued to the host.
- The corresponding  $RX_n$ CP register contains the value FFFF FFFCh.
- The host should acknowledge a teardown interrupt with an FFFF FFFCh acknowledge value.

Channel teardown may be commanded on any channel at any time. The host is informed of the teardown completion by the set teardown complete buffer descriptor bit. The EMAC does not clear any channel enables due to a teardown command. A teardown command to an inactive channel issues an interrupt that software should acknowledge with an FFFF FFFCh acknowledge value to  $RX_n$ CP (note that there is no buffer descriptor in this case). Software may read  $RX_n$ CP to determine if the interrupt was due to a commanded teardown. The read value is FFFF FFFCh if the interrupt was due to a teardown command.

### 2.10.7 Receive Frame Classification

Received frames are proper, or good, frames if they are between 64 and RXMAXLEN in length (inclusive) and contain no code, align, or CRC errors.

Received frames are long frames if their frame count exceeds the value in the RXMAXLEN register. The RXMAXLEN register default reset value is 5EEh (1518 in decimal). Long received frames are either oversized or jabber frames. Long frames with no errors are oversized frames; long frames with CRC, code, or alignment errors are jabber frames.

Received frames are short frames if their frame count is less than 64 bytes. Short frames that address match and contain no errors are undersized frames; short frames with CRC, code, or alignment errors are fragment frames. If the frame length is less than or equal to 20, then the frame CRC is passed regardless of whether the RXPASSCRC bit is set or cleared in the RXMBPENABLE register.

A received long packet always contains RXMAXLEN number of bytes transferred to memory (if the RXCEFEN bit is set in RXMBPENABLE) regardless of the value of the RXPASSCRC bit. Following is an example with RXMAXLEN set to 1518:

- If the frame length is 1518, then the packet is not a long packet and there are 1514 or 1518 bytes transferred to memory depending on the value of the RXPASSCRC bit.
- If the frame length is 1519, there are 1518 bytes transferred to memory regardless of the RXPASSCRC bit value. The last three bytes are the first three CRC bytes.
- If the frame length is 1520, there are 1518 bytes transferred to memory regardless of the RXPASSCRC bit value. The last two bytes are the first two CRC bytes.
- If the frame length is 1521, there are 1518 bytes transferred to memory regardless of the RXPASSCRC bit value. The last byte is the first CRC byte.
- If the frame length is 1522, there are 1518 bytes transferred to memory. The last byte is the last data byte.

## 2.10.8 Promiscuous Receive Mode

When the promiscuous receive mode is enabled by setting the RXCAFEN bit in the RXMBPENABLE register, non-address matching frames that would normally be filtered are transferred to the promiscuous channel. Address matching frames that would normally be filtered due to errors are transferred to the address match channel when RXCAFEN and RXCEFEN bits are set. Address matching frames with the filter bit set (MATCHFILT = 0) are always filtered regardless of the RXCAFEN and RXCEFEN bit setting. A frame is considered to be an address matching frame only if it is enabled to be received on a unicast, multicast, or broadcast channel. Frames received to disabled unicast, multicast, or broadcast channels are considered non-address matching.

MAC control frames address match only if RXCMFEN bit is set. RXCEFEN and RXCSFEN determine whether error frames are transferred to memory or not, but they do not determine whether error frames are address matching or not. Short frames are a special type of error frames.

A single channel is selected as the promiscuous channel by the RXPROMCH field in the RXMBPENABLE register. The promiscuous receive mode is enabled by the RXCMFEN, RXCEFEN, RXCSFEN, and RXCAFEN bits in RXMBPENABLE. [Table 8](#) shows the effects of the promiscuous enable bits. Proper frames are frames that are between 64 and RXMAXLEN bytes in length inclusive and contain no code, align, or CRC errors.

**Table 8. Receive Frame Treatment Summary**

ADDRESS MATCH	RXMBPENABLE Bits				Frame treatment
	RXCAFEN	RXCEFEN	RXCMFEN	RXCSFEN	
0	0	X	X	X	No frames transferred.
0	1	0	0	0	Proper frames transferred to promiscuous channel.
0	1	0	0	1	Proper/undersized data frames transferred to promiscuous channel.
0	1	0	1	0	Proper data and control frames transferred to promiscuous channel.
0	1	0	1	1	Proper/undersized data and control frames transferred to promiscuous channel.
0	1	1	0	0	Proper/oversize/jabber/code/align/CRC data frames transferred to promiscuous channel. No control or undersized/fragment frames are transferred.
0	1	1	0	1	Proper/undersized/fragment/oversize/jabber/code/align/CRC data frames transferred to promiscuous channel. No control frames are transferred.
0	1	1	1	0	Proper/oversize/jabber/code/align/CRC data and control frames transferred to promiscuous channel. No undersized frames are transferred.
0	1	1	1	1	All non-address matching frames with and without errors transferred to promiscuous channel.
1	X	0	0	0	Proper data frames transferred to address match channel.
1	X	0	0	1	Proper/undersized data frames transferred to address match channel.
1	X	0	1	0	Proper data and control frames transferred to address match channel.
1	X	0	1	1	Proper/undersized data and control frames transferred to address match channel.
1	X	1	0	0	Proper/oversize/jabber/code/align/CRC data frames transferred to address match channel. No control or undersized frames are transferred.

**Table 8. Receive Frame Treatment Summary (continued)**

ADDRESS MATCH	RXMBPENABLE Bits				Frame treatment
	RXCAFEN	RXCEFEN	RXCMFEN	RXCSEFEN	
1	X	1	0	1	Proper/oversize/jabber/fragment/undersized/code/align/CRC data frames transferred to address match channel. No control frames are transferred.
1	X	1	1	0	Proper/oversize/jabber/code/align/CRC data and control frames transferred to address match channel. No undersized/fragment frames are transferred.
1	X	1	1	1	All address matching frames with and without errors transferred to the address match channel.

### 2.10.9 Receive Overrun

The types of receive overrun are:

- FIFO start of frame overrun (FIFO\_SOF)
- FIFO middle of frame overrun (FIFO\_MOF)
- DMA start of frame overrun (DMA\_SOF)
- DMA middle of frame overrun (DMA\_MOF)

The statistics counters used to track these types of receive overrun are:

- Receive Start of Frame Overruns Register (RXSOFOVERRUNS)
- Receive Middle of Frame Overruns Register (RXMOFOVERRUNS)
- Receive DMA Overruns Register (RXDMAOVERRUNS)

Start of frame overruns happen when there are no resources available when frame reception begins. Start of frame overruns increment the appropriate overrun statistic(s) and the frame is filtered.

Middle of frame overruns happen when there are some resources to start the frame reception, but the resources run out during frame reception. In normal operation, a frame that overruns after starting the frame reception is filtered and the appropriate statistic(s) are incremented; however, the RXCEFEN bit in the RXMBPENABLE register affects overrun frame treatment. [Table 9](#) shows how the overrun condition is handled for the middle of frame overrun.

**Table 9. Middle of Frame Overrun Treatment**

ADDRESS MATCH	RXCAFEN	RXCEFEN	Middle of frame overrun treatment
0	0	X	Overrun frame filtered.
0	1	0	Overrun frame filtered.
0	1	1	As much frame data as possible is transferred to the promiscuous channel until overrun. The appropriate overrun statistic(s) is incremented and the OVERRUN and NOMATCH flags are set in the SOP buffer descriptor. Note that the RXMAXLEN number of bytes cannot be reached for an overrun to occur (it would be truncated and be a jabber or oversize).
1	X	0	Overrun frame filtered with the appropriate overrun statistic(s) incremented.
1	X	1	As much frame data as possible is transferred to the address match channel until overrun. The appropriate overrun statistic(s) is incremented and the OVERRUN flag is set in the SOP buffer descriptor. Note that the RXMAXLEN number of bytes cannot be reached for an overrun to occur (it would be truncated).

## 2.11 Packet Transmit Operation

The transmit DMA is an eight channel interface. Priority between the eight queues may be either fixed or round robin as selected by the TXPTYPE bit in the MACCONTROL register. If the priority type is fixed, then channel 7 has the highest priority and channel 0 has the lowest priority. Round robin priority proceeds from channel 0 to channel 7.

### 2.11.1 Transmit DMA Host Configuration

To configure the transmit DMA for operation, the host must perform the following:

- Write the MACSRCADDRLO and MACSRCADDRHI registers (used for pause frames on transmit).
- Initialize the TXnHDP registers to zero.
- Enable the desired transmit interrupts using the TXINTMASKSET and TXINTMASKCLEAR registers.
- Set the appropriate configuration bits in the MACCONTROL register.
- Set up the transmit channel(s) buffer descriptors in host memory.
- Enable the transmit DMA controller by setting the TXEN bit in the TXCONTROL register.
- Write the appropriate TXnHDP registers with the pointer to the first descriptor to start transmit operations.

### 2.11.2 Transmit Channel Teardown

The host commands a transmit channel teardown by writing the channel number to the TXTEARDOWN register. When a teardown command is issued to an enabled transmit channel, the following occurs:

- Any frame currently in transmission completes normally.
- The TDOWNCMPLT flag is set in the next SOP buffer descriptor in the chain, if there is one.
- The channel head descriptor pointer is cleared.
- A transmit interrupt is issued, informing the host of the channel teardown.
- The corresponding TXnCP register contains the value FFFF FFFCh.
- The host should acknowledge a teardown interrupt with an FFFF FFFCh acknowledge value.

Channel teardown may be commanded on any channel at any time. The host is informed of the teardown completion by the set teardown complete buffer descriptor bit (TDOWNCMPLT). The EMAC does not clear any channel enables due to a teardown command. A teardown command to an inactive channel issues an interrupt that software should acknowledge with an FFFF FFFCh acknowledge value to TXnCP (note that there is no buffer descriptor). Software may read the interrupt acknowledge location (TXnCP) to determine if the interrupt was due to a commanded teardown. The read value is FFFF FFFCh if the interrupt was due to a teardown command.

## 2.12 Receive and Transmit Latency

The transmit FIFO contains twenty four 64-byte cells, and the receive FIFO contains sixty eight 64-byte cells. The EMAC begins transmission of a packet on the wire after TXCELLTHRESH cells (configurable through the FIFOCONTROL register) or a complete packet are available in the FIFO.

Transmit under-run cannot occur for packet sizes of TXCELLTHRESH times 64 bytes (or less). For larger packet sizes, transmit under-run can occur if the memory latency is greater than the time required to transmit a 64-byte cell on the wire; this is 0.512  $\mu$ s in 1 Gbit mode, 5.12  $\mu$ s in 100 Mbps mode, and 51.2  $\mu$ s in 10 Mbps mode. The memory latency time includes all buffer descriptor reads for the entire cell data.

The EMAC transmit FIFO uses 24 cells; thus, under-run cannot happen for a normal size packet (less than 1536 packet bytes). Cell transmission can be configured to start only after an entire packet is contained in the FIFO; for a maximum-size packet, set the TXCELLTHRESH field to the maximum possible value of 24.

Receive overrun is prevented if the receive memory cell latency is less than the time required to transmit a 64-byte cell on the wire (0.512  $\mu$ s in 1 Gbps mode, 5.12  $\mu$ s in 100 Mbps mode, or 51.2 $\mu$ s in 10 Mbps mode). The latency time includes any required buffer descriptor reads for the cell data.

Latency to system's internal and external RAM can be controlled through the use of the transfer node priority allocation register in the C645x devices. Latency to descriptor RAM is low because RAM is local to the EMAC, as it is part of the EMAC control module.

## 2.13 Transfer Node Priority

The C645x devices contain a system level priority allocation register (PRI\_ALLOC) that sets the priority of the transfer node used in issuing memory transfer requests to system memory.

Although the EMAC has internal FIFOs to help alleviate memory transfer arbitration problems, the average transfer rate of data read and written by the EMAC to internal or external DSP memory must be at least equal to the Ethernet wire rate. In addition, the internal FIFO system can not withstand a single memory latency event greater than the time it takes to fill or empty a TXCELLTHRESH number of internal 64-byte FIFO cells.

For example, for 1000 Mbps operation, these restrictions translate into the following rules:

- For the short-term average, each 64-byte memory read/write request from the EMAC must be serviced in no more than 0.512  $\mu$ s.
- Any single latency event in request servicing can be no longer than  $(0.512 * TXCELLTHRESH) \mu$ s.

Bits [0-2] of the PRI\_ALLOC register set the transfer node priority for all the master peripherals in the device, including EMAC. A value of 000b will have the highest priority, while 111b will have the lowest. The default priority assigned to EMAC is 001b. It is important to have a balance between all peripherals. In most cases, the default priorities will not need adjustment.

## 2.14 Reset Considerations

### 2.14.1 Software Reset Considerations

For information on the chip level reset capabilities of various peripherals, see the device-specific data manual.

Within the peripheral itself, the EMAC component of the Ethernet MAC peripheral can be placed in a reset state by writing to the SOFTRESET register located in EMAC memory map. Writing a one to bit 0 of this register causes the EMAC logic to be reset, and the register values to be set to their default values. Software reset occurs when the receive and transmit DMA controllers are in an idle state to avoid locking up the configuration bus; it is the responsibility of the software to verify that there are no pending frames to be transferred. After writing a one to this bit, it may be polled to determine if the reset has occurred. A value of one indicates that the reset has not yet occurred. A value of zero indicates that a reset has occurred.

After a software reset operation, all the EMAC registers need to be re-initialized for proper data transmission.

Unlike the EMAC module, the MDIO and EMAC control modules cannot be placed in reset from a register inside their memory map.

### 2.14.2 Hardware Reset Considerations

When a hardware reset occurs, the EMAC peripheral will have its register values reset, and all the sub-modules will return to their default state. After the hardware reset, the EMAC needs to be initialized before resuming its data transmission, as described in [Section 2.15](#).

A hardware reset is the only means of recovering from the error interrupts (HOSTPEND), which are triggered by errors in packet buffer descriptors. Before doing a hardware reset, you should inspect the error codes in the MACSTATUS register. This register provides information about the software error type that needs correction. For more information on error interrupts, see [Section 2.16.1.4](#).



---

## 2.15 Initialization

### 2.15.1 Enabling the EMAC/MDIO Peripheral

When the device is powered on, the EMAC peripheral is disabled. Prior to EMAC-specific initialization, the EMAC must be enabled; otherwise its registers cannot be written, and the reads will all return a value of zero. The interface to be used (MII, RMII, GMII, or RGMII) is automatically selected at power-on reset, based on the state of the MACSEL configuration pins.

EMAC/MDIO is enabled through the chip level module state control register 0 (MDCTL0) and module status register 0 (MDSTAT0). For detailed information on the programming sequence, see the device-specific data manual. This sequence will enable the EMAC peripheral, and the register values are reset to default. Module-specific initialization may proceed.

### 2.15.2 EMAC Control Module Initialization

The EMAC control module is used for global interrupt enable, and to pace back-to-back interrupts using an interrupt re-trigger count based on the peripheral clock (CPUclk/6). There is also an 8K block of RAM local to the EMAC that holds packet buffer descriptors.

Note that although the EMAC control module and the EMAC module have slightly different functions, in practice, the type of maintenance performed on the EMAC control module is more commonly conducted from the EMAC module software (as opposed to the MDIO module).

The initialization of the EMAC control module consists of two parts:

1. Configuration of the interrupt on the DSP.
2. Initialization of the EMAC control module:
  - Setting the interrupt pace count (using EWINTTCNT)
  - Initializing the EMAC and MDIO modules
  - Enabling interrupts in the EMAC control module (using EWCTL)

See [Example 4](#) to view example code used to perform the actions associated with the second part of the EMAC control module initialization when using the register-level CSL.

Use the system's interrupt controller to map the EMAC interrupts to one of the CPU's interrupts. Once the interrupt is mapped to a CPU interrupt, general masking and unmasking of the interrupt (to control reentrancy) should be done at the chip level by manipulating the interrupt enable mask. The EMAC control module control register (EWCTL) should only enable and disable interrupts from within the EMAC interrupt service routine (ISR), as disabling and re-enabling the interrupt in EWCTL also resets the interrupt pace counter.

#### Example 4. EMAC Control Module Initialization Code

```

Uint32 tmpval;

/*
// Globally disable EMAC/MDIO interrupts in the control module
*/

CSL_FINST(ECTL_REGS->EWCTL, ECTL_EWCTL_INTEN, DISABLE );

/* Wait about 100 cycles */
for( I=0; i<5; I++ )
    tmpval = ECTL_REGS->EWCTL;

/* Set Interrupt Timer Count (CPUclk/6) */
ECTL_REGS->EWINTCNT = 1500 ;

/*
// Initialize MDIO and EMAC Module
*/

[Discussed Later in this document]

/* Enable global interrupt in the control module */
CSL_FINST(ECTL_REGS->EWCTL, ECTL_EWCTL_INTEN, ENABLE );

```

### 2.15.3 MDIO Module Initialization

The MDIO module initially configures and monitors one or more external PHY devices. Other than initializing the software state machine (details on the MDIO state machine can be found in the IEEE 802.3 standard), the MDIO module only needs the MDIO engine enabled and the clock divider configured. To set the clock divider, supply an MDIO clock of 1 MHz. As the peripheral clock is used as the base clock (CPUclk/6), the divider can be set to 125 for a 750 MHz device. Slower MDIO clocks for slower CPU frequencies are acceptable.

Both the state machine enable and the MDIO clock divider are controlled through the MDIO control register (CONTROL). If none of the potentially connected PHYs require the access preamble, the PREAMBLE bit can also be set in CONTROL to speed up PHY register access. See [Example 5](#) for an example of the code for initialization.

#### Example 5. MDIO Module Initialization Code

```

#define PCLK 125
...
/* Enable MDIO and setup divider */
MDIO_REGS->CONTROL = CSL_FMKT( MDIO_CONTROL_ENABLE, YES) |
                    CSL_FMK( MDIO_CONTROL_CLKDIV, PCLK ) ;

```

If the MDIO module must operate on an interrupt basis, the interrupts can be enabled at this time using the USERINTMASKSET register for register access and the USERPHYSEL<sub>n</sub> register if a target PHY is already known.

Once the MDIO state machine has been initialized and enabled, it starts polling all 32 PHY addresses on the MDIO bus, looking for active PHYs. Since it can take up to 50 μs to read one register, the MDIO module provides an accurate representation of all the PHYs available after a reasonable interval. Also, a PHY can take up to 3 seconds to negotiate a link. Thus, it is advisable to run the MDIO software off a time-based event rather than polling.

For more information on PHY control registers, see your PHY device documentation.

#### 2.15.4 EMAC Module Initialization

The EMAC module sends and receives data packets over the network by maintaining up to 8 transmit and receive descriptor queues. The EMAC module configuration must also be kept current based on the PHY negotiation results returned from the MDIO module. Programming this module is the most time-consuming aspect of developing an application or device driver for Ethernet.

A device driver should follow this initialization procedure to get the EMAC to the state where it is ready to receive and send Ethernet packets. Some of these steps are not necessary when performed immediately after device reset.

1. If enabled, clear the device interrupt enable in EWCTL.
2. Clear the MACCONTROL, RXCONTROL, and TXCONTROL registers (not necessary immediately after reset).
3. Initialize all 16 Head Descriptor Pointer registers (RXnHDP and TXnHDP) to 0.
4. Clear all 36 statistics registers by writing 0 (not necessary immediately after reset).
5. Initialize all 32 receive address RAM locations to 0. Set up the addresses to be matched to the eight receive channels and the addresses to be filtered, through programming the MACINDEX, MACADDRHI, and MACADDRLO registers.
6. Initialize the RXnFREEBUFFER, RXnFLOWTHRESH, and RXFILTERLOWTHRESH registers, if buffer flow control is to be enabled. Program the FIFOCONTROL register if FIFO flow control is desired.
7. Most device drivers open with no multicast addresses, so clear the MACHASH1 and MACHASH2 registers.
8. Write the RXBUFFEROFFSET register value (typically zero).
9. Initially clear all unicast channels by writing FFh to the RXUNICASTCLEAR register. If unicast is desired, it can be enabled now by writing the RXUNICASTSET register. Some drivers will default to unicast on device open while others will not.
10. If you desire to transfer jumbo frames, set the RXMAXLEN register to the maximum frame length you want to allow to be received.
11. Set up the RXMBPENABLE register with an initial configuration. The configuration is based on the current receive filter settings of the device driver. Some drivers may enable things like broadcast and multicast packets immediately, while others may not.
12. Set the appropriate configuration bits in the MACCONTROL register (do not set the GMIEN bit yet).
13. Clear all unused channel interrupt bits by writing RXINTMASKCLEAR and TXINTMASKCLEAR.
14. Enable the receive and transmit channel interrupt bits in RXINTMASKSET and TXINTMASKSET for the channels to be used, and enable the HOSTMASK and STATMASK bits using the MACINTMASKSET register.
15. Initialize the receive and transmit descriptor list queues using the 8K descriptor memory block contained in the EMAC control module.
16. Prepare receive by writing a pointer to the head of the receive buffer descriptor list to RXnHDP.
17. Enable the receive and transmit DMA controllers by setting the RXEN bit in the RXCONTROL register and the TXEN bit in the TXCONTROL register. Then set the GMIEN bit in MACCONTROL.
18. If the gigabit mode is desired (available only if using GMII or RGMII interface), set the GIG bit in the MACCONTROL register.
19. When using RMII, release the interface logic from reset by clearing the RMII\_RST field of the EMAC Configuration register (EMACCFG), found at device level.
20. Enable the device interrupt in EWCTL.

## 2.16 Interrupt Support

### 2.16.1 EMAC Module Interrupt Events and Requests

The EMAC/MDIO generates 18 interrupt events, as follows:

- TXPEND $n$ : Transmit packet completion interrupt for transmit channels 7 through 0
- RXPEND $n$ : Receive packet completion interrupt for receive channels 7 through 0
- STATPEND: Statistics interrupt
- HOSTPEND: Host error interrupt

#### 2.16.1.1 Transmit Packet Completion Interrupts

The transmit DMA engine has eight channels, and each channel has a corresponding interrupt (TXPEND $n$ ). The transmit interrupts are level interrupts that remain asserted until cleared by the CPU.

Each of the eight transmit channel interrupts may be individually enabled by setting the appropriate bit in the TXINTMASKSET register. Each of the eight transmit channel interrupts may be individually disabled by clearing the appropriate bit in the TXINTMASKCLEAR register. The raw and masked transmit interrupt status may be read by reading the TXINTSTATRAW and TXINTSTATMASKED registers, respectively.

When the EMAC completes the transmission of a packet, the EMAC issues an interrupt to the CPU by writing the packet's last buffer descriptor address to the appropriate channel queue's TX completion pointer located in the state RAM block. The write generates the interrupt when enabled by the interrupt mask, regardless of the value written.

Upon interrupt reception, the CPU processes one or more packets from the buffer chain and then acknowledges an interrupt by writing the address of the last buffer descriptor processed to the queue's associated TX completion pointer in the transmit DMA state RAM.

The data written by the host (buffer descriptor address of the last processed buffer) is compared to the data in the register written by the EMAC port (address of last buffer descriptor used by the EMAC). If the two values are not equal, indicating that the EMAC has transmitted more packets than the CPU has processed interrupts for, then the transmit packet completion interrupt signal remains asserted. If the two values are equal, indicating that the host has processed all packets that the EMAC has transferred, then the pending interrupt is cleared. Reading the TX $n$ CP register displays the value that the EMAC is expecting.

The EMAC write to the completion pointer stores the value in the state RAM. The CPU written value does not change the register value. The host-written value is compared to the register content, which was written by the EMAC. If the two values are equal, then the interrupt is removed; otherwise the interrupt remains asserted. The host may process multiple packets prior to acknowledging an interrupt, or the host may acknowledge interrupts for every packet.

### 2.16.1.2 Receive Packet Completion Interrupts

The receive DMA engine has eight channels, and each channel has a corresponding interrupt (RXPEND $n$ ). The receive interrupts are level interrupts that remain asserted until cleared by the CPU.

Each of the eight receive channel interrupts may be individually enabled by setting the appropriate bit in the RXINTMASKSET register. Each of the eight receive channel interrupts may be individually disabled by clearing the appropriate bit in the RXINTMASKCLEAR register. The raw and masked receive interrupt status may be read by reading the RXINTSTATRAW and RXINTSTATMASKED registers, respectively.

When the EMAC completes a packet reception, the EMAC issues an interrupt to the CPU by writing the packet's last buffer descriptor address to the appropriate channel queue's RX completion pointer located in the state RAM block. The write generates the interrupt when enabled by the interrupt mask, regardless of the value written.

Upon interrupt reception, the CPU processes one or more packets from the buffer chain and then acknowledges one or more interrupt(s) by writing the address of the last buffer descriptor processed to the queue's associated RX completion pointer in the receive DMA state RAM.

The data written by the host (buffer descriptor address of the last processed buffer) is compared to the data in the register written by the EMAC (address of last buffer descriptor used by the EMAC). If the two values are not equal, indicating that the EMAC has received more packets than the CPU has processed interrupts for, the receive packet completion interrupt signal remains asserted. If the two values are equal, indicating that the host has processed all packets that the EMAC has received, the pending interrupt is de-asserted. Reading the RX $n$ CP register displays the value that the EMAC is expecting .

The EMAC write to the completion pointer stores the value in the state RAM. The CPU written value does not change the register value. The host-written value is compared to the register content, which was written by the EMAC. If the two values are equal, then the interrupt is removed; otherwise the interrupt remains asserted. The host may process multiple packets prior to acknowledging an interrupt, or the host may acknowledge interrupts for every packet.

### 2.16.1.3 Statistics Interrupt

The statistics level interrupt (STATPEND) is issued when any statistics value is greater than or equal to 8000 0000h, if it has been enabled by the STATMASK bit in the MACINTMASKSET register. The statistics interrupt is removed by writing to decrement any statistics value greater than 8000 0000h. The interrupt remains asserted as long as the most-significant-bit of any statistics value is set.

### 2.16.1.4 Host Error Interrupt

The host error interrupt (HOSTPEND) is issued, if enabled, under error conditions due to the handling of buffer descriptors detected during transmit or receive DMA transactions. The failure of the software application to supply properly formatted buffer descriptors results in this error. The error bit can only be cleared by resetting the EMAC module in hardware.

The host error interrupt is enabled by setting the HOSTMASK bit in the MACINTMASKSET register. The host error interrupt is disabled by clearing the appropriate bit in the MACINTMASKCLEAR register. The raw and masked host error interrupt status may be read by reading the MACINTSTATRAW and MACINTSTATMASKED registers, respectively.

Transmit host error conditions include:

- SOP error
- Ownership bit not set in SOP buffer
- Zero next buffer descriptor pointer without EOP
- Zero buffer pointer
- Zero buffer length
- Packet length error

Receive host error conditions include:

- Ownership bit not set in input buffer
- Zero buffer pointer

### 2.16.2 MDIO Module Interrupt Events and Requests

The MDIO module generates two interrupt events, as follows:

- LINKINT: Serial interface link change interrupt. Indicates a change in the state of the PHY link.
- USERINT: Serial interface user command event complete interrupt.

#### 2.16.2.1 Link Change Interrupt

The MDIO module asserts a link change interrupt (LINKINT) if there is a change in the link state of the PHY corresponding to the address in the PHYADRMON bits in the USERPHYSEL $n$  register, and if the LINKINTENB bit is also set in USERPHYSEL $n$ . This interrupt event is also captured in the LINKINTRAW bits of the LINKINTRAW register. The LINKINTRAW bits 0 and 1 correspond to USERPHYSEL0 and USERPHYSEL1, respectively.

When the interrupt is enabled and generated, the corresponding bit is also set in the LINKINTMASKED register. The interrupt is cleared by writing back the same bit to LINKINTMASKED (write to clear).

#### 2.16.2.2 User Access Completion Interrupt

A user access completion interrupt (USERINT) is asserted when the GO bit in one of the USERACCESS $n$  registers transitions from 1 to 0 (indicating completion of a user access) and the bit in the USERINTMASKSET register corresponding to USERACCESS0 or USERACCESS1 is set. This interrupt event is also captured in bits 0 and 1 of the USERINTRAW register. USERINTRAW bits 0 and bit 1 correspond to USERACCESS0 and USERACCESS1, respectively.

When the interrupt is enabled and generated, the corresponding USERINTMASKED bit is also set in the USERINTMASKED register. The interrupt is cleared by writing back the same bit to USERINTMASKED (write to clear).

### 2.16.3 Proper Interrupt Processing

All the interrupts signaled from the EMAC and MDIO modules are level-driven. If they remain active, their level remains constant. However, the CPU core requires edge-triggered interrupts. To properly convert the level-driven interrupt signal to an edge-triggered signal, the application software must use the interrupt control logic of the EMAC control module.

[Section 2.6.3](#) discusses interrupt control in the EMAC control module. For safe interrupt processing, the software application should disable interrupts using the EMAC Control Module Interrupt Control (EWCTL) register upon entry to the ISR, and re-enable them upon leaving the ISR. If any interrupt signals are active at that time, this creates another rising edge on the interrupt signal routed to the CPU interrupt controller, thus triggering another interrupt. The EMAC control module also uses the EMAC Control Module Interrupt Timer Count (EWINTCNT) register to implement interrupt pacing.

### 2.16.4 Interrupt Multiplexing

The EMAC control module combines different interrupt signals from both the EMAC and MDIO modules and generates a single interrupt signal that is wired to the CPU interrupt controller.

Once this interrupt is generated, the reason for the interrupt can be read from the MACINVECTOR register located in the EMAC memory map. MACINVECTOR combines the status of the following 20 interrupt signals: TXPEND<sub>n</sub>, RXPEND<sub>n</sub>, STATPEND, HOSTPEND, LINKINT, and USERINT.

The EMAC and MDIO interrupts are combined within the EMAC control module and mapped to system event 17 through the use of the enhanced interrupt selector within the C64x+ core. For more details, see the Interrupt Controller chapter in the *TMS320C64x+ Megamodule Peripherals Reference Guide SPRU871*.

## 2.17 Power Management

The power saver module integrated in this device allows the clock going to different peripherals to be shut down when that peripheral is not being used. For more information on the power conservation modes available for the EMAC/MDIO peripheral, see the device-specific data manual.

## 2.18 Emulation Considerations

EMAC emulation control is implemented for compatibility with other peripherals. The SOFT and FREE bits from the EMCONTROL register allow EMAC operation to be suspended.

When the emulation suspend state is entered, the EMAC will stop processing receive and transmit frames at the next frame boundary. Any frame currently in reception or transmission will be completed normally without suspension. For transmission, any complete or partial frame in the transmit cell FIFO will be transmitted. For receive, frames that are detected by the EMAC after the suspend state is entered are ignored. No statistics will be kept for ignored frames.

[Table 10](#) shows how the SOFT and FREE bits affect the operation of the emulation suspend.

**Table 10. Emulation Control**

SOFT	FREE	Description
0	0	Normal operation
1	0	Emulation suspend
X	1	Normal operation

### 3 EMAC Control Module Registers

#### 3.1 Introduction

Table 11 lists the memory-mapped registers for the EMAC Control Module. See the device-specific data manual for the memory address of these registers.

**Table 11. EMAC Control Module Registers**

Offset	Acronym	Register Description	Section
4h	EWCTL	EMAC Control Module Interrupt Control Register	<a href="#">Section 3.2</a>
8h	EWINTCNT	EMAC Control Module Interrupt Timer Count Register	<a href="#">Section 3.3</a>

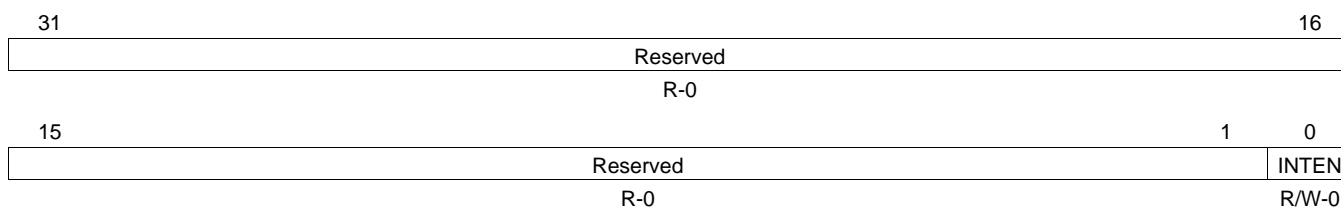
#### 3.2 EMAC Control Module Interrupt Control Register (EWCTL)

The EMAC control module interrupt control register (EWCTL) is used to enable and disable the central interrupt from the EMAC and MDIO modules.

It is expected that any time the EMAC and MDIO interrupt is being serviced, the software disables the INTEN bit in EWCTL. This ensures that the interrupt line goes back to zero. The software reenables the INTEN bit after clearing all the pending interrupts and before leaving the interrupt service routine. At this point, if the EMAC control module monitors any interrupts still pending, it reasserts the interrupt line, and generates a new edge that the DSP can recognize.

The EMAC control module interrupt control register (EWCTL) is shown in [Figure 14](#) and described in [Table 12](#).

**Figure 14. EMAC Control Module Interrupt Control Register (EWCTL)**



LEGEND: R = Read only; R/W = Read/Write; -n = value after reset

**Table 12. EMAC Control Module Interrupt Control Register (EWCTL) Field Descriptions**

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	INTEN	0	Controls the EMAC_MDIO_INT interrupt generation to the CPU. Any time the INTEN bit in the EWCTL register is 0, the EMAC_MDIO_INT signal to the CPU is kept de-asserted. If the INTEN bit in the EWCTL register is 1, then the interrupt control logic checks all the interrupt lines from EMAC and MDIO. If any of these interrupt lines are active, the EMAC_MDIO_INT signal is asserted. Assertion of this signal generates an edge, which can then be recognized as a valid interrupt by the CPU. The INTEN bit takes care of two problems associated with level interrupts from the EMAC and the MDIO modules. One, it makes sure that none of interrupts are missed, and second, it makes sure that only the required number of interrupts are sent to the CPU.
		0	EMAC and MDIO interrupts are disabled
		1	EMAC and MDIO interrupts are enabled

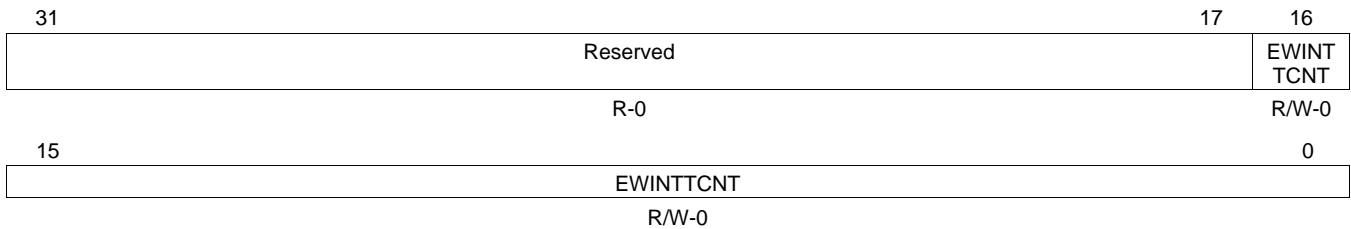


### 3.3 EMAC Control Module Interrupt Timer Count Register (EWINTTCNT)

The EMAC control module interrupt timer count register (EWINTTCNT) is used to control the generation of back-to-back interrupts from the EMAC and MDIO modules. The value of this timer count is loaded into an internal counter every time interrupts are enabled using EWCTL. A second interrupt cannot be generated until this count reaches 0. The counter is decremented at a frequency of CPU clock/6; its default reset count is 0 (inactive), its maximum value is 1 FFFFh (131 071).

The EMAC control module interrupt timer count register (EWINTTCNT) is shown in [Figure 15](#) and described in [Table 13](#).

**Figure 15. EMAC Control Module Interrupt Timer Count Register (EWINTTCNT)**



LEGEND: R = Read only; R/W = Read/Write; -n = value after reset

**Table 13. EMAC Control Module Interrupt Timer Count Register (EWINTTCNT) Field Descriptions**

Bit	Field	Value	Description
31-17	Reserved	0	Reserved
16-0	EWINTTCNT		Interrupt timer count. EWINTTCNT is a 17-bit interrupt timer count that is used to control the generation of back-to-back interrupts from the EMAC and MDIO modules. The value of EWINTTCNT is loaded in an internal time counter every time interrupts are enabled using EWCTL register by writing a '1' to INTEN bit. (Note the INTEN bit must transition from '0' to '1' to initialize the internal time counter.) Once initialized, the time counter will count down with each peripheral clock till it reaches zero. A second interrupt cannot be generated until this counter reaches 0. Any time the time counter has a non-zero value, the interrupt logic will block the EMAC_MDIO_INT interrupt to the CPU. Thus, if any of the interrupts coming to the EMAC control module is asserted, the interrupt logic will assert the EMAC_MDIO_INT signal to the CPU, provided the INTEN bit in the EWCTL register is set, and the time counter value is zero.

## 4 MDIO Registers

### 4.1 Introduction

Table 14 lists the memory-mapped registers for the Management Data Input/Output (MDIO). See the device-specific data manual for the memory address of these registers.

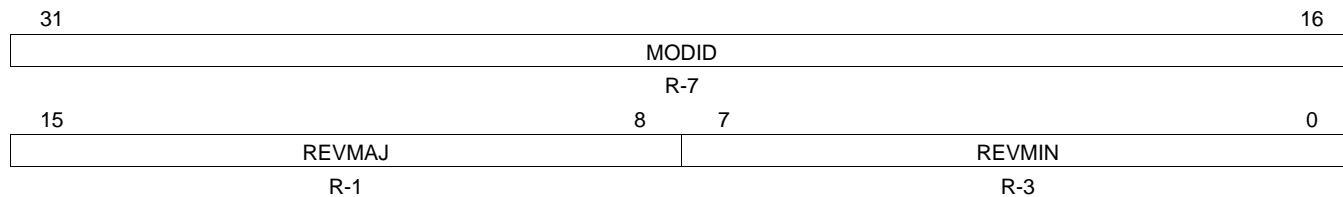
**Table 14. Management Data Input/Output (MDIO) Registers**

Offset	Acronym	Register Description	Section
0h	VERSION	MDIO Version Register	<a href="#">Section 4.2</a>
4h	CONTROL	MDIO Control Register	<a href="#">Section 4.3</a>
8h	ALIVE	PHY Alive Status register	<a href="#">Section 4.4</a>
Ch	LINK	PHY Link Status Register	<a href="#">Section 4.5</a>
10h	LINKINTRAW	MDIO Link Status Change Interrupt (Unmasked) Register	<a href="#">Section 4.6</a>
14h	LINKINTMASKED	MDIO Link Status Change Interrupt (Masked) Register	<a href="#">Section 4.7</a>
20h	USERINTRAW	MDIO User Command Complete Interrupt (Unmasked) Register	<a href="#">Section 4.8</a>
24h	USERINTMASKED	MDIO User Command Complete Interrupt (Masked) Register	<a href="#">Section 4.9</a>
28h	USERINTMASKSET	MDIO User Command Complete Interrupt Mask Set Register	<a href="#">Section 4.10</a>
2Ch	USERINTMASKCLEAR	MDIO User Command Complete Interrupt Mask Clear Register	<a href="#">Section 4.11</a>
80h	USERACCESS0	MDIO User Access Register 0	<a href="#">Section 4.12</a>
84h	USERPHYSEL0	MDIO User PHY Select Register 0	<a href="#">Section 4.13</a>
88h	USERACCESS1	MDIO User Access Register 1	<a href="#">Section 4.14</a>
8Ch	USERPHYSEL1	MDIO User PHY Select Register 1	<a href="#">Section 4.15</a>

## 4.2 MDIO Version Register (VERSION)

The MDIO version register (VERSION) is shown in [Figure 16](#) and described in [Table 15](#).

**Figure 16. MDIO Version Register (VERSION)**



LEGEND: R = Read only; R/W = Read/Write; -n = value after reset

**Table 15. MDIO Version Register (VERSION) Field Descriptions**

Bit	Field	Value	Description
31-16	MODID		Identifies the type of peripheral
15-8	REVMAJ		Management Interface Module major revision value
7-0	REVMIN		Management Interface Module minor revision value

### 4.3 MDIO Control Register (CONTROL)

The MDIO control register (CONTROL) is shown in [Figure 17](#) and described in [Table 16](#).

**Figure 17. MDIO Control Register (CONTROL)**

31	30	29	28	24	23	21	20	19	18	17	16
IDLE	ENABLE	Reserved	HIGHEST_USER_CHANNEL	Reserved	Reserved	PREAMBLE	FAULT	FAULT ENB	Reserved		
R-1	R/W-0	R-0	R-1	R-0	R-0	R/W-0	R/WC-0	R/W-0	R-0		
										15	0
CLKDIV											
R/W-255											

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

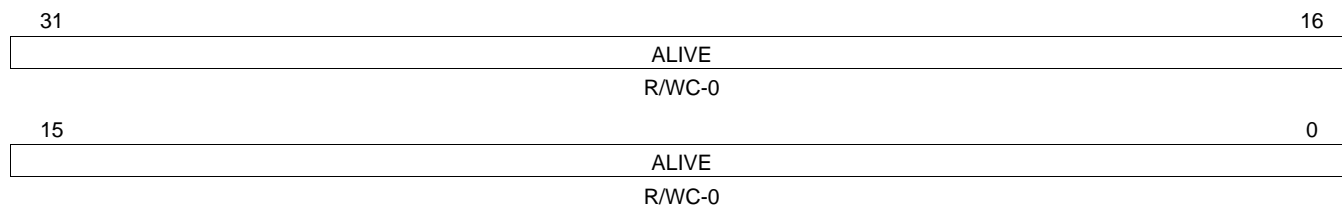
**Table 16. MDIO Control Register (CONTROL) Field Descriptions**

Bit	Field	Value	Description
31	IDLE	0 1	State machine IDLE status bit State machine is not in idle state State machine is in idle state
30	ENABLE	0 1	State machine enable control bit. . If the MDIO state machine is active at the time it is disabled, it will complete the current operation before halting and setting the idle bit. Disables the MDIO state machine Enable the MDIO state machine
29	Reserved	0	Reserved
28-24	HIGHEST_USER_CHANNEL		Highest user channel that is available in the module. It is currently set to 1. This implies that MDIOUserAccess1 is the highest available user access channel.
23-21	Reserved	0	Reserved
20	PREAMBLE	0 1	Preamble disable Standard MDIO preamble is used Disables this device from sending MDIO frame preambles
19	FAULT	0 1	Fault indicator. This bit is set to 1 if the MDIO pins fail to read back what the device is driving onto them. This indicates a physical layer fault and the module state machine is reset. Writing a 1 to it clears this bit. No failure Physical layer fault; the MDIO state machine is reset
18	FAULTENB	0 1	Fault detect enable. This bit has to be set to 1 to enable the physical layer fault detection. Disables the physical layer fault detection Enables the physical layer fault detection
17-16	Reserved	0	Reserved
15-0	CLKDIV		Clock Divider bits. This field specifies the division ratio between VBUS peripheral clock and the frequency of MDCLK. MDCLK is disabled when CLKDIV is set to 0. MDCLK frequency = peripheral clock frequency/(CLKDIV + 1).

#### 4.4 PHY Acknowledge Status Register (ALIVE)

The PHY acknowledge status register (ALIVE) is shown in [Figure 18](#) and described in [Table 17](#).

**Figure 18. PHY Acknowledge Status Register (ALIVE)**



LEGEND: R/W = Read/Write; R/WC = Read/Write 1 to clear; -n = value after reset

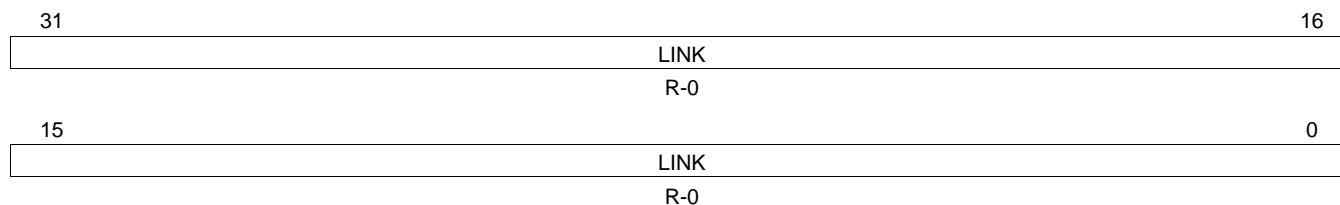
**Table 17. PHY Acknowledge Status Register (ALIVE) Field Descriptions**

Bit	Field	Value	Description
31-0	ALIVE		MDIO Alive bits. Each of the 32 bits of this register is set if the most recent access to the PHY with address corresponding to the register bit number was acknowledged by the PHY; the bit is reset if the PHY fails to acknowledge the access. Both the user and polling accesses to a PHY will cause the corresponding alive bit to be updated. The alive bits are only meant to be used to give an indication of the presence or not of a PHY with the corresponding address. Writing a 1 to any bit will clear it, writing a 0 has no effect.
		0	The PHY fails to acknowledge the access
		1	The most recent access to the PHY with an address corresponding to the register bit number was acknowledged by the PHY.

#### 4.5 PHY Link Status Register (LINK)

The PHY link status register (LINK) is shown in [Figure 19](#) and described in [Table 18](#).

**Figure 19. PHY Link Status Register (LINK)**



LEGEND: R = Read only; -n = value after reset

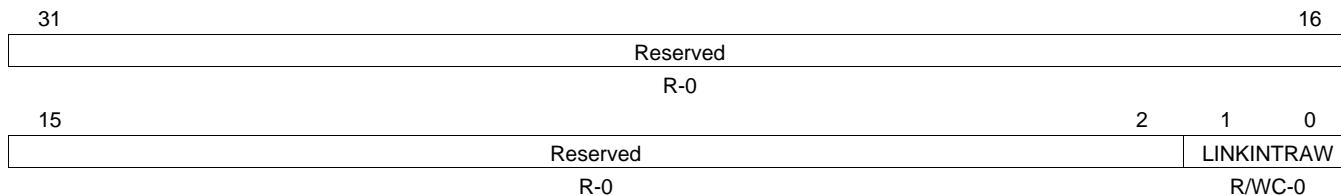
**Table 18. PHY Link Status Register (LINK) Field Descriptions**

Bit	Field	Value	Description
31-0	LINK		MDIO Link state bits. This register is updated after a read of the Generic Status Register of a PHY. The bit is set if the PHY with the corresponding address has link and the PHY acknowledges the read transaction. The bit is reset if the PHY indicates it does not have link or fails to acknowledge the read transaction. Writes to the register have no effect.
		0	The PHY indicates it does not have a link or fails to acknowledge the read transaction
		1	The PHY with the corresponding address has a link and the PHY acknowledges the read transaction

#### 4.6 MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRA<sup>W</sup>)

The MDIO link status change interrupt (unmasked) register (LINKINTRA<sup>W</sup>) is shown in [Figure 20](#) and described in [Table 19](#).

**Figure 20. MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRA<sup>W</sup>)**



LEGEND: R = Read only; R/WC = Read/Write 1 to clear; -n = value after reset

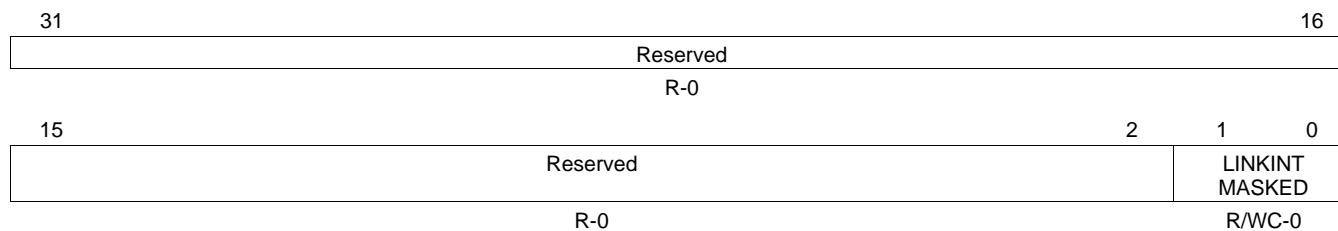
**Table 19. MDIO Link Status Change Interrupt (Unmasked) Register (LINKINTRA<sup>W</sup>) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	LINKINTRA <sup>W</sup>		MDIO Link change event, raw value. When asserted, a bit indicates that there was an MDIO link change event (i.e. change in the LINK register) corresponding to the PHY address in the USERPHYSEL register. LINKINTRA <sup>W</sup> [0] and LINKINTRA <sup>W</sup> [1] correspond to USERPHYSEL0 and USERPHYSEL1, respectively. Writing a 1 will clear the event and writing 0 has no effect.

#### 4.7 MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED)

The MDIO link status change interrupt (masked) register (LINKINTMASKED) is shown in [Figure 21](#) and described in [Table 20](#).

**Figure 21. MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED)**



LEGEND: R = Read only; R/WC = Read/Write 1 to clear; -n = value after reset

**Table 20. MDIO Link Status Change Interrupt (Masked) Register (LINKINTMASKED) Field Descriptions**

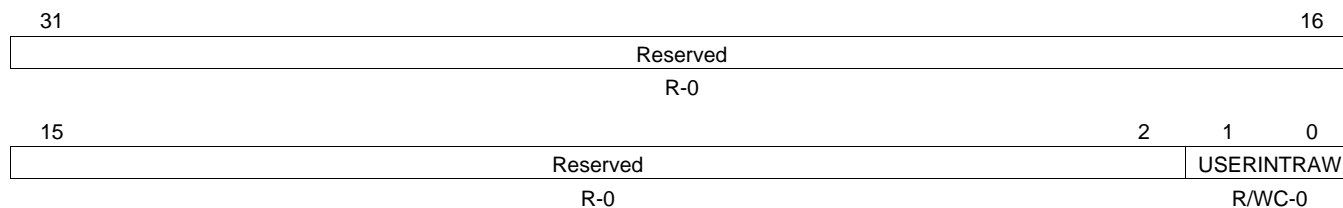
Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	LINKINTMASKED		MDIO Link change interrupt, masked value. . When asserted, a bit indicates that there was an MDIO link change event (i.e. change in the LINK register) corresponding to the PHY address in the USERPHYSEL register and the corresponding LINKINTENB bit was set. LINKINTRAW[0] and LINKINTRAW[1] correspond to USERPHYSEL0 and USERPHYSEL1, respectively. Writing a 1 will clear the event and writing 0 has no effect.



#### 4.8 MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW)

The MDIO user command complete interrupt (unmasked) register (USERINTRAW) is shown in [Figure 22](#) and described in [Table 21](#).

**Figure 22. MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW)**



LEGEND: R = Read only; R/WC = Read/Write 1 to clear; -n = value after reset

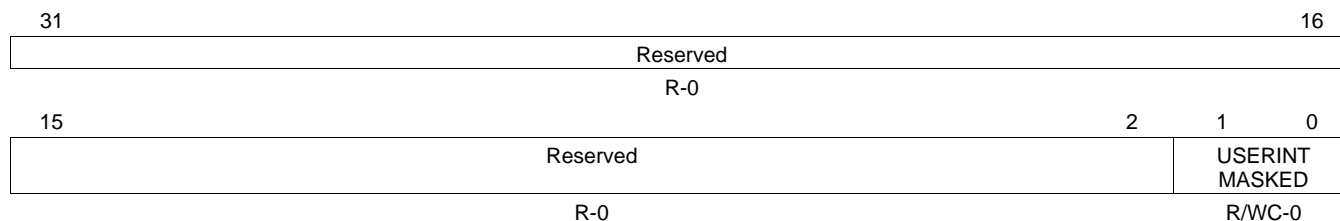
**Table 21. MDIO User Command Complete Interrupt (Unmasked) Register (USERINTRAW) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTRAW		MDIO User command complete event bits. When asserted, a bit indicates that the previously scheduled PHY read or write command using that particular USERACCESS register has completed. Writing a 1 will clear the event and writing 0 has no effect.

#### 4.9 MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED)

The MDIO user command complete interrupt (Masked) register (USERINTMASKED) is shown in [Figure 23](#) and described in [Table 22](#).

**Figure 23. MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED)**



LEGEND: R = Read only; R/WC = Read/Write 1 to clear; -n = value after reset

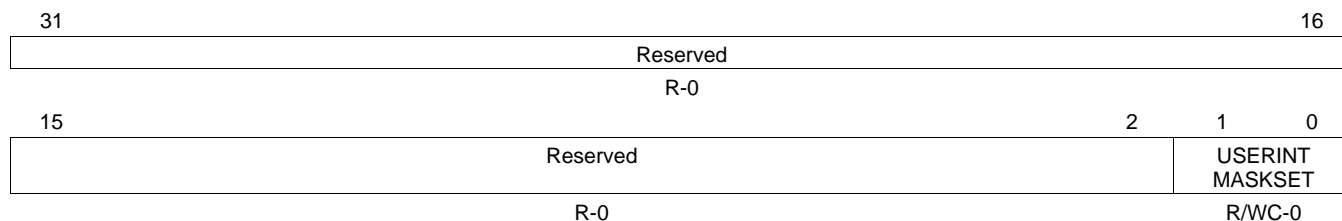
**Table 22. MDIO User Command Complete Interrupt (Masked) Register (USERINTMASKED) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTMASKED		Masked value of MDIO User command complete interrupt. When asserted, a bit indicates that the previously scheduled PHY read or write command using that particular USERACCESS register has completed and the corresponding USERINTMASKSET bit is set to 1. Writing a 1 will clear the interrupt and writing 0 has no effect.

#### 4.10 MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET)

The MDIO user command complete interrupt mask set register (USERINTMASKSET) is shown in Figure 24 and described in Table 23.

**Figure 24. MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET)**



LEGEND: R = Read only; R/WC = Read/Write 1 to clear; -n = value after reset

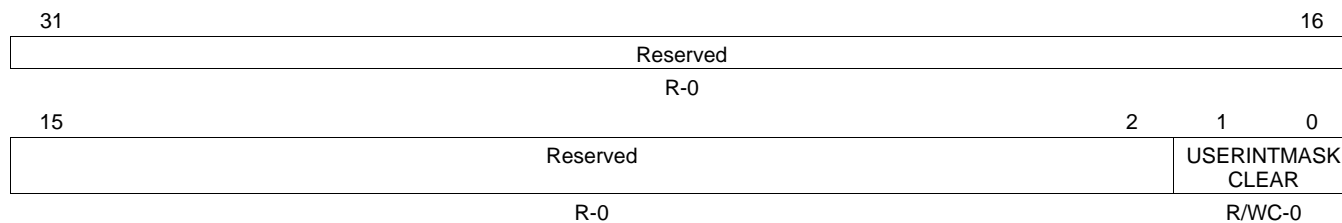
**Table 23. MDIO User Command Complete Interrupt Mask Set Register (USERINTMASKSET) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTMASKSET		MDIO user interrupt mask set for USERINTMASKED[1:0] respectively. Setting a bit to 1 will enable MDIO user command complete interrupts for that particular USERACCESS register. MDIO user interrupt for a particular USERACCESS register is disabled if the corresponding bit is 0. Writing a 0 to this register has no effect.

#### 4.11 MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR)

The MDIO user command complete interrupt mask clear register (USERINTMASKCLEAR) is shown in [Figure 25](#) and described in [Table 24](#).

**Figure 25. MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR)**



LEGEND: R = Read only; R/WC = Read/Write 1 to clear; -n = value after reset

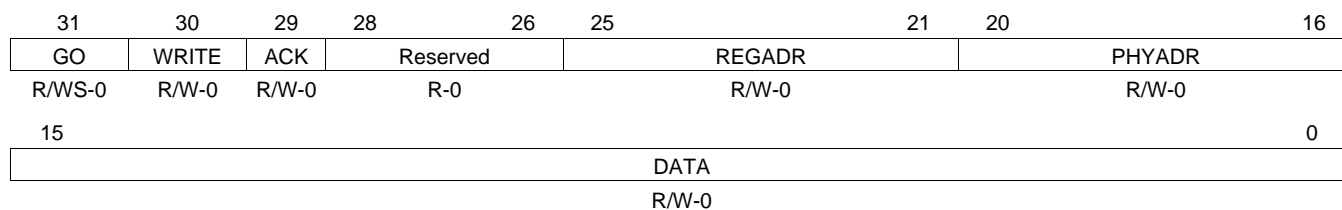
**Table 24. MDIO User Command Complete Interrupt Mask Clear Register (USERINTMASKCLEAR)  
Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1-0	USERINTMASK CLEAR		MDIO user command complete interrupt mask clear for USERINTMASKED[1:0] respectively. Setting a bit to 1 will disable further user command complete interrupts for that particular USERACCESS register. Writing a 0 to this register has no effect.

#### 4.12 MDIO User Access Register 0 (USERACCESS0)

The MDIO user access register 0 (USERACCESS0) is shown in [Figure 26](#) and described in [Table 25](#).

**Figure 26. MDIO User Access Register 0 (USERACCESS0)**



LEGEND: R = Read only; R/W = Read/Write; R/WS = Read/Write 1 to set; -n = value after reset

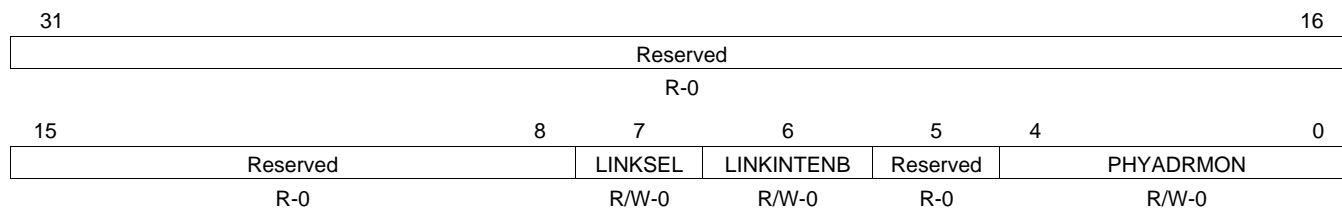
**Table 25. MDIO User Access Register 0 (USERACCESS0) Field Descriptions**

Bit	Field	Value	Description
31	GO		Go bit. Writing a 1 to this bit causes the MDIO state machine to perform an MDIO access when it is convenient for it to do so; this is not an instantaneous process. Writing a 0 to this bit has no effect. This bit is writeable only if the MDIO state machine is enabled. This bit will self clear when the requested access has been completed. Any writes to the USERACCESS0 register are blocked when the GO bit is 1.
30	WRITE	0 1	Write enable bit. Setting this bit to a 1 causes the MDIO transaction to be a register write, otherwise it is a register read. The user command is a read operation The user command is a write operation
29	ACK		Acknowledge bit. This bit is set if the PHY acknowledged the read transaction.
28-26	Reserved	0	Reserved
25-21	REGADR		Register address bits. This field specifies the PHY register to be accessed for this transaction
20-16	PHYADR		PHY address bits. This field specifies the PHY to be accesses for this transaction
15-0	DATA		User data bits. These bits specify the data value read from or to be written to the specified PHY register.

### 4.13 MDIO User PHY Select Register 0 (USERPHYSEL0)

The MDIO user PHY select register 0 (USERPHYSEL0) is shown in [Figure 27](#) and described in [Table 26](#).

**Figure 27. MDIO User PHY Select Register 0 (USERPHYSEL0)**



LEGEND: R = Read only; R/W = Read/Write; -n = value after reset

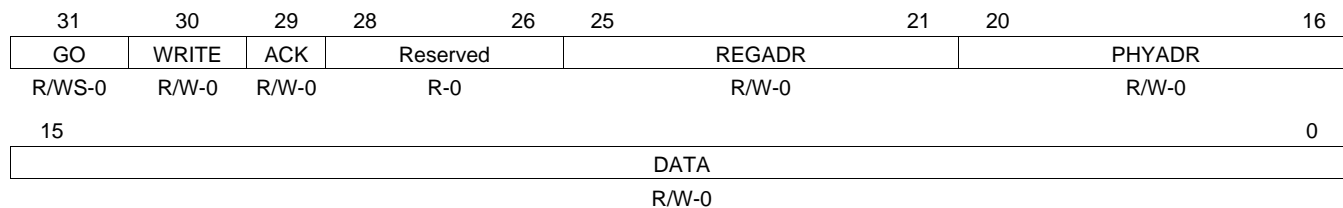
**Table 26. MDIO User PHY Select Register 0 (USERPHYSEL0) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	LINKSEL		Link status determination select bit. Default value is 0 which implies that the link status is determined by the MDIO state machine. This is the only option supported on this device.
6	LINKINTENB	0	Link change interrupt enable. Set to 1 to enable link change status interrupts for PHY address specified in PHYADRMON. Link change interrupts are disabled if this bit is set to 0.
		1	Link change interrupts are disabled
			Link change status interrupts for PHY address specified in PHYADRMON bits are enabled
5	Reserved	0	Reserved
4-0	PHYADRMON		PHY address whose link status is to be monitored

#### 4.14 MDIO User Access Register 1 (USERACCESS1)

The MDIO user access register 1 (USERACCESS1) is shown in [Figure 28](#) and described in [Table 27](#).

**Figure 28. MDIO User Access Register 1 (USERACCESS1)**



LEGEND: R = Read only; R/W = Read/Write; R/WS = Read/Write 1 to set; -n = value after reset

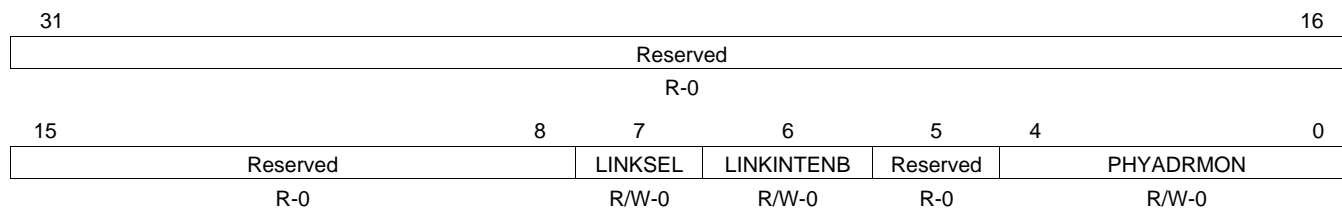
**Table 27. MDIO User Access Register 1 (USERACCESS1) Field Descriptions**

Bit	Field	Value	Description
31	GO		Go bit. Writing a 1 to this bit causes the MDIO state machine to perform an MDIO access when it is convenient for it to do so, this is not an instantaneous process. Writing a 0 to this bit has no effect. This bit is write able only if the MDIO state machine is enabled. This bit will self clear when the requested access has been completed. Any writes to the USERACCESS0 register are blocked when the go bit is 1.
30	WRITE	0 1	Write enable bit. Setting this bit to a 1 causes the MDIO transaction to be a register write, otherwise it is a register read. The user command is a read operation The user command is a write operation
29	ACK		Acknowledge bit. This bit is set if the PHY acknowledged the read transaction.
28-26	Reserved	0	Reserved
25-21	REGADR		Register address bits. This field specifies the PHY register to be accessed for this transaction
20-16	PHYADR		PHY address bits. This field specifies the PHY to be accesses for this transaction
15-0	DATA		User data bits. These bits specify the data value read from or to be written to the specified PHY register.

#### 4.15 MDIO User PHY Select Register 1 (USERPHYSEL1)

The MDIO user PHY select register 1 (USERPHYSEL1) is shown in [Figure 29](#) and described in [Table 28](#).

**Figure 29. MDIO User PHY Select Register 1 (USERPHYSEL1)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 28. MDIO User PHY Select Register 1 (USERPHYSEL1) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	LINKSEL		Link status determination select bit. Default value is 0 which implies that the link status is determined by the MDIO state machine. This is the only option supported on this device.
6	LINKINTENB	0	Link change interrupt enable. Set to 1 to enable link change status interrupts for PHY address specified in PHYADDRMON. Link change interrupts are disabled if this bit is set to 0.
		1	Link change interrupts are disabled
			Link change status interrupts for PHY address specified in PHYADDRMON bits are enabled
5	Reserved	0	PHY address whose link status is to be monitored
4-0	PHYADDRMON		PHY address whose link status is to be monitored



## 5 EMAC Port Registers

### 5.1 Introduction

Table 29 lists the memory-mapped registers for the Ethernet Media Access Controller (EMAC). See the device-specific data manual for the memory address of these registers.

**Table 29. Ethernet Media Access Controller (EMAC) Registers**

Offset	Acronym	Register Description	Section
0h	TXIDVER	Transmit Identification and Version Register	<a href="#">Section 5.2</a>
4h	TXCONTROL	Transmit Control Register	<a href="#">Section 5.3</a>
8h	TXTEARDOWN	Transmit Teardown Register	<a href="#">Section 5.4</a>
10h	RXIDVER	Receive Identification and Version Register	<a href="#">Section 5.5</a>
14h	RXCONTROL	Receive Control Register	<a href="#">Section 5.6</a>
18h	RXTEARDOWN	Receive Teardown Register	<a href="#">Section 5.7</a>
80h	TXINTSTATRAW	Transmit Interrupt Status (Unmasked) Register	<a href="#">Section 5.8</a>
84h	TXINTSTATMASKED	Transmit Interrupt Status (Masked) Register	<a href="#">Section 5.9</a>
88h	TXINTMASKSET	Transmit Interrupt Mask Set Register	<a href="#">Section 5.10</a>
8Ch	TXINTMASKCLEAR	Transmit Interrupt Clear Register	<a href="#">Section 5.11</a>
90h	MACINVECTOR	MAC Input Vector Register	<a href="#">Section 5.12</a>
A0h	RXINTSTATRAW	Receive Interrupt Status (Unmasked) Register	<a href="#">Section 5.13</a>
A4h	RXINTSTATMASKED	Receive Interrupt Status (Masked) Register	<a href="#">Section 5.14</a>
A8h	RXINTMASKSET	Receive Interrupt Mask Set Register	<a href="#">Section 5.15</a>
ACCh	RXINTMASKCLEAR	Receive Interrupt Mask Clear Register	<a href="#">Section 5.16</a>
B0h	MACINTSTATRAW	MAC Interrupt Status (Unmasked) Register	<a href="#">Section 5.17</a>
B4h	MACINTSTATMASKED	MAC Interrupt Status (Masked) Register	<a href="#">Section 5.18</a>
B8h	MACINTMASKSET	MAC Interrupt Mask Set Register	<a href="#">Section 5.19</a>
BCh	MACINTMASKCLEAR	MAC Interrupt Mask Clear Register	<a href="#">Section 5.20</a>
100h	RXMBPENABLE	Receive Multicast/Broadcast/Promiscuous Channel Enable Register	<a href="#">Section 5.21</a>
104h	RXUNICASTSET	Receive Unicast Enable Set Register	<a href="#">Section 5.22</a>
108h	RXUNICASTCLEAR	Receive Unicast Clear Register	<a href="#">Section 5.23</a>
10Ch	RXMAXLEN	Receive Maximum Length Register	<a href="#">Section 5.24</a>
110h	RXBUFFEROFFSET	Receive Buffer Offset Register	<a href="#">Section 5.25</a>
114h	RXFILTERLOWTHRESH	Receive Filter Low Priority Frame Threshold Register	<a href="#">Section 5.26</a>
120h	RX0FLOWTHRESH	Receive Channel 0 Flow Control Threshold Register	<a href="#">Section 5.27</a>
124h	RX1FLOWTHRESH	Receive Channel 1 Flow Control Threshold Register	<a href="#">Section 5.27</a>
128h	RX2FLOWTHRESH	Receive Channel 2 Flow Control Threshold Register	<a href="#">Section 5.27</a>
12Ch	RX3FLOWTHRESH	Receive Channel 3 Flow Control Threshold Register	<a href="#">Section 5.27</a>
130h	RX4FLOWTHRESH	Receive Channel 4 Flow Control Threshold Register	<a href="#">Section 5.27</a>
134h	RX5FLOWTHRESH	Receive Channel 5 Flow Control Threshold Register	<a href="#">Section 5.27</a>
138h	RX6FLOWTHRESH	Receive Channel 6 Flow Control Threshold Register	<a href="#">Section 5.27</a>
13Ch	RX7FLOWTHRESH	Receive Channel 7 Flow Control Threshold Register	<a href="#">Section 5.27</a>
140h	RX0FREEBUFFER	Receive Channel 0 Free Buffer Count Register	<a href="#">Section 5.28</a>
144h	RX1FREEBUFFER	Receive Channel 1 Free Buffer Count Register	<a href="#">Section 5.28</a>
148h	RX2FREEBUFFER	Receive Channel 2 Free Buffer Count Register	<a href="#">Section 5.28</a>
14Ch	RX3FREEBUFFER	Receive Channel 3 Free Buffer Count Register	<a href="#">Section 5.28</a>
150h	RX4FREEBUFFER	Receive Channel 4 Free Buffer Count Register	<a href="#">Section 5.28</a>
154h	RX5FREEBUFFER	Receive Channel 5 Free Buffer Count Register	<a href="#">Section 5.28</a>
158h	RX6FREEBUFFER	Receive Channel 6 Free Buffer Count Register	<a href="#">Section 5.28</a>

**Table 29. Ethernet Media Access Controller (EMAC) Registers (continued)**

Offset	Acronym	Register Description	Section
15Ch	RX7FREEBUFFER	Receive Channel 7 Free Buffer Count Register	<a href="#">Section 5.28</a>
160h	MACCONTROL	MAC Control Register	<a href="#">Section 5.29</a>
164h	MACSTATUS	MAC Status Register	<a href="#">Section 5.30</a>
168h	EMCONTROL	Emulation Control Register	<a href="#">Section 5.31</a>
16Ch	FIFOCONTROL	FIFO Control Register	<a href="#">Section 5.32</a>
170h	MACCONFIG	MAC Configuration Register	<a href="#">Section 5.33</a>
174h	SOFTRESET	Soft Reset Register	<a href="#">Section 5.34</a>
1D0h	MACSRCADDRLO	MAC Source Address Low Bytes Register	<a href="#">Section 5.35</a>
1D4h	MACSRCADDRHI	MAC Source Address High Bytes Register	<a href="#">Section 5.36</a>
1D8h	MACHASH1	MAC Hash Address Register 1	<a href="#">Section 5.37</a>
1DCh	MACHASH2	MAC Hash Address Register 2	<a href="#">Section 5.38</a>
1E0h	BOFFTEST	Back Off Test Register	<a href="#">Section 5.39</a>
1E4h	TPACETEST	Transmit Pacing Algorithm Test Register	<a href="#">Section 5.40</a>
1E8h	RXPAUSE	Receive Pause Timer Register	<a href="#">Section 5.41</a>
1ECh	TXPAUSE	Transmit Pause Timer Register	<a href="#">Section 5.42</a>
500h	MACADDRLO	MAC Address Low Bytes Register, Used in Receive Address Matching	<a href="#">Section 5.43</a>
504h	MACADDRHI	MAC Address High Bytes Register, Used in Receive Address Matching	<a href="#">Section 5.44</a>
508h	MACINDEX	MAC Index Register	<a href="#">Section 5.45</a>
600h	TX0HDP	Transmit Channel 0 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
604h	TX1HDP	Transmit Channel 1 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
608h	TX2HDP	Transmit Channel 2 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
60Ch	TX3HDP	Transmit Channel 3 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
610h	TX4HDP	Transmit Channel 4 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
614h	TX5HDP	Transmit Channel 5 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
618h	TX6HDP	Transmit Channel 6 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
61Ch	TX7HDP	Transmit Channel 7 DMA Head Descriptor Pointer Register	<a href="#">Section 5.46</a>
620h	RX0HDP	Receive Channel 0 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
624h	RX1HDP	Receive Channel 1 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
628h	RX2HDP	Receive Channel 2 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
62Ch	RX3HDP	Receive Channel 3 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
630h	RX4HDP	Receive Channel 4 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
634h	RX5HDP	Receive Channel 5 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
638h	RX6HDP	Receive Channel 6 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
63Ch	RX7HDP	Receive Channel 7 DMA Head Descriptor Pointer Register	<a href="#">Section 5.47</a>
640h	TX0CP	Transmit Channel 0 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
644h	TX1CP	Transmit Channel 1 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
648h	TX2CP	Transmit Channel 2 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
64Ch	TX3CP	Transmit Channel 3 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
650h	TX4CP	Transmit Channel 4 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
654h	TX5CP	Transmit Channel 5 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>

**Table 29. Ethernet Media Access Controller (EMAC) Registers (continued)**

Offset	Acronym	Register Description	Section
658h	TX6CP	Transmit Channel 6 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
65Ch	TX7CP	Transmit Channel 7 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.48</a>
660h	RX0CP	Receive Channel 0 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
664h	RX1CP	Receive Channel 1 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
668h	RX2CP	Receive Channel 2 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
66Ch	RX3CP	Receive Channel 3 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
670h	RX4CP	Receive Channel 4 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
674h	RX5CP	Receive Channel 5 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
678h	RX6CP	Receive Channel 6 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
67Ch	RX7CP	Receive Channel 7 Completion Pointer (Interrupt Acknowledge) Register	<a href="#">Section 5.49</a>
<b>Network Statistics Registers</b>			
200h	RXGOODFRAMES	Good Receive Frames	<a href="#">Section 5.50.1</a>
204h	RXBCASTFRAMES	Total number of good broadcast frames received	<a href="#">Section 5.50.2</a>
208h	RXMCASTFRAMES	Total number of good multicast frames received	<a href="#">Section 5.50.3</a>
20ch	RXPAUSEFRAMES	Pause Receive Frames Register	<a href="#">Section 5.50.4</a>
210h	RXCRCERRORS	Total number of frames received with CRC errors	<a href="#">Section 5.50.5</a>
214h	RXALIGNCODEERRORS	Total number of frames received with alignment/code errors	<a href="#">Section 5.50.6</a>
218h	RXOVERSIZED	Total number of oversized frames received	<a href="#">Section 5.50.7</a>
21ch	RXJABBER	Total number of jabber frames received	<a href="#">Section 5.50.8</a>
220h	RXUNDERSIZED	Total number of undersized frames received	<a href="#">Section 5.50.9</a>
224h	RXFRAGMENTS	Receive Frame Fragments Register	<a href="#">Section 5.50.10</a>
228h	RXFILTERED	Filtered Receive Frames	<a href="#">Section 5.50.11</a>
22Ch	RXQOSFILTERED	Received Frames Filtered by QOS	<a href="#">Section 5.50.12</a>
230h	RXOCTETS	Total number of received bytes in good frames	<a href="#">Section 5.50.13</a>
234h	TXGOODFRAMES	Total number of good frames transmitted	<a href="#">Section 5.50.14</a>
238h	TXBCASTFRAMES	Broadcast Transmit Frames Register	<a href="#">Section 5.50.15</a>
23Ch	TXMCASTFRAMES	Multicast Transmit Frames Register	<a href="#">Section 5.50.16</a>
240h	TXPAUSEFRAMES	Pause Transmit Frames Register	<a href="#">Section 5.50.17</a>
244h	TXDEFERRED	Deferred Transmit Frames Register	<a href="#">Section 5.50.18</a>
248h	TXCOLLISION	Transmit Collision Frames Register	<a href="#">Section 5.50.19</a>
24Ch	TXSINGLECOLL	Transmit Single Collision Frames Register	<a href="#">Section 5.50.20</a>
250h	TXMULTICOLL	Transmit Multiple Collision Frames Register	<a href="#">Section 5.50.21</a>
254h	TXEXCESSIVECOLL	Transmit Excessive Collision Frames Register	<a href="#">Section 5.50.22</a>
258h	TXLATECOLL	Transmit Late Collision Frames Register	<a href="#">Section 5.50.23</a>
25Ch	TXUNDERRUN	Transmit Underrun Error Register	<a href="#">Section 5.50.24</a>
260h	TXCARRIERSENSE	Transmit Carrier Sense Errors Register	<a href="#">Section 5.50.25</a>
264h	TXOCTETS	Transmit Octet Frames Register	<a href="#">Section 5.50.26</a>
268h	FRAME64	Transmit and Receive 64 Octet Frames Register	<a href="#">Section 5.50.27</a>
26Ch	FRAME65T127	Transmit and Receive 65 to 127 Octet Frames Register	<a href="#">Section 5.50.28</a>

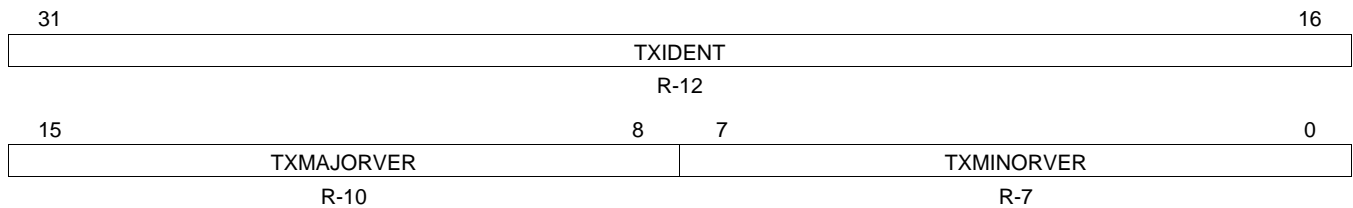
**Table 29. Ethernet Media Access Controller (EMAC) Registers (continued)**

Offset	Acronym	Register Description	Section
270h	FRAME128T255	Transmit and Receive 128 to 255 Octet Frames Register	<a href="#">Section 5.50.29</a>
274h	FRAME256T511	Transmit and Receive 256 to 511 Octet Frames Register	<a href="#">Section 5.50.30</a>
278h	FRAME512T1023	Transmit and Receive 512 to 1023 Octet Frames Register	<a href="#">Section 5.50.31</a>
27Ch	FRAME1024TUP	Transmit and Receive 1024 to RXMAXLEN Octet Frames Register	<a href="#">Section 5.50.32</a>
280h	NETOCTETS	Network Octet Frames Register	<a href="#">Section 5.50.33</a>
284h	RXSOFOVERRUNS	Receive FIFO or DMA Start of Frame Overruns Register	<a href="#">Section 5.50.34</a>
288h	RXMOFOVERRUNS	Receive FIFO or DMA Middle of Frame Overruns Register	<a href="#">Section 5.50.35</a>
28Ch	RXDMAOVERRUNS	Receive DMA Start of Frame and Middle of Frame Overruns Register	<a href="#">Section 5.50.36</a>

## 5.2 Transmit Identification and Version Register (TXIDVER)

The transmit identification and version register (TXIDVER) is shown in [Figure 30](#) and described in [Table 30](#).

**Figure 30. Transmit Identification and Version Register (TXIDVER)**



LEGEND: R = Read only; -n = value after reset

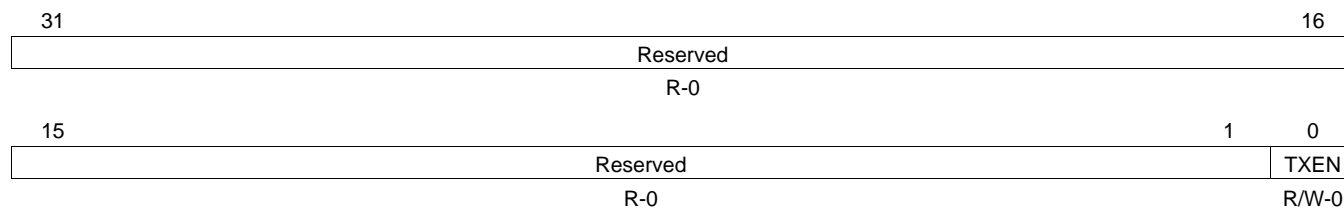
**Table 30. Transmit Identification and Version Register (TXIDVER) Field Descriptions**

Bit	Field	Value	Description
31-16	TXIDENT		Transmit identification value
15-8	TXMAJORVER		Transmit major version value
7-0	TXMINORVER		Transmit minor version value

### 5.3 Transmit Control Register (TXCONTROL)

The transmit control register (TXCONTROL) is shown in [Figure 31](#) and described in [Table 31](#).

**Figure 31. Transmit Control Register (TXCONTROL)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

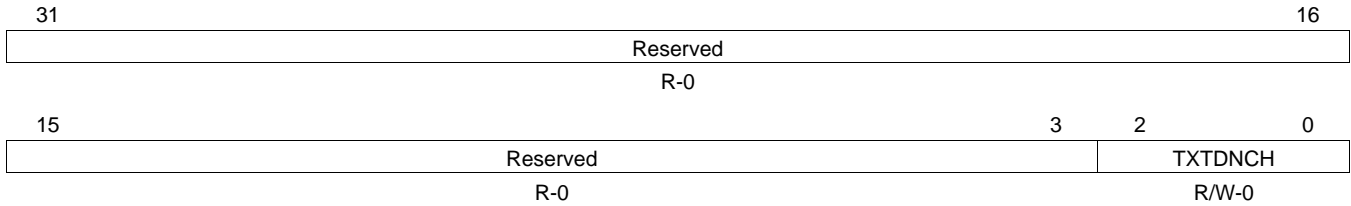
**Table 31. Transmit Control Register (TXCONTROL) Field Descriptions**

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	TXEN		Transmit enable
		0	Transmit is disabled
		1	Transmit is enabled

## 5.4 Transmit Teardown Register (TXTEARDOWN)

The transmit teardown register (TXTEARDOWN) is shown in [Figure 32](#) and described in [Table 32](#).

**Figure 32. Transmit Teardown Register (TXTEARDOWN)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

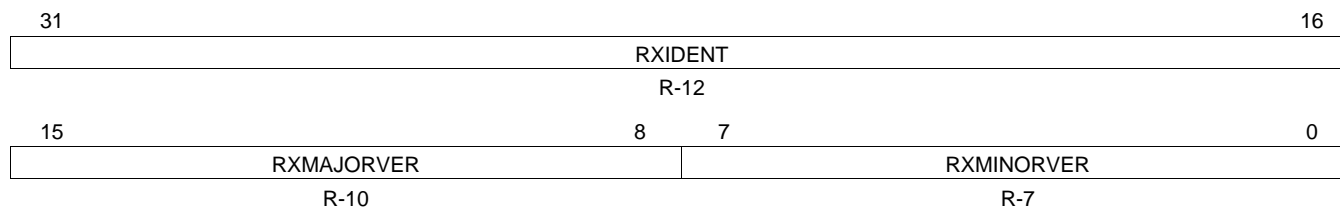
**Table 32. Transmit Teardown Register (TXTEARDOWN) Field Descriptions**

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	TXTDNCH	0-3h	Transmit teardown channel. Transmit channel teardown is commanded by writing the encoded value of the transmit channel to be torn down. The teardown register is read as zero.
		0	Teardown transmit channel 0
		1h	Teardown transmit channel 1
		2h	Teardown transmit channel 2
		3h	Teardown transmit channel 3
		4h	Teardown transmit channel 4
		5h	Teardown transmit channel 5
		6h	Teardown transmit channel 6
		7h	Teardown transmit channel 7

## 5.5 Receive Identification and Version Register (RXIDVER)

The receive identification and version register (RXIDVER) is shown in [Figure 33](#) and described in [Table 33](#).

**Figure 33. Receive Identification and Version Register (RXIDVER)**



LEGEND: R = Read only; -n = value after reset

**Table 33. Receive Identification and Version Register (RXIDVER) Field Descriptions**

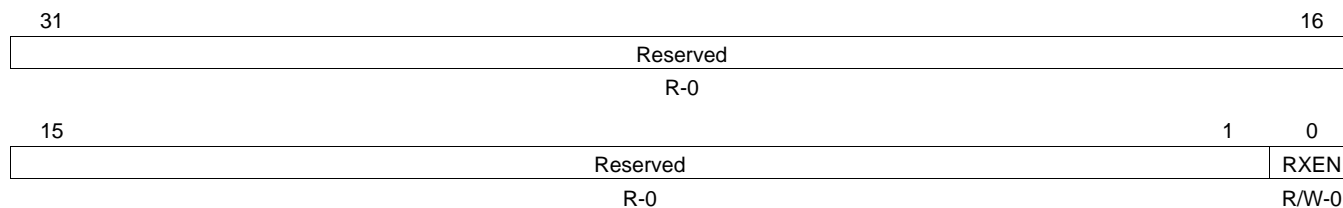
Bit	Field	Value	Description
31-16	RXIDENT		Receive identification value
15-8	RXMAJORVER		Receive major version value
7-0	RXMINORVER		Receive minor version value



## 5.6 Receive Control Register (RXCONTROL)

The receive control register (RXCONTROL) is shown in [Figure 34](#) and described in [Table 34](#).

**Figure 34. Receive Control Register (RXCONTROL)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

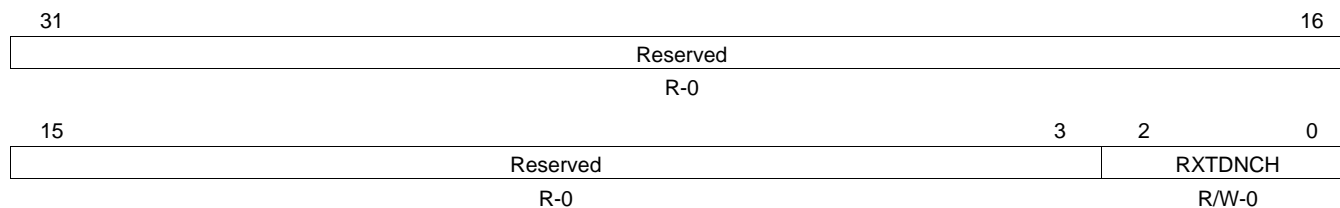
**Table 34. Receive Control Register (RXCONTROL) Field Descriptions**

Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	RXEN	0	Receive DMA enable Receive is disabled
		1	Receive is enabled

## 5.7 Receive Teardown Register (RXTEARDOWN)

The receive teardown register (RXTEARDOWN) is shown in [Figure 35](#) and described in [Table 35](#).

**Figure 35. Receive Teardown Register (RXTEARDOWN)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

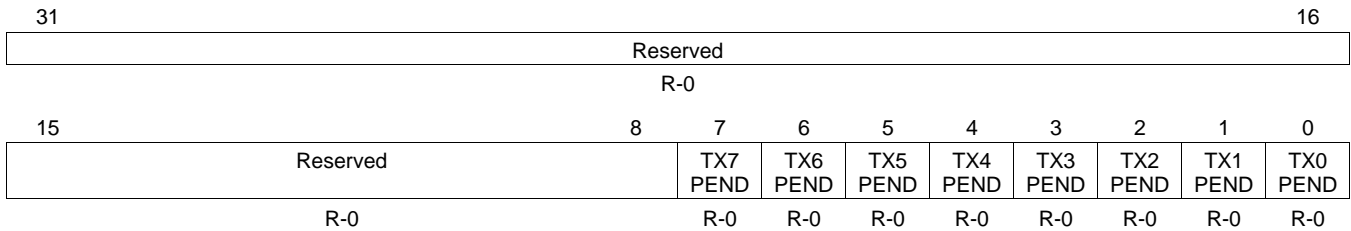
**Table 35. Receive Teardown Register (RXTEARDOWN) Field Descriptions**

Bit	Field	Value	Description
31-3	Reserved	0	Reserved
2-0	RXTDNCH	0-3h	Receive teardown channel. Receive channel teardown is commanded by writing the encoded value of the receive channel to be torn down. The teardown register is read as zero.
		0	Teardown receive channel 0
		1h	Teardown receive channel 1
		2h	Teardown receive channel 2
		3h	Teardown receive channel 3
		4h	Teardown receive channel 4
		5h	Teardown receive channel 5
		6h	Teardown receive channel 6
		7h	Teardown receive channel 7

### 5.8 Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW)

The transmit interrupt status (unmasked) register (TXINTSTATRAW) is shown in [Figure 36](#) and described in [Table 36](#).

**Figure 36. Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW)**



LEGEND: R = Read only; -n = value after reset

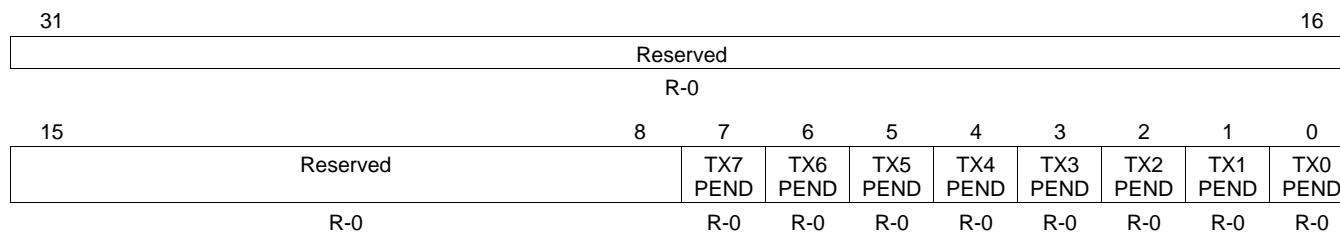
**Table 36. Transmit Interrupt Status (Unmasked) Register (TXINTSTATRAW) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7PEND		TX7PEND raw interrupt read (before mask)
6	TX6PEND		TX6PEND raw interrupt read (before mask)
5	TX5PEND		TX5PEND raw interrupt read (before mask)
4	TX4PEND		TX4PEND raw interrupt read (before mask)
3	TX3PEND		TX3PEND raw interrupt read (before mask)
2	TX2PEND		TX2PEND raw interrupt read (before mask)
1	TX1PEND		TX1PEND raw interrupt read (before mask)
0	TX0PEND		TX0PEND raw interrupt read (before mask)

## 5.9 Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED)

The transmit interrupt status (Masked) register (TXINTSTATMASKED) is shown in [Figure 37](#) and described in [Table 37](#).

**Figure 37. Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED)**



LEGEND: R/W = R = Read only; -n = value after reset

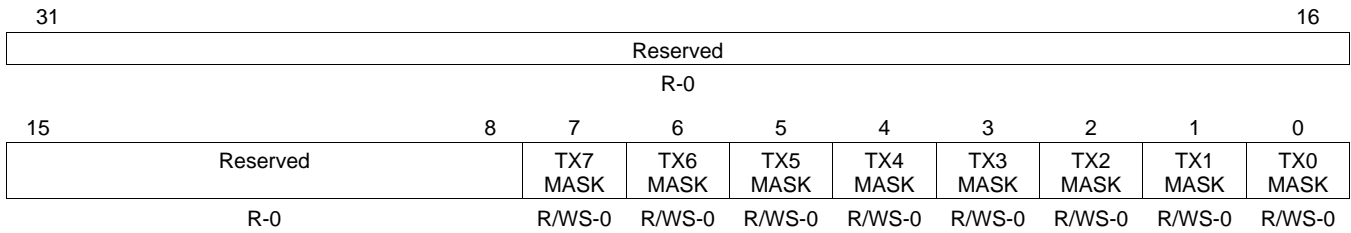
**Table 37. Transmit Interrupt Status (Masked) Register (TXINTSTATMASKED) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7PEND		TX7PEND masked interrupt read
6	TX6PEND		TX6PEND masked interrupt read
5	TX5PEND		TX5PEND masked interrupt read
4	TX4PEND		TX4PEND masked interrupt read
3	TX3PEND		TX3PEND masked interrupt read
2	TX2PEND		TX2PEND masked interrupt read
1	TX1PEND		TX1PEND masked interrupt read
0	TX0PEND		TX0PEND masked interrupt read

### 5.10 Transmit Interrupt Mask Set Register (TXINTMASKSET)

The transmit interrupt mask set register (TXINTMASKSET) is shown in [Figure 38](#) and described in [Table 38](#).

**Figure 38. Transmit Interrupt Mask Set Register (TXINTMASKSET)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

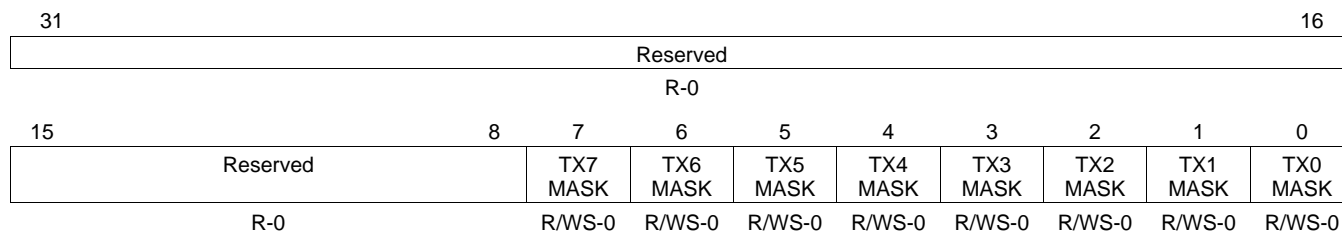
**Table 38. Transmit Interrupt Mask Set Register (TXINTMASKSET) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7MASK		Transmit channel 7 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
6	TX6MASK		Transmit channel 6 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
5	TX5MASK		Transmit channel 5 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
4	TX4MASK		Transmit channel 4 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
3	TX3MASK		Transmit channel 3 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
2	TX2MASK		Transmit channel 2 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
1	TX1MASK		Transmit channel 1 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
0	TX0MASK		Transmit channel 0 interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.

### 5.11 Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR)

The transmit interrupt mask clear register (TXINTMASKCLEAR) is shown in [Figure 39](#) and described in [Table 39](#).

**Figure 39. Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

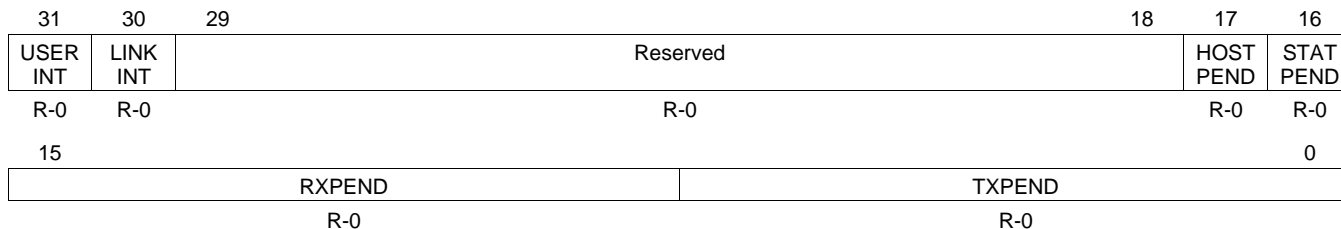
**Table 39. Transmit Interrupt Mask Clear Register (TXINTMASKCLEAR) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	TX7MASK		Transmit channel 7 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
6	TX6MASK		Transmit channel 6 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
5	TX5MASK		Transmit channel 5 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
4	TX4MASK		Transmit channel 4 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
3	TX3MASK		Transmit channel 3 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
2	TX2MASK		Transmit channel 2 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
1	TX1MASK		Transmit channel 1 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
0	TX0MASK		Transmit channel 0 interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.

## 5.12 MAC Input Vector Register (MACINVECTOR)

The MAC input vector register (MACINVECTOR) is shown in [Figure 40](#) and described in [Table 40](#).

**Figure 40. MAC Input Vector Register (MACINVECTOR)**



LEGEND: R = Read only; -n = value after reset

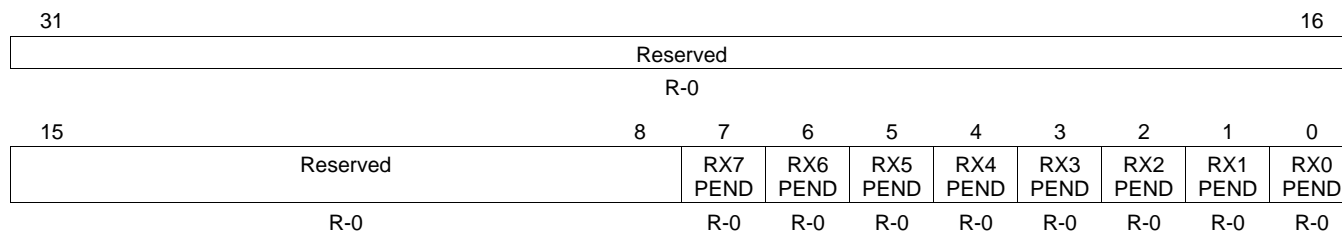
**Table 40. MAC Input Vector Register (MACINVECTOR) Field Descriptions**

Bit	Field	Value	Description
31	USERINT		MDIO module user interrupt (USERINT) pending status bit
30	LINKINT		MDIO module link change interrupt (LINKINT) pending status bit
29-18	Reserved	0	Reserved
17	HOSTPEND		EMAC module host error interrupt (HOSTPEND) pending status bit
16	STATPEND		EMAC module statistics interrupt (STATPEND) pending status bit
15-8	RXPEND		Receive channels 0-7 interrupt (RX $n$ PEND) pending status bit. Bit 8 is receive channel 0.
7-0	TXPEND		Transmit channels 0-7 interrupt (TX $n$ PEND) pending status bit. Bit 0 is transmit channel 0.

### 5.13 Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW)

The receive interrupt status (Unmasked) register (RXINTSTATRAW) is shown in [Figure 41](#) and described in [Table 41](#).

**Figure 41. Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW)**



LEGEND: R = Read only; -n = value after reset

**Table 41. Receive Interrupt Status (Unmasked) Register (RXINTSTATRAW) Field Descriptions**

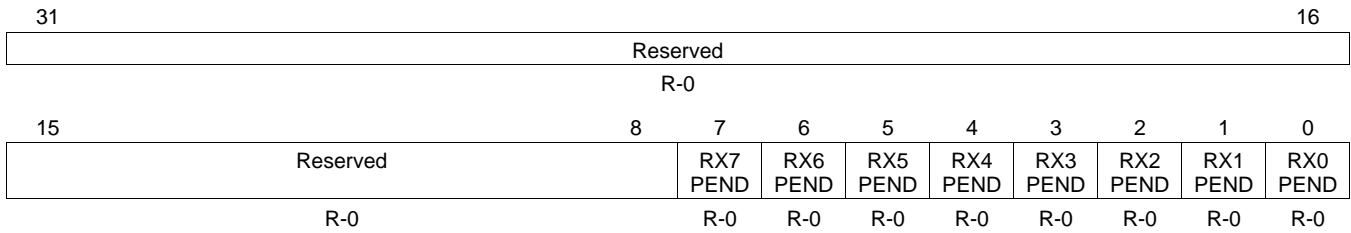
Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7PEND		RX7PEND raw interrupt read (before mask)
6	RX6PEND		RX6PEND raw interrupt read (before mask)
5	RX5PEND		RX5PEND raw interrupt read (before mask)
4	RX4PEND		RX4PEND raw interrupt read (before mask)
3	RX3PEND		RX3PEND raw interrupt read (before mask)
2	RX2PEND		RX2PEND raw interrupt read (before mask)
1	RX1PEND		RX1PEND raw interrupt read (before mask)
0	RX0PEND		RX0PEND raw interrupt read (before mask)



### 5.14 Receive Interrupt Status (Masked) Register (RXINTSTATMASKED)

The receive interrupt status (Masked) register (RXINTSTATMASKED) is shown in [Figure 42](#) and described in [Table 42](#).

**Figure 42. Receive Interrupt Status (Masked) Register (RXINTSTATMASKED)**



LEGEND: R = Read only; -n = value after reset

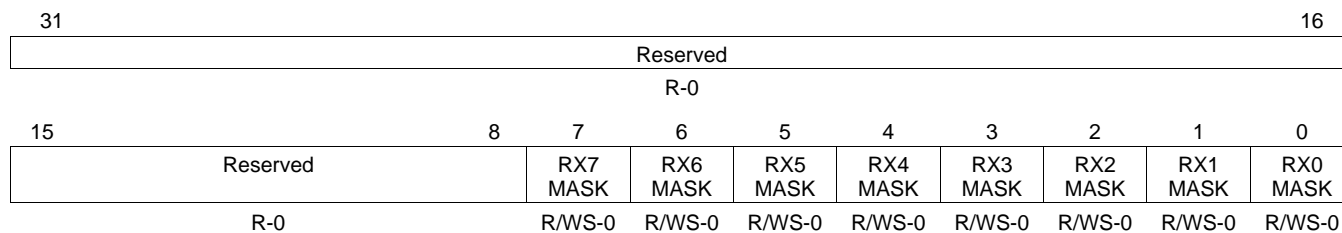
**Table 42. Receive Interrupt Status (Masked) Register (RXINTSTATMASKED) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7PEND		RX7PEND masked interrupt read
6	RX6PEND		RX6PEND masked interrupt read
5	RX5PEND		RX5PEND masked interrupt read
4	RX4PEND		RX4PEND masked interrupt read
3	RX3PEND		RX3PEND masked interrupt read
2	RX2PEND		RX2PEND masked interrupt read
1	RX1PEND		RX1PEND masked interrupt read
0	RX0PEND		RX0PEND masked interrupt read

### 5.15 Receive Interrupt Mask Set Register (RXINTMASKSET)

The receive interrupt mask set register (RXINTMASKSET) is shown in [Figure 43](#) and described in [Table 43](#).

**Figure 43. Receive Interrupt Mask Set Register (RXINTMASKSET)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

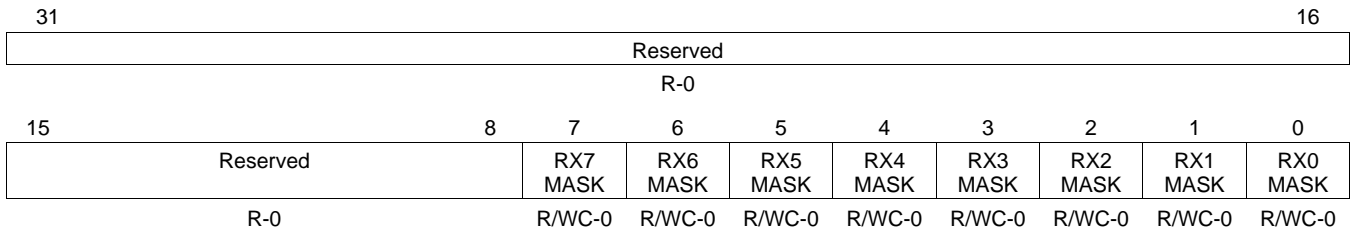
**Table 43. Receive Interrupt Mask Set Register (RXINTMASKSET) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7MASK		Receive channel 7 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
6	RX6MASK		Receive channel 6 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
5	RX5MASK		Receive channel 5 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
4	RX4MASK		Receive channel 4 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
3	RX3MASK		Receive channel 3 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
2	RX2MASK		Receive channel 2 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
1	RX1MASK		Receive channel 1 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
0	RX0MASK		Receive channel 0 mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.

### 5.16 Receive Interrupt Mask Clear Register (RXINTMASKCLEAR)

The receive interrupt mask clear register (RXINTMASKCLEAR) is shown in [Figure 44](#) and described in [Table 44](#).

**Figure 44. Receive Interrupt Mask Clear Register (RXINTMASKCLEAR)**



LEGEND: R = Read only; R/W = Read/Write; R/WC = Read/Write 1 to clear; -n = value after reset

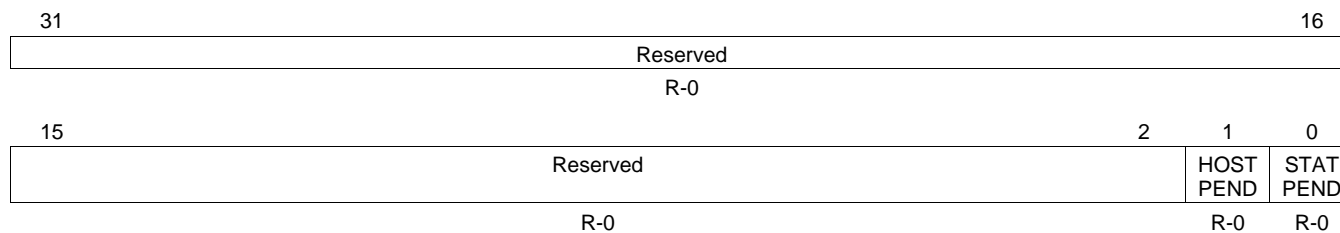
**Table 44. Receive Interrupt Mask Clear Register (RXINTMASKCLEAR) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RX7MASK		Receive channel 7 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
6	RX6MASK		Receive channel 6 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
5	RX5MASK		Receive channel 5 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
4	RX4MASK		Receive channel 4 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
3	RX3MASK		Receive channel 3 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
2	RX2MASK		Receive channel 2 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
1	RX1MASK		Receive channel 1 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
0	RX0MASK		Receive channel 0 mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.

### 5.17 MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW)

The MAC interrupt status (unmasked) register (MACINTSTATRAW) is shown in [Figure 45](#) and described in [Table 45](#).

**Figure 45. MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW)**



LEGEND: R = Read only; -n = value after reset

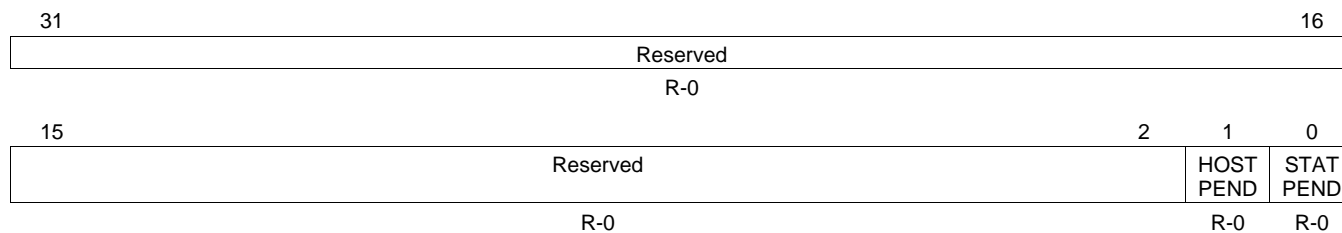
**Table 45. MAC Interrupt Status (Unmasked) Register (MACINTSTATRAW) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTPEND		Host pending interrupt (HOSTPEND); raw interrupt read (before mask)
0	STATPEND		Statistics pending interrupt (STATPEND); raw interrupt read (before mask)

### 5.18 MAC Interrupt Status (Masked) Register (MACINTSTATMASKED)

The MAC interrupt status (masked) register (MACINTSTATMASKED) is shown in [Figure 46](#) and described in [Table 46](#).

**Figure 46. MAC Interrupt Status (Masked) Register (MACINTSTATMASKED)**



LEGEND: R/W = R = Read only; -n = value after reset

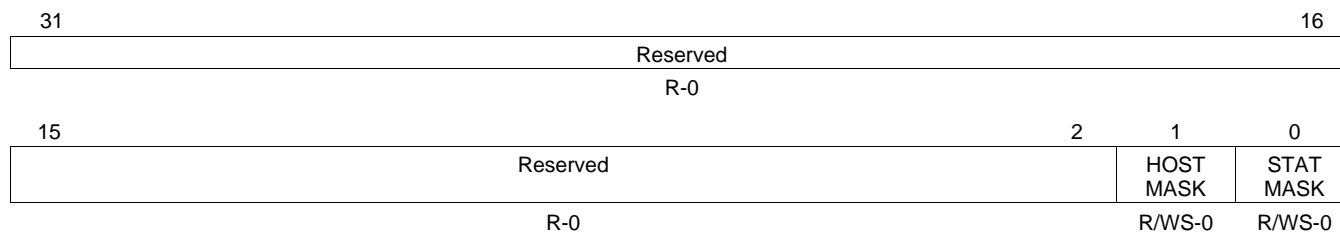
**Table 46. MAC Interrupt Status (Masked) Register (MACINTSTATMASKED) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTPEND		Host pending interrupt (HOSTPEND); masked interrupt read
0	STATPEND		Statistics pending interrupt (STATPEND); masked interrupt read

### 5.19 MAC Interrupt Mask Set Register (MACINTMASKSET)

The MAC interrupt mask set register (MACINTMASKSET) is shown in [Figure 47](#) and described in [Table 47](#).

**Figure 47. MAC Interrupt Mask Set Register (MACINTMASKSET)**



LEGEND: R = Read only; R/W = Read/Write; R/WS = Read/Write 1 to set; -n = value after reset

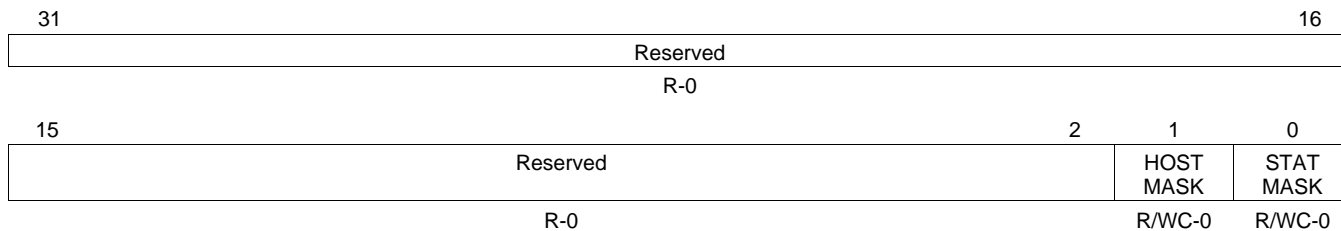
**Table 47. MAC Interrupt Mask Set Register (MACINTMASKSET) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTMASK		Host error interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.
0	STATMASK		Statistics interrupt mask set bit. Write 1 to enable interrupt, a write of 0 has no effect.

## 5.20 MAC Interrupt Mask Clear Register (MACINTMASKCLEAR)

The MAC interrupt mask clear register (MACINTMASKCLEAR) is shown in [Figure 48](#) and described in [Table 48](#).

**Figure 48. MAC Interrupt Mask Clear Register (MACINTMASKCLEAR)**



LEGEND: R = Read only; R/W = Read/Write; R/WC = Read/Write 1 to clear; -n = value after reset

**Table 48. MAC Interrupt Mask Clear Register (MACINTMASKCLEAR) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	HOSTMASK		Host error interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.
0	STATMASK		Statistics interrupt mask clear bit. Write 1 to disable interrupt, a write of 0 has no effect.

## 5.21 Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE)

The receive multicast/broadcast/promiscuous channel enable register (RXMBPENABLE) is shown in Figure 49 and described in Table 49.

**Figure 49. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE)**

31	30	29	28	27	25	24
Reserved	RXPASSCRC	RXQOSEN	RXNOCHAIN	Reserved		RXCMFEN
R-0	R/W-0	R/W-0	R/W-0	R-0		R/W-0
23	22	21	20	19	18	16
RXCSFEN	RXCEFEN	RXCAFEN	Reserved		RXBPROMCH	
R/W-0	R/W-0	R/W-0	R-0		R/W-0	
15	14	13	12	11	10	8
Reserved		RXBROADEN	Reserved		RXBROADCH	
R-0		R/W-0	R-0		R/W-0	
7	6	5	4	3	2	0
Reserved		RXMULTEN	Reserved		RXMULTCH	
R-0		R/W-0	R-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 49. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions**

Bit	Field	Value	Description
31	Reserved	0	Reserved
30	RXPASSCRC	0	Pass receive CRC enable bit
		0	Received CRC is discarded for all channels and is not included in the buffer descriptor packet length field
		1	Received CRC is transferred to memory for all channels and is included in the buffer descriptor packet length
29	RXQOSEN	0	Receive quality of service enable bit
		0	Receive QOS is disabled
		1	Receive QOS is enabled
28	RXNOCHAIN	0	Receive no buffer chaining bit
		0	Received frames can span multiple buffers
		1	Receive DMA controller transfers each frame into a single buffer regardless of the frame or buffer size. All remaining frame data after the first buffer is discarded. The buffer descriptor buffer length field will contain the entire frame byte count (up to 65535 bytes).
27-25	Reserved	0	Reserved
24	RXCMFEN	0	Receive copy MAC control frames enable bit. Enables MAC control frames to be transferred to memory. MAC control frames are normally acted upon (if enabled), but not copied to memory. MAC control frames that are pause frames will be acted upon if enabled in MACCONTROL, regardless of the value of RXCMFEN. Frames transferred to memory due to RXCMFEN will have the CONTROL bit set in their EOP buffer descriptor.
		0	MAC control frames are filtered (but acted upon if enabled)
		1	MAC control frames are transferred to memory
23	RXCSFEN	0	Receive copy short frames enable bit. Enables frames or fragments shorter than 64 bytes to be copied to memory. Frames transferred to memory due to RXCSFEN will have the FRAGMENT or UNDERSIZE bit set in their EOP buffer descriptor. Fragments are short frames that contain CRC/align/code errors and undersized are short frames without errors.
		0	Short frames are filtered
		1	Short frames are transferred to memory



**Table 49. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions (continued)**

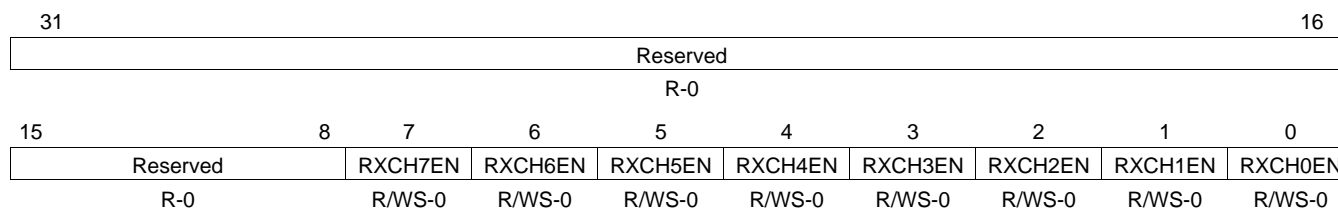
Bit	Field	Value	Description
22	RXCEFEN	0 1	Receive copy error frames enable bit. Enables frames containing errors to be transferred to memory. The appropriate error bit will be set in the frame EOP buffer descriptor. Frames containing errors are filtered Frames containing errors are transferred to memory
21	RXCAFEN	0 1	Receive copy all frames enable bit. Enables frames that do not address match (includes multicast frames that do not hash match) to be transferred to the promiscuous channel selected by RXPROMCH bits. Such frames will be marked with the NOMATCH bit in their EOP buffer descriptor. Frames that do not address match are filtered Frames that do not address match are transferred to the promiscuous channel selected by RXPROMCH bits
20-19	Reserved	0	Reserved
18-16	RXPROMCH	0-3h 0 1h 2h 3h 4h 5h 6h 7h	Receive promiscuous channel select Select channel 0 to receive promiscuous frames Select channel 1 to receive promiscuous frames Select channel 2 to receive promiscuous frames Select channel 3 to receive promiscuous frames Select channel 4 to receive promiscuous frames Select channel 5 to receive promiscuous frames Select channel 6 to receive promiscuous frames Select channel 7 to receive promiscuous frames
15-14	Reserved	0	Reserved
13	RXBROADEN	0 1	Receive broadcast enable. Enable received broadcast frames to be copied to the channel selected by RXBROADCH bits. Broadcast frames are filtered Broadcast frames are copied to the channel selected by RXBROADCH bits
12-11	Reserved	0	Reserved
10-8	RXBROADCH	0-3h 0 1h 2h 3h 4h 5h 6h 7h	Receive broadcast channel select Select channel 0 to receive broadcast frames Select channel 1 to receive broadcast frames Select channel 2 to receive promiscuous frames Select channel 3 to receive broadcast frames Select channel 4 to receive broadcast frames Select channel 5 to receive broadcast frames Select channel 6 to receive broadcast frames Select channel 7 to receive broadcast frames
7-6	Reserved	0	Reserved
5	RXMULTEN	0 1	RX multicast enable. Enable received hash matching multicast frames to be copied to the channel selected by RXMULTCH bits. Multicast frames are filtered Multicast frames are copied to the channel selected by RXMULTCH bits
4-3	Reserved	0	Reserved

**Table 49. Receive Multicast/Broadcast/Promiscuous Channel Enable Register (RXMBPENABLE) Field Descriptions (continued)**

Bit	Field	Value	Description
2-0	RXMULTCH	0-3h	Receive multicast channel select
		0	Select channel 0 to receive multicast frames
		1h	Select channel 1 to receive multicast frames
		2h	Select channel 2 to receive promiscuous frames
		3h	Select channel 3 to receive multicast frames
		4h	Select channel 4 to receive multicast frames
		5h	Select channel 5 to receive multicast frames
		6h	Select channel 6 to receive multicast frames
		7h	Select channel 7 to receive multicast frames

## 5.22 Receive Unicast Enable Set Register (RXUNICASTSET)

The receive unicast enable set register (RXUNICASTSET) is shown in [Figure 50](#) and described in [Table 50](#).

**Figure 50. Receive Unicast Enable Set Register (RXUNICASTSET)**


LEGEND: R = Read only; R/W = Read/Write; R/WS = Read/Write 1 to set; -n = value after reset

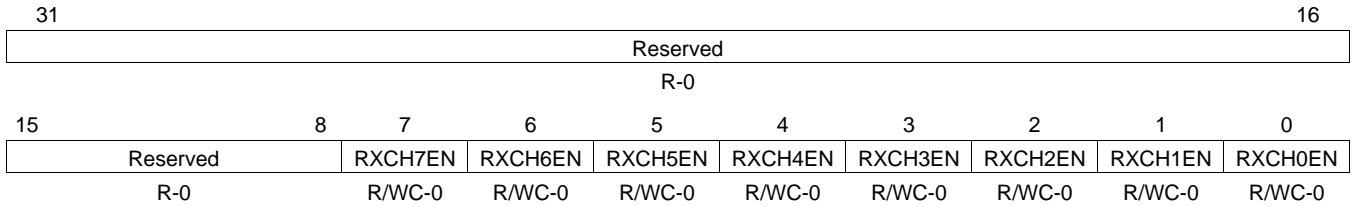
**Table 50. Receive Unicast Enable Set Register (RXUNICASTSET) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RXCH7EN		Receive channel 7 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
6	RXCH6EN		Receive channel 6 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
5	RXCH5EN		Receive channel 5 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
4	RXCH4EN		Receive channel 4 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
3	RXCH3EN		Receive channel 3 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
2	RXCH2EN		Receive channel 2 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
1	RXCH1EN		Receive channel 1 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.
0	RXCH0EN		Receive channel 0 unicast enable set bit. Write 1 to set the enable, a write of 0 has no effect. May be read.

### 5.23 Receive Unicast Clear Register (RXUNICASTCLEAR)

The receive unicast clear register (RXUNICASTCLEAR) is shown in [Figure 51](#) and described in [Table 51](#).

**Figure 51. Receive Unicast Clear Register (RXUNICASTCLEAR)**



LEGEND: R = Read only; R/W = Read/Write; R/WC = Read/Write 1 to clear; -n = value after reset

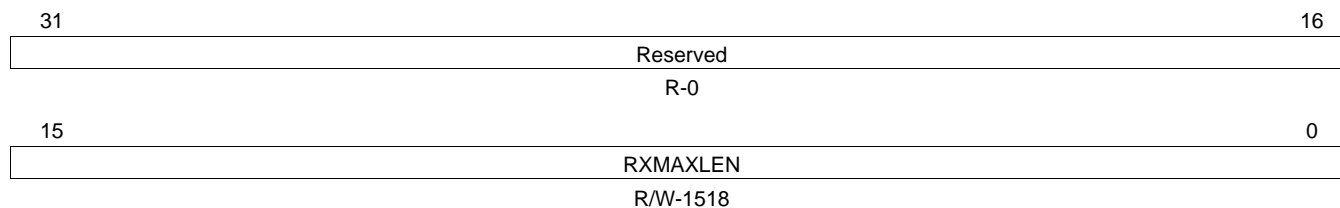
**Table 51. Receive Unicast Clear Register (RXUNICASTCLEAR) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7	RXCH7EN		Receive channel 7 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
6	RXCH6EN		Receive channel 6 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
5	RXCH5EN		Receive channel 5 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
4	RXCH4EN		Receive channel 4 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
3	RXCH3EN		Receive channel 3 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
2	RXCH2EN		Receive channel 2 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
1	RXCH1EN		Receive channel 1 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.
0	RXCH0EN		Receive channel 0 unicast enable clear bit. Write 1 to clear the enable, a write of 0 has no effect.

## 5.24 Receive Maximum Length Register (RXMAXLEN)

The receive maximum length register (RXMAXLEN) is shown in [Figure 52](#) and described in [Table 52](#).

**Figure 52. Receive Maximum Length Register (RXMAXLEN)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

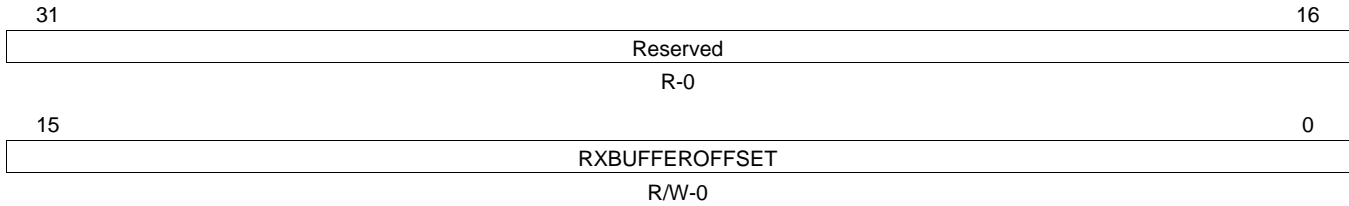
**Table 52. Receive Maximum Length Register (RXMAXLEN) Field Descriptions**

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	RXMAXLEN		Receive maximum frame length. These bits determine the maximum length of a received frame. The reset value is 5EEh (1518). Frames with byte counts greater than RXMAXLEN are long frames. Long frames with no errors are oversized frames. Long frames with CRC, code, or alignment error are jabber frames.

### 5.25 Receive Buffer Offset Register (RXBUFFEROFFSET)

The receive buffer offset register (RXBUFFEROFFSET) is shown in [Figure 53](#) and described in [Table 53](#).

**Figure 53. Receive Buffer Offset Register (RXBUFFEROFFSET)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

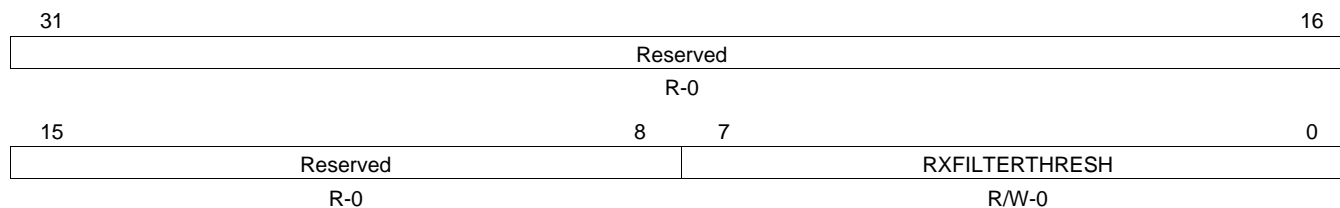
**Table 53. Receive Buffer Offset Register (RXBUFFEROFFSET) Field Descriptions**

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	RXBUFFER OFFSET		Receive buffer offset value. These bits are written by the EMAC into each frame SOP buffer descriptor Buffer Offset field. The frame data begins after the RXBUFFEROFFSET value of bytes. A value of 0 indicates that there are no unused bytes at the beginning of the data and that valid data begins on the first byte of the buffer. A value of Fh (15) indicates that the first 15 bytes of the buffer are to be ignored by the EMAC and that valid buffer data starts on byte 16 of the buffer. This value is used for all channels.

## 5.26 Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH)

The receive filter low priority frame threshold register (RXFILTERLOWTHRESH) is shown in [Figure 54](#) and described in [Table 54](#).

**Figure 54. Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

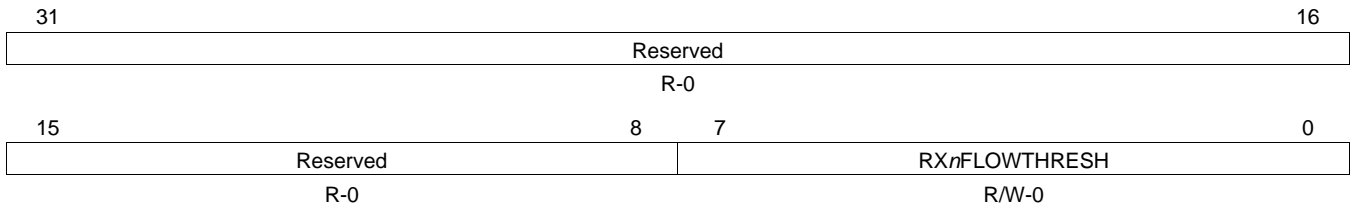
**Table 54. Receive Filter Low Priority Frame Threshold Register (RXFILTERLOWTHRESH) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	RXFILTERTHRESH		Receive filter low threshold. These bits contain the free buffer count threshold value for filtering low priority incoming frames. This field should remain zero, if no filtering is desired.

### 5.27 Receive Channel 0-7 Flow Control Threshold Register (RXnFLOWTHRESH)

The receive channel 0-7 flow control threshold register (RXnFLOWTHRESH) is shown in [Figure 55](#) and described in [Table 55](#).

**Figure 55. Receive Channel *n* Flow Control Threshold Register (RXnFLOWTHRESH)**



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

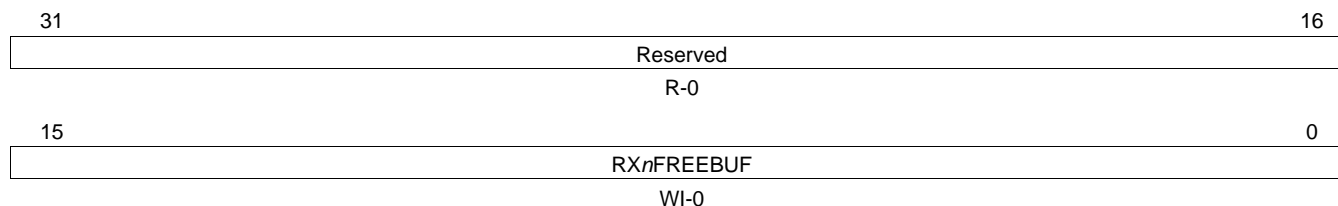
**Table 55. Receive Channel *n* Flow Control Threshold Register (RXnFLOWTHRESH) Field Descriptions**

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	RXnFLOW THRESH		Receive flow threshold. These bits contain the threshold value for issuing flow control on incoming frames for channel <i>n</i> (when enabled).

## 5.28 Receive Channel 0-7 Free Buffer Count Register (RXnFREEBUFFER)

The receive channel 0-7 free buffer count register (RXnFREEBUFFER) is shown in [Figure 56](#) and described in [Table 56](#).

**Figure 56. Receive Channel  $n$  Free Buffer Count Register (RXnFREEBUFFER)**



tLEGEND: R = Read only; R/W = Read/Write; WI = Write to increment; - $n$  = value after reset

**Table 56. Receive Channel  $n$  Free Buffer Count Register (RXnFREEBUFFER) Field Descriptions**

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	RXnFREEBUF		Receive free buffer count. These bits contain the count of free buffers available. The RXFILTERTHRESH value is compared with this field to determine if low priority frames should be filtered. The RXnFLOWTHRESH value is compared with this field to determine if receive flow control should be issued against incoming packets (if enabled). This is a write-to-increment field. This field rolls over to zero on overflow. If hardware flow control or QOS is used, the host must initialize this field to the number of available buffers (one register per channel). The EMAC decrements (by the number of buffers in the received frame) the associated channel register for each received frame. The host must write this field with the number of buffers that have been freed due to host processing.



## 5.29 MAC Control Register (MACCONTROL)

The MAC control register (MACCONTROL) is shown in [Figure 57](#) and described in [Table 57](#).

**Figure 57. MAC Control Register (MACCONTROL)**

31								24									
Reserved																	
R-0																	
23								19				18		17		16	
Reserved								RGMIIEN		GIGFORCE		RMIIDUPLEX-MODE					
R-0								R/W-0		R/W-0		R/W-0					
15		14		13		12		11		10		9		8			
RMIISPEED	RXOFFLENBLOCK	RXOWNERSHIP	RXFIFO FLOWEN	CMDIDLE		Reserved		TXPTYPE		Reserved							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R-0		R/W-0		R-0							
7		6		5		4		3		2		1		0			
GIG	TXPACE	GMIEN	TXFLOWEN	RXBUFFERFLOWEN		Reserved		LOOPBACK		FULLDUPLEX							
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R-0		R/W-0		R/W-0							

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 57. MAC Control Register (MACCONTROL) Field Descriptions**

Bit	Field	Value	Description
31-19	Reserved	0	Reserved
18	RGMIIEN	0 1	RGMII enable bit. Enables the full duplex and gigabit mode to be selected from the RGMII FULLDUPLEX and RGMII GIG input signals and not from the FULLDUPLEX and GIG bits contained in this register. This bit is directly connected to the RXINBAND input on the RGMII module. 0 The RGMII interface is in forced link mode. The duplexity is determined by the FULLDUPLEX bit, and the speed is determined by the GIG bit. The speed is either 1 Gbps or 100 Mbps; 10 Mbps is not supported in forced link mode. 1 The RGMII interface requires and uses the in-band signals coming in on its receive.
17	GIGFORCE		Gigabit force mode. This bit is used to force the EMAC into gigabit mode if the input MTCLK signal has been stopped by the PHY.
16	RMIIDUPLEXMODE	0 1	Duplex mode for the RMII interface. 0 The RMII operates in half duplex mode 1 The RMII operates in full duplex mode. This mode is not supported in the C645x devices.
15	RMIISPEED	0 1	Operating speed for the RMII interface 0 The RMII operates at 2.5 MHz (10Mbps mode) 1 The RMII operates at 25 MHz (100 Mbps mode)
14	RXOFFLENBLOCK	0 1	Receive offset / length word write block 0 Do not block the DMA writes to the receive buffer descriptor offset/buffer length word 1 Block all EMAC DMA controller writes to the receive buffer descriptor offset/buffer length words during packet processing. When this bit is set, the EMAC will never write the third word to any receive buffer descriptor.
13	RXOWNERSHIP	0 1	Receive ownership write bit value 0 The EMAC writes the Receive ownership bit to zero at the end of packet processing 1 The EMAC writes the Receive ownership bit to one at the end of packet processing. If you do not use the ownership mechanism, you can set this mode to preclude the necessity of software having to set this bit each time the buffer descriptor is used.
12	RXFIFO FLOWEN	0	Receive FIFO flow control enable 0 Receive flow control disabled. For full-duplex mode, no outgoing pause frames are sent.

**Table 57. MAC Control Register (MACCONTROL) Field Descriptions (continued)**

Bit	Field	Value	Description
		1	Receive flow control enabled. For full-duplex mode, outgoing pause frames are sent when receive FIFO flow control is triggered.
11	CMDIDLE	0	Command Idle bit
		1	Idle not commanded
		1	Idle commanded (read IDLE in the MACSTATUS register)
10	Reserved	0	Reserved
9	TXPTYPE	0	Transmit queue priority type
		0	The queue uses a round-robin scheme to select the next channel for transmission
		1	The queue uses a fixed-priority (channel 7 highest priority) scheme to select the next channel for transmission
8	Reserved	0	Reserved
7	GIG	0	Gigabit mode
		0	Gigabit mode is disabled; 10/100 mode is in operation
		1	Gigabit mode is enabled (full-duplex only)
6	TXPACE	0	Transmit pacing enable bit
		0	Transmit pacing is disabled
		1	Transmit pacing is enabled
5	GMIEN	0	GMI enable bit
		0	GMI RX and TX are held in reset
		1	GMI RX and TX are enabled for receive and transmit
4	TXFLOWEN	0	Transmit flow control enable bit. This bit determines if incoming pause frames are acted upon in full-duplex mode. Incoming pause frames are not acted upon in half-duplex mode, regardless of this bit setting. The RXMBPENABLE bits determine whether or not received pause frames are transferred to memory.
		0	Transmit flow control is disabled. Full-duplex mode: incoming pause frames are not acted upon.
		1	Transmit flow control is enabled. Full-duplex mode: incoming pause frames are acted upon.
3	RXBUFFER FLOWEN	0	Receive buffer flow control enable bit
		0	Receive flow control is disabled. Half-duplex mode: no flow control generated collisions are sent. Full-duplex mode: no outgoing pause frames are sent.
		1	Receive flow control is enabled. Half-duplex mode: collisions are initiated when receive buffer flow control is triggered. Full-duplex mode: outgoing pause frames are sent when receive flow control is triggered.
2	Reserved	0	Reserved
1	LOOPBACK	0	Loopback mode. The loopback mode forces internal full-duplex mode regardless of the FULLDUPLEX bit. The loopback bit should be changed only when GMIEN bit is de-asserted.
		0	Loopback mode is disabled
		1	Loopback mode is enabled
0	FULLDUPLEX	0	Full duplex mode. The gigabit mode forces full duplex mode regardless of whether the FULLDUPLEX bit is set or not.
		0	Half-duplex mode is enabled
		1	Full-duplex mode is enabled

### 5.30 MAC Status Register (MACSTATUS)

The MAC status register (MACSTATUS) is shown in [Figure 58](#) and described in [Table 58](#).

**Figure 58. MAC Status Register (MACSTATUS)**

31	30					24
IDLE		Reserved				
R-0		R-0				
23	20	19	18	16		
TXERRCODE		Reserved	TXERRCH			
R-0		R-0	R-0			
15	12	11	10	8		
RXERRCODE		Reserved	RXERRCH			
R-0		R-0	R-0			
7	5	4	3	2	1	0
Reserved		RDMIIGIG	RGMIIFULLDUPLEX	RXQOSACT	RXFLOWACT	TXFLOWACT
R-0		R-x	R-x	R-0	R-0	R-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 58. MAC Status Register (MACSTATUS) Field Descriptions**

Bit	Field	Value	Description
31	IDLE	0 1	EMAC idle bit. This bit is set to 0 at reset; one clock after reset it goes to 1. The EMAC is not idle The EMAC is in the idle state
30-24	Reserved	0	Reserved
23-20	TXERRCODE	0 1h 2h 3h 4h 5h 6h	Transmit host error code. These bits indicate that EMAC detected transmit DMA related host errors. The host should read this field after a host error interrupt (HOSTPEND) to determine the error. Host error interrupts require hardware reset in order to recover. A zero packet length is an error, but it is not detected. No error SOP error; the buffer is the first buffer in a packet, but the SOP bit is not set in software. Ownership bit not set in SOP buffer Zero next buffer descriptor pointer without EOP Zero buffer pointer Zero buffer length Packet length error (sum of buffers < packet length)
19	Reserved	0	Reserved
18-16	TXERRCH	0-3h 0 1h 2h 3h 4h 5h 6h 7h	Transmit host error channel. These bits indicate which transmit channel the host error occurred on. This field is cleared to 0 on a host read. The host error occurred on transmit channel 0 The host error occurred on transmit channel 1 The host error occurred on transmit channel 2 The host error occurred on transmit channel 3 The host error occurred on transmit channel 4 The host error occurred on transmit channel 5 The host error occurred on transmit channel 6 The host error occurred on transmit channel 7

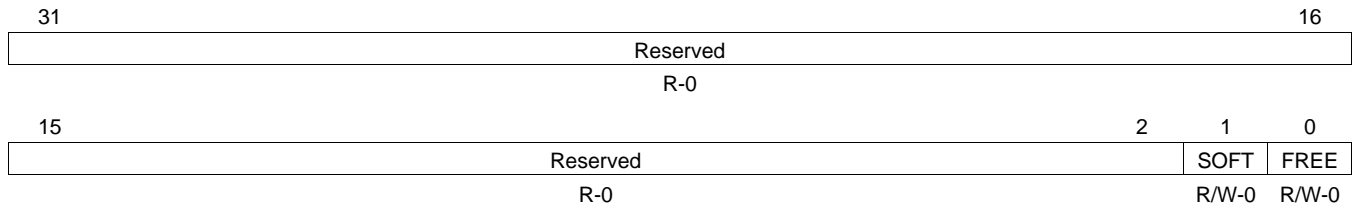
**Table 58. MAC Status Register (MACSTATUS) Field Descriptions (continued)**

Bit	Field	Value	Description
15-12	RXERRCODE	0 2h 4h	Receive host error code. These bits indicate that EMAC detected receive DMA related host errors. The host should read this field after a host error interrupt (HOSTPEND) to determine the error. Host error interrupts require hardware reset in order to recover. No error Ownership bit not set in SOP buffer Zero buffer pointer
11	Reserved	0	Reserved
10-8	RXERRCH	0-3h 0 1h 2h 3h 4h 5h 6h 7h	Receive host error channel. These bits indicate which receive channel the host error occurred on. This field is cleared to 0 on a host read. The host error occurred on receive channel 0 The host error occurred on receive channel 1 The host error occurred on receive channel 2 The host error occurred on receive channel 3 The host error occurred on receive channel 4 The host error occurred on receive channel 5 The host error occurred on receive channel 6 The host error occurred on receive channel 7
7-5	Reserved	0	Reserved
4	RGMIIGIG		RGMIIG gigabit. This is the value of RGMIIGIG input.
3	RGMIIFULL DUPLEX		RGMIIF full-duplex. This is the value of RGMIIFULLDUPLICATE input.
2	RXQOSACT	0 1	Receive Quality of Service (QOS) active bit. When asserted, indicates that receive quality of service is enabled and that at least one channel freebuffer count (RXnFREEBUFFER) is less than or equal to the RXFILTERLOWTHRESH value. Receive quality of service is disabled. Receive quality of service is enabled
1	RXFLOWACT	0 1	Receive flow control active bit. When asserted, indicates that at least one channel freebuffer count (RXnFREEBUFFER) is less than or equal to the channel's corresponding RXnFILTERTHRESH value. Receive flow control is inactive Receive flow control is active
0	TXFLOWACT	0 1	Transmit flow control active bit. When asserted, this bit indicates that the pause time period is being observed for a received pause frame. No new transmissions will begin while this bit is asserted except for the transmission of pause frames. Any transmission in progress when this bit is asserted will complete. Transmit flow control is inactive Transmit flow control is active

### 5.31 Emulation Control Register (EMCONTROL)

The emulation control register (EMCONTROL) is shown in [Figure 59](#) and described in [Table 59](#).

**Figure 59. Emulation Control Register (EMCONTROL)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

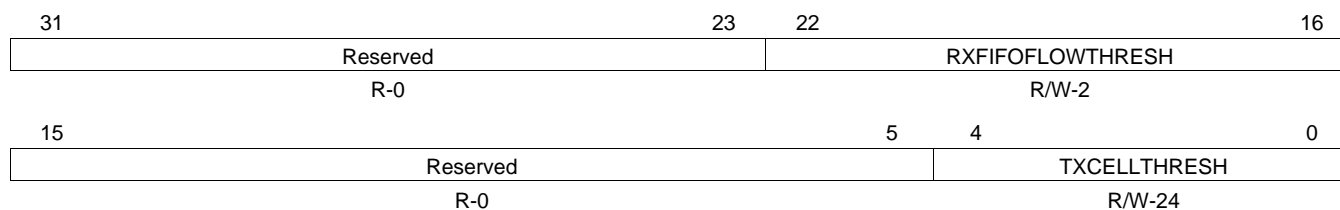
**Table 59. Emulation Control Register (EMCONTROL) Field Descriptions**

Bit	Field	Value	Description
31-2	Reserved	0	Reserved
1	SOFT		Emulation soft bit
0	FREE		Emulation free bit

### 5.32 FIFO Control Register (FIFOCONTROL)

The FIFO control register (FIFOCONTROL) is shown in [Figure 60](#) and described in [Table 60](#).

**Figure 60. FIFO Control Register (FIFOCONTROL)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

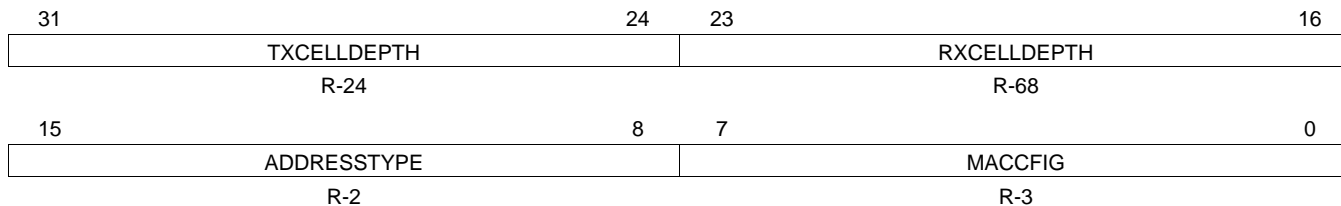
**Table 60. FIFO Control Register (FIFOCONTROL) Field Descriptions**

Bit	Field	Value	Description
31-23	Reserved	0	Reserved
22-16	RXFIFOFLOWTHRESH		Receive FIFO flow control threshold. Occupancy of the receive FIFO when Receive FIFO flow control is triggered (if enabled). The default value is 0x2 which means that receive FIFO flow control will be triggered when the occupancy of the FIFO reaches two cells.
15-5	Reserved	0	Reserved
4-0	TXCELLTHRESH		Transmit FIFO cell threshold. Indicates the number of 64-byte packet cells required to be in the transmit FIFO before the packet transfer is initiated. Packets with fewer cells will be initiated when the complete packet is contained in the FIFO. This value must be greater than or equal to 2 and less than or equal to 24.

### 5.33 MAC Configuration Register (MACCONFIG)

The MAC configuration register (MACCONFIG) is shown in [Figure 61](#) and described in [Table 61](#).

**Figure 61. MAC Configuration Register (MACCONFIG)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

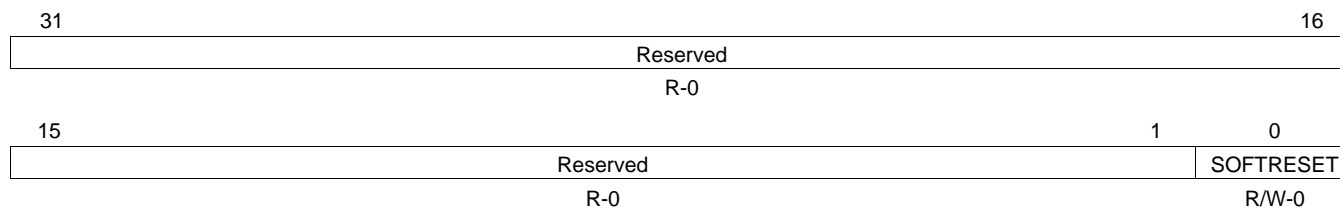
**Table 61. MAC Configuration Register (MACCONFIG) Field Descriptions**

Bit	Field	Value	Description
31-24	TXCELLDEPTH		Transmit cell depth. These bit indicates the number of cells in the transmit FIFO.
23-16	RXCELLDEPTH		Receive cell depth. These bits indicate the number of cells in the receive FIFO.
15-8	ADDRESSTYPE		Address type
7-0	MACCFIG		MAC configuration value

### 5.34 Soft Reset Register (SOFTRESET)

The Soft Reset Register (SOFTRESET) is shown in [Figure 62](#) and described in [Table 62](#).

**Figure 62. Soft Reset Register (SOFTRESET)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 62. Soft Reset Register (SOFTRESET) Field Descriptions**

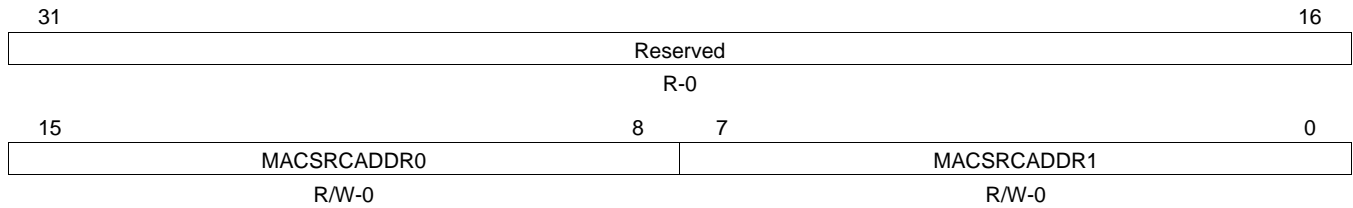
Bit	Field	Value	Description
31-1	Reserved	0	Reserved
0	SOFTRESET	0	Software reset. Writing a one to this bit causes the EMAC logic to be reset. Software reset occurs when the receive and transmit DMA controllers are in an idle state to avoid locking up the Configuration bus. After writing a one to this bit, it may be polled to determine if the reset has occurred. If a one is read, the reset has not yet occurred. If a zero is read then reset has occurred.
		1	A software reset has occurred



### 5.35 MAC Source Address Low Bytes Register (MACSRCADDRLO)

The MAC source address low bytes register (MACSRCADDRLO) is shown in [Figure 63](#) and described in [Table 63](#).

**Figure 63. MAC Source Address Low Bytes Register (MACSRCADDRLO)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

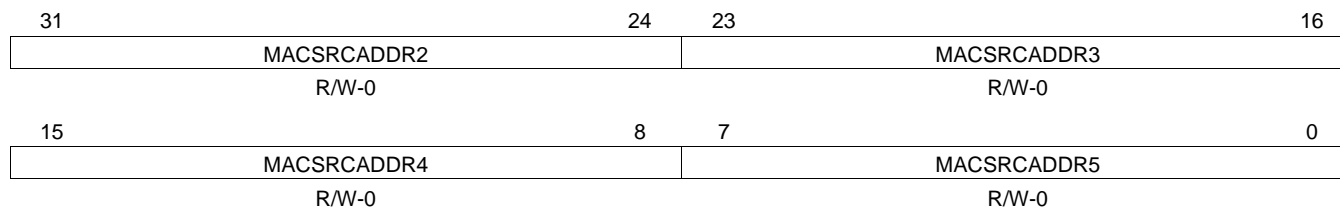
**Table 63. MAC Source Address Low Bytes Register (MACSRCADDRLO) Field Descriptions**

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-8	MACSRCADDR0		MAC source address lower 8 bits (byte 0)
7-0	MACSRCADDR1		MAC source address bits 15-8 (byte 1)

### 5.36 MAC Source Address High Bytes Register (MACSRCADDRHI)

The MAC Source Address High Bytes Register (MACSRCADDRHI) is shown in [Figure 64](#) and described in [Table 64](#).

**Figure 64. MAC Source Address High Bytes Register (MACSRCADDRHI)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 64. MAC Source Address High Bytes Register (MACSRCADDRHI) Field Descriptions**

Bit	Field	Value	Description
31-24	MACSRCADDR2		MAC source address bits 23-16 (byte 2)
23-16	MACSRCADDR3		MAC source address bits 31-24 (byte 3)
15-8	MACSRCADDR4		MAC source address bits 39-32 (byte 4)
7-0	MACSRCADDR5		MAC source address bits 47-40 (byte 5)

### 5.37 MAC Hash Address Register 1 (MACHASH1)

The MAC hash registers allow group addressed frames to be accepted on the basis of a hash function of the address. The hash function creates a 6-bit data value (hash\_fun) from the 48-bit destination address (DA) as follows:

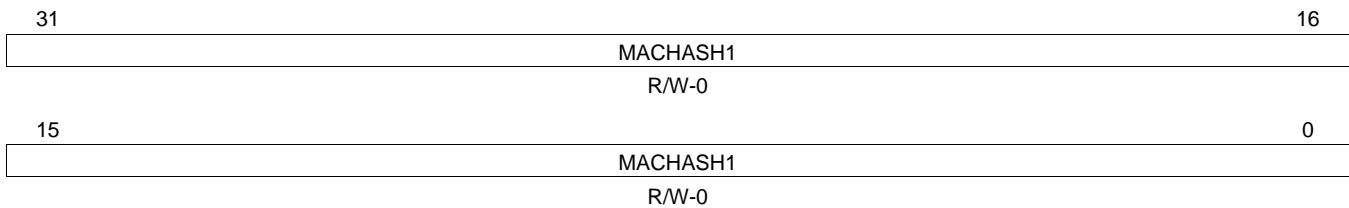
```

Hash_fun(0)=DA(0) XOR DA(6) XOR DA(12) XOR DA(18) XOR DA(24) XOR DA(30) XOR DA(36) XOR DA(42);
Hash_fun(1)=DA(1) XOR DA(7) XOR DA(13) XOR DA(19) XOR DA(25) XOR DA(31) XOR DA(37) XOR DA(43);
Hash_fun(2)=DA(2) XOR DA(8) XOR DA(14) XOR DA(20) XOR DA(26) XOR DA(32) XOR DA(38) XOR DA(44);
Hash_fun(3)=DA(3) XOR DA(9) XOR DA(15) XOR DA(21) XOR DA(27) XOR DA(33) XOR DA(39) XOR DA(45);
Hash_fun(4)=DA(4) XOR DA(10) XOR DA(16) XOR DA(22) XOR DA(28) XOR DA(34) XOR DA(40) XOR DA(46);
Hash_fun(5)=DA(5) XOR DA(11) XOR DA(17) XOR DA(23) XOR DA(29) XOR DA(35) XOR DA(41) XOR DA(47);
  
```

This function is used as an offset into a 64-bit hash table stored in MACHASH1 and MACHASH2 that indicates whether a particular address should be accepted or not.

The MAC hash address register 1 (MACHASH1) is shown in [Figure 65](#) and described in [Table 65](#).

**Figure 65. MAC Hash Address Register 1 (MACHASH1)**



LEGEND: R/W = Read/Write; -n = value after reset

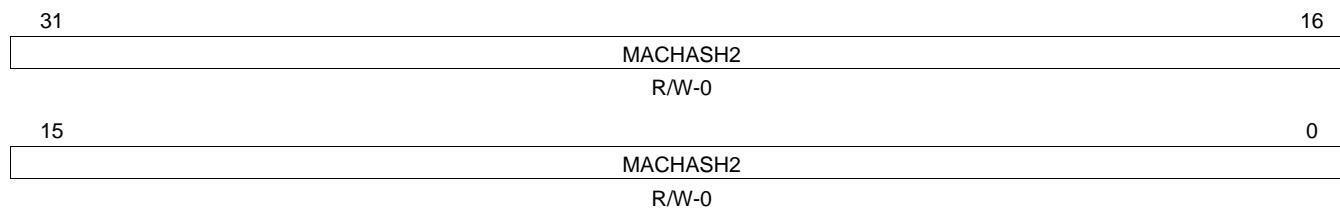
**Table 65. MAC Hash Address Register 1 (MACHASH1) Field Descriptions**

Bit	Field	Value	Description
31-0	MACHASH1		Least-significant 32 bits of the hash table corresponding to hash values 0 to 31. If a hash table bit is set, then a group address that hashes to that bit index is accepted.

### 5.38 MAC Hash Address Register 2 (MACHASH2)

The MAC hash address register 2 (MACHASH2) is shown in [Figure 66](#) and described in [Table 66](#).

**Figure 66. MAC Hash Address Register 2 (MACHASH2)**



LEGEND: R/W = Read/Write; -n = value after reset

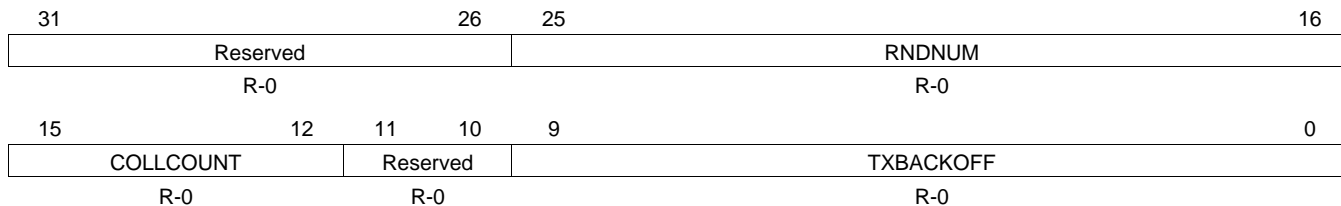
**Table 66. MAC Hash Address Register 2 (MACHASH2) Field Descriptions**

Bit	Field	Value	Description
31-0	MACHASH2		Most-significant 32 bits of the hash table corresponding to hash values 32 to 63. If a hash table bit is set, then a group address that hashes to that bit index is accepted.

### 5.39 Back Off Test Register (BOFFTEST)

The back off test register (BOFFTEST) is shown in [Figure 67](#) and described in [Table 67](#).

**Figure 67. Back Off Random Number Generator Test Register (BOFFTEST)**



LEGEND: R = Read only; -n = value after reset

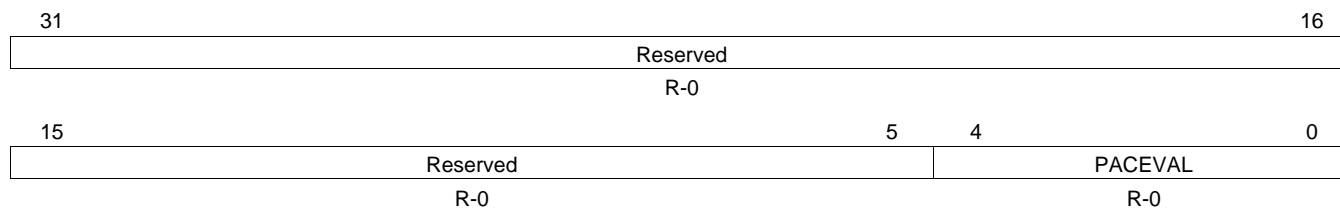
**Table 67. Back Off Test Register (BOFFTEST) Field Descriptions**

Bit	Field	Value	Description
31-26	Reserved	0	Reserved
25-16	RNDNUM		Back off random number generator. This field allows the Back off Random Number Generator to be read. Reading this field returns the generator's current value. The value is reset to zero and begins counting on the clock after the de-assertion of reset.
15-12	COLLCOUNT		Collision count. These bits indicate the number of collisions the current frame has experienced.
11-10	Reserved	0	Reserved
9-0	TXBACKOFF		Back off count. This field allows the current value of the back off counter to be observed for test purposes. This field is loaded automatically according to the back off algorithm, and is decremented by one for each slot time after the collision.

### 5.40 Transmit Pacing Algorithm Test Register (TPACETEST)

The Transmit Pacing Algorithm Test Register (TPACETEST) is shown in [Figure 68](#) and described in [Table 68](#).

**Figure 68. Transmit Pacing Algorithm Test Register (TPACETEST)**



LEGEND: R = Read only; -n = value after reset

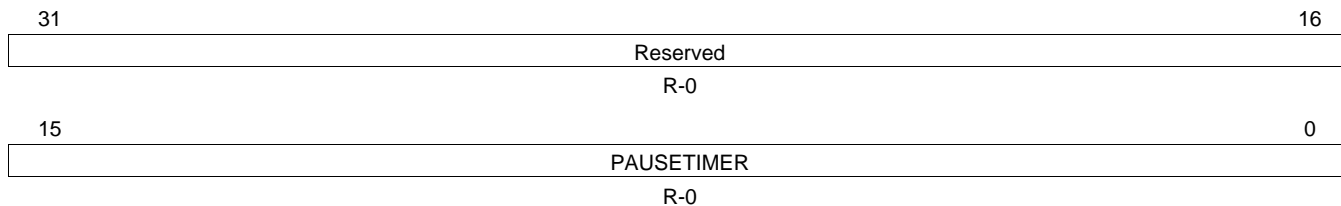
**Table 68. Transmit Pacing Algorithm Test Register (TPACETEST) Field Descriptions**

Bit	Field	Value	Description
31-5	Reserved	0	Reserved
4-0	PACEVAL		Pacing register current value. A nonzero value in this field indicates that transmit pacing is active. A transmit frame collision or deferral causes PACEVAL to be loaded with 1Fh (31); good frame transmissions (with no collisions or deferrals) cause PACEVAL to be decremented down to 0. When PACEVAL is nonzero, the transmitter delays four Inter Packet Gaps between new frame transmissions after each successfully transmitted frame that had no deferrals or collisions. If a transmit frame is deferred or suffers a collision, the IPG time is not stretched to four times the normal value. Transmit pacing helps reduce capture effects, which improves overall network bandwidth.

### 5.41 Receive Pause Timer Register (RXPAUSE)

The receive pause timer register (RXPAUSE) is shown in [Figure 69](#) and described in [Table 69](#).

**Figure 69. Receive Pause Timer Register (RXPAUSE)**



LEGEND: R = Read only; -n = value after reset

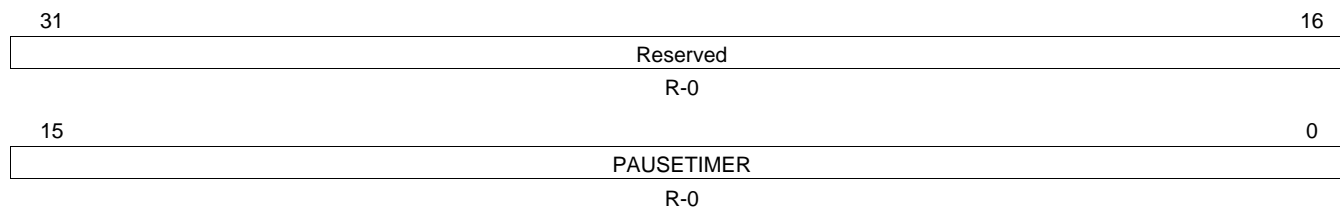
**Table 69. Receive Pause Timer Register (RXPAUSE) Field Descriptions**

Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	PAUSETIMER		Receive pause timer value. These bits allow the contents of the receive pause timer to be observed. The receive pause timer is loaded with FF00h when the EMAC sends an outgoing pause frame (with pause time of FFFFh). The receive pause timer is decremented at slot time intervals. If the receive pause timer decrements to 0, then another outgoing pause frame is sent and the load/decrement process is repeated.

## 5.42 Transmit Pause Timer Register (TXPAUSE)

The Transmit Pause Timer Register (TXPAUSE) is shown in [Figure 70](#) and described in [Table 70](#).

**Figure 70. Transmit Pause Timer Register (TXPAUSE)**



LEGEND: R = Read only; -n = value after reset

**Table 70. Transmit Pause Timer Register (TXPAUSE) Field Descriptions**

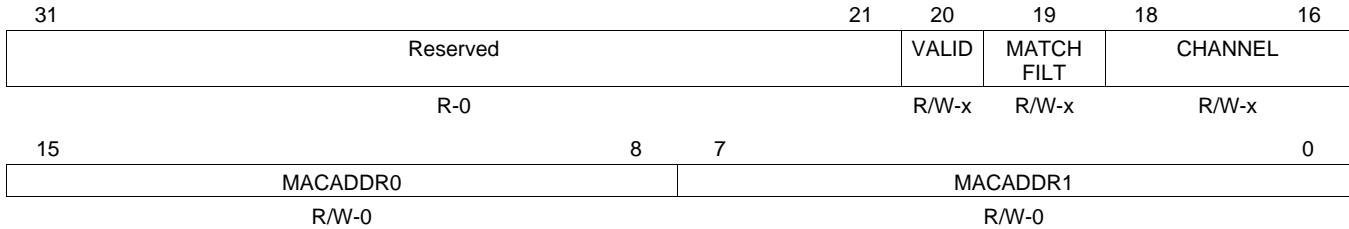
Bit	Field	Value	Description
31-16	Reserved	0	Reserved
15-0	PAUSETIMER		Transmit pause timer value. These bits allow the contents of the transmit pause timer to be observed. The transmit pause timer is loaded by a received (incoming) pause frame, and then decremented at slot time intervals down to 0 at which time EMAC transmit frames are again enabled.



### 5.43 MAC Address Low Bytes Register (MACADDRLO)

The MAC address low bytes register (MACADDRLO) is shown in [Figure 71](#) and described in [Table 71](#).

**Figure 71. MAC Address Low Bytes Register (MACADDRLO)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

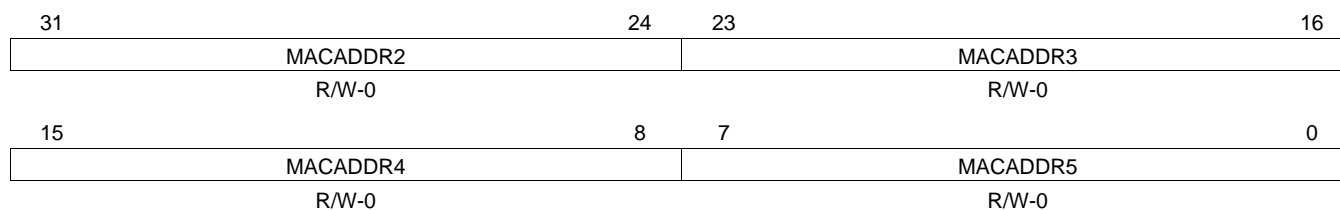
**Table 71. MAC Address Low Bytes Register (MACADDRLO) Field Descriptions**

Bit	Field	Value	Description
31-21	Reserved	0	Reserved
20	VALID	0	Address location is not valid and will not be used in determining whether or not an incoming packet matches or is filtered
		1	Address location is valid and will be used in determining whether or not an incoming packet matches or is filtered
19	MATCHFILT	0	Match or filter bit. The address will be used (if VALID is set) to determine if the incoming packet address should be filtered
		1	The address will be used (if VALID is set) to determine if the incoming packet address is a match
18-16	CHANNEL		Channel bit; determines which receive channel a valid address match will be transferred to. The channel is a don't care if the MATCHFILT bit is cleared to zero.
15-8	MACADDR0		MAC address lower 8 bits (byte 0)
7-0	MACADDR1		MAC address bits 15-8 (byte 1)

### 5.44 MAC Address High Bytes Register (MACADDRHI)

The MAC address high bytes register (MACADDRHI) is shown in [Figure 72](#) and described in [Table 72](#).

**Figure 72. MAC Address High Bytes Register (MACADDRHI)**



LEGEND: R/W = Read/Write; -n = value after reset

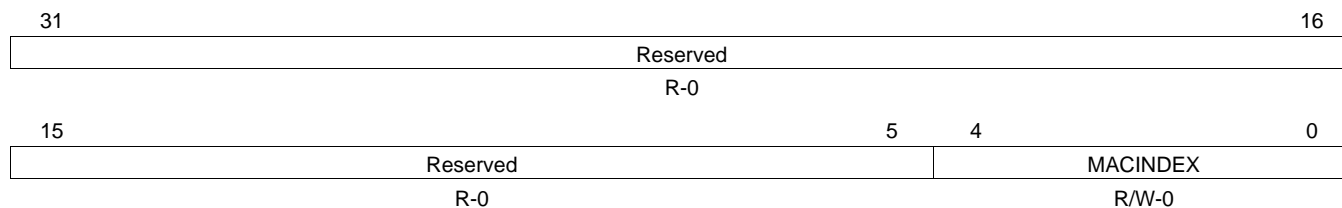
**Table 72. MAC Address High Bytes Register (MACADDRHI) Field Descriptions**

Bit	Field	Value	Description
31-24	MACADDR2		MAC source address bits 23-16 (byte 2)
23-16	MACADDR3		MAC source address bits 31-24 (byte 3)
15-8	MACADDR4		MAC source address bits 39-32 (byte 4)
7-0	MACADDR5		MAC source address bits 47-40 (byte 5)

### 5.45 MAC Index Register (MACINDEX)

The MAC index register (MACINDEX) is shown in [Figure 73](#) and described in [Table 73](#).

**Figure 73. MAC Index Register (MACINDEX)**



LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

**Table 73. MAC Index Register (MACINDEX) Field Descriptions**

Bit	Field	Value	Description
31-5	Reserved	0	Reserved
4-0	MACINDEX		MAC address index. The host must write the index into the RX ADDR RAM in the MACINDEX field, followed by the upper 32-bits of address, followed by the lower 16-bits of address (with control bits). The 53-bit indexed ram location is written when the low location is written. All 32 address RAM locations must be initialized prior to enabling packet reception.

### 5.46 Transmit Channel 0-7 DMA Head Descriptor Pointer Register (TXnHDP)

The transmit channel 0-7 DMA head descriptor pointer register (TXnHDP) is shown in [Figure 74](#) and described in [Table 74](#).

**Figure 74. Transmit Channel  $n$  DMA Head Descriptor Pointer Register (TXnHDP)**



LEGEND: R/W = Read/Write; -n = value after reset

**Table 74. Transmit Channel  $n$  DMA Head Descriptor Pointer Register (TXnHDP) Field Descriptions**

Bit	Field	Value	Description
31-0	TXnHDP		Transmit channel $n$ DMA Head Descriptor pointer. Writing a transmit DMA buffer descriptor address to a head pointer location initiates transmit DMA operations in the queue for the selected channel. Writing to these locations when they are nonzero is an error (except at reset). Host software must initialize these locations to zero on reset.

### 5.47 Receive Channel 0-7 DMA Head Descriptor Pointer Register (RXnHDP)

The receive channel 0-7 DMA head descriptor pointer register (RXnHDP) is shown in [Figure 75](#) and described in [Table 75](#).

**Figure 75. Receive Channel *n* DMA Head Descriptor Pointer Register (RXnHDP)**



LEGEND: R/W = Read/Write; -*n* = value after reset

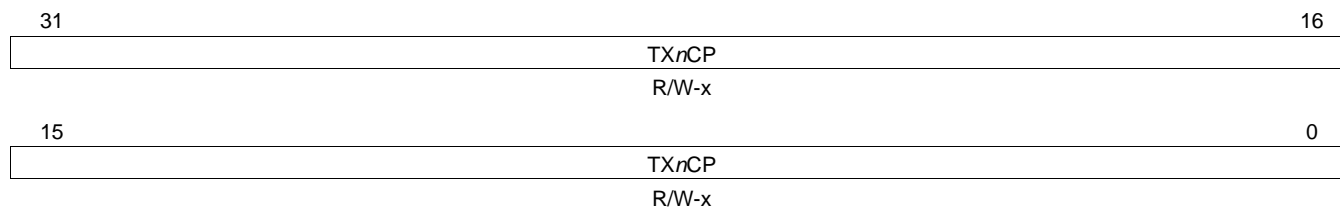
**Table 75. Receive Channel *n* DMA Head Descriptor Pointer Register (RXnHDP) Field Descriptions**

Bit	Field	Value	Description
31-0	RXnHDP		Receive channel <i>n</i> DMA Head Descriptor pointer. Writing a receive DMA buffer descriptor address to this location allows receive DMA operations in the selected channel when a channel frame is received. Writing to these locations when they are nonzero is an error (except at reset). Host software must initialize these locations to zero on reset.

### 5.48 Transmit Channel 0-7 Completion Pointer Register (TX<sub>n</sub>CP)

The Transmit Channel 0-7 Completion Pointer Register (TX<sub>n</sub>CP) is shown in [Figure 76](#) and described in [Table 76](#).

**Figure 76. Transmit Channel *n* Completion Pointer Register (TX<sub>n</sub>CP)**



LEGEND: R/W = Read/Write; -*n* = value after reset

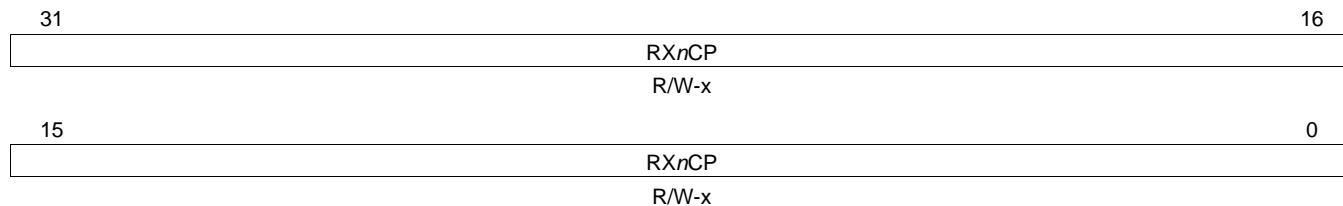
**Table 76. Transmit Channel *n* Completion Pointer Register (TX<sub>n</sub>CP) Field Descriptions**

Bit	Field	Value	Description
31-0	TX <sub>n</sub> CP		Transmit channel <i>n</i> completion pointer register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The EMAC uses the value written to determine if the interrupt should be de-asserted.

### 5.49 Receive Channel 0-7 Completion Pointer Register (RXnCP)

The receive channel 0-7 completion pointer register (RXnCP) is shown in [Figure 77](#) and described in [Table 77](#).

**Figure 77. Receive Channel *n* Completion Pointer Register (RXnCP)**



LEGEND: R/W = Read/Write; -*n* = value after reset

**Table 77. Receive Channel *n* Completion Pointer Register (RXnCP) Field Descriptions**

Bit	Field	Value	Description
31-0	RXnCP		Receive channel <i>n</i> completion pointer register is written by the host with the buffer descriptor address for the last buffer processed by the host during interrupt processing. The EMAC uses the value written to determine if the interrupt should be de-asserted.

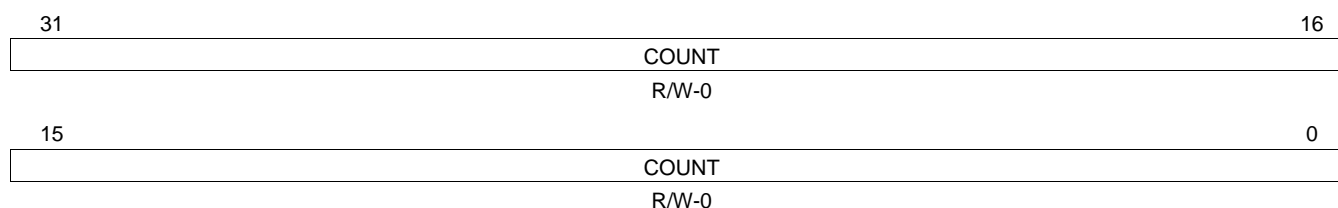
## 5.50 Network Statistics Registers

The EMAC has a set of statistics that record events associated with frame traffic. The statistics values are cleared to zero 38 clocks after the rising edge of reset. When the GMIIEN bit in the MACCONTROL register is set, all statistics registers are write-to-decrement. The value written is subtracted from the register value with the result stored in the register. If a value greater than the statistics value is written, then zero is written to the register (writing FFFF FFFFh clears a statistics location). When the GMIIEN bit is cleared, all statistics registers are read/write (normal write direct, so writing 0000 0000h clears a statistics location). All write accesses must be 32-bit accesses.

The statistics interrupt (STATPEND) is issued, if enabled, when any statistics value is greater than or equal to 8000 0000h. The statistics interrupt is removed by writing to decrement any statistics value greater than 8000 0000h. The statistics are mapped into internal memory space and are 32-bits wide. All statistics rollover from FFFF FFFFh to 0000 0000h.

The network statistics register is shown in [Figure 78](#) and described in [Table 78](#).

**Figure 78. Statistics Register**



LEGEND: R/W = Read/Write; -n = value after reset

**Table 78. Statistics Register Field Descriptions**

Bit	Field	Value	Description
31-0	COUNT		Count

### 5.50.1 Good Receive Frames Register (RXGOODFRAMES)

The total number of good frames received on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

### 5.50.2 Broadcast Receive Frames Register (RXBCASTFRAMES)

The total number of good broadcast frames received on the EMAC. A good broadcast frame is defined as having all of the following:

- Any data or MAC control frame that was destined for address FF-FF-FF-FF-FF-FFh only
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.



### 5.50.3 Multicast Receive Frames Register (RXMCASTFRAMES)

The total number of good multicast frames received on the EMAC. A good multicast frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any multicast address other than FF-FF-FF-FF-FF-FFh
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for alignment/code/CRC error definitions. Overruns have no effect on this statistic.

### 5.50.4 Pause Receive Frames Register (RXPAUSEFRAMES)

The total number of IEEE 802.3X pause frames received by the EMAC (whether acted upon or not). A pause frame is defined as having all of the following:

- Contained any unicast, broadcast, or multicast address
- Contained the length/type field value 88.08h and the opcode 0001h
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error
- Pause-frames had been enabled on the EMAC (TXFLOWEN bit is set in MACCONTROL).

The EMAC could have been in either half-duplex or full-duplex mode. See [Section 2.5.5](#) for alignment/code/CRC error definitions. Overruns have no effect on this statistic.

### 5.50.5 Receive CRC Errors Register (RXCRCERRORS)

The total number of frames received on the EMAC that experienced a CRC error. A frame with CRC errors is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no alignment or code error
- Had a CRC error. A CRC error is defined as having all of the following:
  - A frame containing an even number of nibbles
  - Fails the frame check sequence test

See [Section 2.5.5](#) for definitions of alignment and code errors. Overruns have no effect on this statistic.

### 5.50.6 Receive Alignment/Code Errors Register (RXALIGNCODEERRORS)

The total number of frames received on the EMAC that experienced an alignment error or code error. Such a frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had either an alignment error or a code error
  - An alignment error is defined as having all of the following:
    - A frame containing an odd number of nibbles
    - Fails the frame check sequence test, if the final nibble is ignored
  - A code error is defined as a frame that has been discarded because the EMACs MRXER pin is driven with a one for at least one bit-time's duration at any point during the frame's reception.

Overruns have no effect on this statistic.

CRC alignment or code errors can be calculated by summing receive alignment errors, receive code errors, and receive CRC errors.

**5.50.7 Receive Oversized Frames Register (RXOVERSIZED)**

The total number of oversized frames received on the EMAC. An oversized frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was greater than RXMAXLEN in bytes
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code/CRC errors. Overruns have no effect on this statistic.

**5.50.8 Receive Jabber Frames Register (RXJABBER)**

The total number of jabber frames received on the EMAC. A jabber frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was greater than RXMAXLEN bytes long
- Had a CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

**5.50.9 Receive Undersized Frames Register (RXUNDERSIZED)**

The total number of undersized frames received on the EMAC. An undersized frame is defined as having all of the following:

- Was any data frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was less than 64 bytes long
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

**5.50.10 Receive Frame Fragments Register (RXFRAGMENTS)**

The total number of frame fragments received on the EMAC. A frame fragment is defined as having all of the following:

- Any data frame (address matching does not matter)
- Was less than 64 bytes long
- Had a CRC error, alignment error, or code error
- Was not the result of a collision caused by half duplex, collision based flow control

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

### 5.50.11 Filtered Receive Frames Register (RXFILTERED)

The total number of frames received on the EMAC that the EMAC address matching process indicated should be discarded. Such a frame is defined as having all of the following:

- Was any data frame (not MAC control frame) destined for any unicast, broadcast, or multicast address
- Did not experience any CRC error, alignment error, code error
- The address matching process decided that the frame should be discarded (filtered) because it did not match the unicast, broadcast, or multicast address, and it did not match due to promiscuous mode.

To determine the number of receive frames discarded by the EMAC for any reason, sum the following statistics (promiscuous mode disabled):

- Receive fragments
- Receive undersized frames
- Receive CRC errors
- Receive alignment/code errors
- Receive jabbers
- Receive overruns
- Receive filtered frames

This may not be an exact count because the receive overruns statistic is independent of the other statistics, so if an overrun occurs at the same time as one of the other discard reasons, then the above sum double-counts that frame.

### 5.50.12 Receive QOS Filtered Frames Register (RXQOSFILTERED)

The total number of frames received on the EMAC that were filtered due to receive quality of service (QOS) filtering. Such a frame is defined as having all of the following:

- Any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- The frame destination channel flow control threshold register (RX $n$ FLOWTHRESH) value was greater than or equal to the channel's corresponding free buffer register (RX $n$ FREEBUFFER) value
- Was of length 64 to RXMAXLEN
- RXQOSEN bit is set in RXMBPENABLE
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

### 5.50.13 Receive Octet Frames Register (RXOCTETS)

The total number of bytes in all good frames received on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of length 64 to RXMAXLEN bytes inclusive
- Had no CRC error, alignment error, or code error

See [Section 2.5.5](#) for definitions of alignment, code, and CRC errors. Overruns have no effect on this statistic.

### 5.50.14 Good Transmit Frames Register (TXGOODFRAMES)

The total number of good frames transmitted on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Was any length
- Had no late or excessive collisions, no carrier loss, and no underrun

**5.50.15 Broadcast Transmit Frames Register (TXBCASTFRAMES)**

The total number of good broadcast frames transmitted on the EMAC. A good broadcast frame is defined as having all of the following:

- Any data or MAC control frame destined for address FF-FF-FF-FF-FF-FFh only
- Was of any length
- Had no late or excessive collisions, no carrier loss, and no underrun

**5.50.16 Multicast Transmit Frames Register (TXMCASTFRAMES)**

The total number of good multicast frames transmitted on the EMAC. A good multicast frame is defined as having all of the following:

- Any data or MAC control frame destined for any multicast address other than FF-FF-FF-FF-FF-FFh
- Was of any length
- Had no late or excessive collisions, no carrier loss, and no underrun

**5.50.17 Pause Transmit Frames Register (TXPAUSEFRAMES)**

The total number of IEEE 802.3X pause frames transmitted by the EMAC. Pause frames cannot underrun or contain a CRC error because they are created in the transmitting MAC, so these error conditions have no effect on this statistic. Pause frames sent by software are not included in this count. Since pause frames are only transmitted in full-duplex mode, carrier loss and collisions have no effect on this statistic.

Transmitted pause frames are always 64-byte multicast frames so appear in the multicast transmit frames register and 64 octet frames register statistics.

**5.50.18 Deferred Transmit Frames Register (TXDEFERRED)**

The total number of frames transmitted on the EMAC that first experienced deferment. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced no collisions before being successfully transmitted
- Found the medium busy when transmission was first attempted, so had to wait.

CRC errors have no effect on this statistic.

**5.50.19 Transmit Collision Frames Register (TXCOLLISION)**

The total number of times that the EMAC experienced a collision. Collisions occur under two circumstances:

- When a transmit data or MAC control frame has all of the following:
  - Was destined for any unicast, broadcast, or multicast address
  - Was any size
  - Had no carrier loss and no underrun
  - Experienced a collision. A jam sequence is sent for every non-late collision, so this statistic increments on each occasion if a frame experiences multiple collisions (and increments on late collisions).

CRC errors have no effect on this statistic.

- When the EMAC is in half-duplex mode, flow control is active, and a frame reception begins.

---

#### 5.50.20 Transmit Single Collision Frames Register (TXSINGLECOLL)

The total number of frames transmitted on the EMAC that experienced exactly one collision. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced one collision before successful transmission. The collision was not late.

CRC errors have no effect on this statistic.

#### 5.50.21 Transmit Multiple Collision Frames Register (TXMULTICOLL)

The total number of frames transmitted on the EMAC that experienced multiple collisions. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced 2 to 15 collisions before being successfully transmitted. None of the collisions were late.

CRC errors have no effect on this statistic.

#### 5.50.22 Transmit Excessive Collision Frames Register (TXEXCESSIVECOLL)

The total number of frames when transmission was abandoned due to excessive collisions. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced 16 collisions before abandoning all attempts at transmitting the frame. None of the collisions were late.

CRC errors have no effect on this statistic.

#### 5.50.23 Transmit Late Collision Frames Register (TXLATECOLL)

The total number of frames when transmission was abandoned due to a late collision. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- Had no carrier loss and no underrun
- Experienced a collision later than 512 bit-times into the transmission. There may have been up to 15 previous (non-late) collisions that had previously required the transmission to be re-attempted. The late collisions statistic dominates over the single, multiple, and excessive collisions statistics. If a late collision occurs, the frame is not counted in any of these other three statistics.

CRC errors, carrier loss, and underrun have no effect on this statistic.

#### 5.50.24 Transmit Underrun Error Register (TXUNDERRUN)

The number of frames sent by the EMAC that experienced FIFO underrun. Late collisions, CRC errors, carrier loss, and underrun have no effect on this statistic.

**5.50.25 Transmit Carrier Sense Errors Register (TXCARRIERSENSE)**

The total number of frames on the EMAC that experienced carrier loss. Such a frame is defined as having all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address
- Was any size
- The carrier sense condition was lost or never asserted when transmitting the frame (the frame is not re-transmitted)

CRC errors and underrun have no effect on this statistic.

**5.50.26 Transmit Octet Frames Register (TXOCTETS)**

The total number of bytes in all good frames transmitted on the EMAC. A good frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Was any length
- Had no late or excessive collisions, no carrier loss, and no underrun

**5.50.27 Transmit and Receive 64 Octet Frames Register (FRAME64)**

The total number of 64-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was exactly 64-bytes long. (If the frame was being transmitted and experienced carrier loss that resulted in a frame of this size being transmitted, then the frame is recorded in this statistic).

CRC errors, alignment/code errors, and overruns do not affect the recording of frames in this statistic.

**5.50.28 Transmit and Receive 65 to 127 Octet Frames Register (FRAME65T127)**

The total number of 65-byte to 127-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 65-bytes to 127-bytes long

CRC errors, alignment/code errors, under-runs, and overruns do not affect the recording of frames in this statistic.

**5.50.29 Transmit and Receive 128 to 255 Octet Frames Register (FRAME128T255)**

The total number of 128-byte to 255-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 128-bytes to 255-bytes long

CRC errors, alignment/code errors, under-runs, and overruns do not affect the recording of frames in this statistic.

---

### 5.50.30 Transmit and Receive 256 to 511 Octet Frames Register (FRAME256T511)

The total number of 256-byte to 511-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 256-bytes to 511-bytes long

CRC errors, alignment/code errors, under-runs, and overruns do not affect the recording of frames in this statistic.

### 5.50.31 Transmit and Receive 512 to 1023 Octet Frames Register (FRAME512T1023)

The total number of 512-byte to 1023-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 512-bytes to 1023-bytes long

CRC errors, alignment/code errors, and overruns do not affect the recording of frames in this statistic.

### 5.50.32 Transmit and Receive 1024 to RXMAXLEN Octet Frames Register (FRAME1024TUP)

The total number of 1024-byte to RXMAXLEN-byte frames received and transmitted on the EMAC. Such a frame is defined as having all of the following:

- Any data or MAC control frame that was destined for any unicast, broadcast, or multicast address
- Did not experience late collisions, excessive collisions, underrun, or carrier sense error
- Was 1024-bytes to RXMAXLEN-bytes long

CRC/alignment/code errors, under-runs, and overruns do not affect frame recording in this statistic.

### 5.50.33 Network Octet Frames Register (NETOCTETS)

The total number of bytes of frame data received and transmitted on the EMAC. Each frame counted has all of the following:

- Was any data or MAC control frame destined for any unicast, broadcast, or multicast address (address match does not matter)
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)

Also counted in this statistic is:

- Every byte transmitted before a carrier-loss was experienced
- Every byte transmitted before each collision was experienced (multiple retries are counted each time)
- Every byte received if the EMAC is in half-duplex mode until a jam sequence was transmitted to initiate flow control. (The jam sequence is not counted to prevent double-counting).

Error conditions such as alignment errors, CRC errors, code errors, overruns, and under-runs do not affect the recording of bytes in this statistic. The objective of this statistic is to give a reasonable indication of Ethernet utilization.

**5.50.34 Receive FIFO or DMA Start of Frame Overruns Register (RXSOFOVERRUNS)**

The total number of frames received on the EMAC that had either a FIFO or DMA start of frame (SOF) overrun. An SOF overrun frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)
- The EMAC was unable to receive it because it did not have the resources to receive it (cell FIFO full or no DMA buffer available at the start of the frame).

CRC errors, alignment errors, and code errors have no effect on this statistic.

**5.50.35 Receive FIFO or DMA Middle of Frame Overruns Register (RXMOFOVERRUNS)**

The total number of frames received on the EMAC that had either a FIFO or DMA middle of frame (MOF) overrun. An MOF overrun frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)
- The EMAC was unable to receive it because it did not have the resources to receive it (cell FIFO full or no DMA buffer available after the frame was successfully started, no SOF overrun)

CRC errors, alignment errors, and code errors have no effect on this statistic.

**5.50.36 Receive DMA Start of Frame and Middle of Frame Overruns Register (RXDMAOVERRUNS)**

The total number of frames received on the EMAC that had either a DMA start of frame (SOF) overrun or a DMA middle of frame (MOF) overrun. A receive DMA overrun frame is defined as having all of the following:

- Was any data or MAC control frame that matched a unicast, broadcast, or multicast address, or matched due to promiscuous mode
- Was of any size (including less than 64-byte and greater than RXMAXLEN-byte frames)
- The EMAC was unable to receive it because it did not have the DMA buffer resources to receive it (zero head descriptor pointer at the start or during the middle of the frame reception).

CRC errors, alignment errors, and code errors have no effect on this statistic.



## Appendix A Glossary

**Broadcast MAC Address** —A special Ethernet MAC address used to send data to all Ethernet devices on the local network. The broadcast address is FFh-FFh-FFh-FFh-FFh-FFh. The LSB of the first byte is odd, qualifying it as a group address; however, its value is reserved for broadcast. It is classified separately by the EMAC.

**Descriptor (Packet Buffer Descriptor)** —A small memory structure that describes a larger block of memory in terms of size, location, and state. Descriptors are used by the EMAC and application to describe the memory buffers that hold Ethernet data.

**Device** —In this document, device refers to the TMS320C645x processor.

**Ethernet MAC Address (MAC Address)** —A unique 6-byte address that identifies an Ethernet device on the network. In an Ethernet packet, a MAC address is used twice, first to identify the packet's destination, and second to identify the packet's sender or source. An Ethernet MAC address is normally specified in hexadecimal, using dashes to separate bytes. For example, 08h-00h-28h-32h-17h-42h.

The first three bytes normally designate the manufacturer of the device. However, when the first byte of the address is odd (LSB is 1), the address is a group address (broadcast or multicast). The second bit specifies whether the address is globally or locally administrated (not considered in this document).

**Ethernet Packet (Packet)** —An Ethernet packet is the collection of bytes that represents the data portion of a single Ethernet frame on the wire.

**Full Duplex** —Full duplex operation allows simultaneous communication between a pair of stations using point-to-point media (dedicated channel). Full duplex operation does not require that transmitters defer, nor do they monitor or react to receive activity, as there is no contention for a shared medium in this mode. Full duplex mode can only be used when all of the following are true:

- The physical medium is capable of supporting simultaneous transmission and reception without interference.
- There are exactly two stations connected with a full duplex point-to-point link. As there is no contention for use of a shared medium, the multiple access (i.e., CSMA/CD) algorithms are unnecessary.
- Both stations on the LAN are capable of, and have been configured to use, full duplex operation.

The most common configuration envisioned for full duplex operation consists of a central bridge (also known as a switch) with a dedicated LAN connecting each bridge port to a single device.

Full duplex operation constitutes a proper subset of the MAC functionality required for half duplex operation.

**Half Duplex** —In half duplex mode, the CSMA/CD media access method is the means by which two or more stations share a common transmission medium. To transmit, a station waits (defers) for a quiet period on the medium, that is, no other station is transmitting. It then sends the intended message in bit-serial form. If, after initiating a transmission, the message collides with that of another station, then each transmitting station intentionally transmits for an additional predefined period to ensure propagation of the collision throughout the system. The station remains silent for a random amount of time (back off) before attempting to transmit again.

**Host** — The host is an intelligent system resource that configures and manages each communications control module. The host is responsible for allocating memory, initializing all data structures, and responding to port (EMAC) interrupts. In this document, host refers to the TMS320C645x device.

**Jabber** — A condition wherein a station transmits for a period of time longer than the maximum permissible packet length, usually due to a fault condition.

**Jumbo Packets** — Jumbo frames are defined as those packets whose length exceeds the standard Ethernet MTU, which is 1500 kbytes. For the C64x+ devices, it is recommended not to use packets exceeding 35K in length. The PHY that you use can place additional limits on to the length of the packets that you can transfer in a system.

**Link** — The transmission path between any two instances of generic cabling.

**Multicast MAC Address** — A class of MAC address that sends a packet to potentially more than one recipient. A group address is specified by setting the LSB of the first MAC address byte. Thus, 01h-02h-03h-04h-05h-06h is a valid multicast address. Typically, an Ethernet MAC looks for only certain multicast addresses on a network to reduce traffic load. The multicast address list of acceptable packets is specified by the application.

**Physical Layer and Media Notation** — To identify different Ethernet technologies, a simple, three-field, type notation is used. The Physical Layer type used by the Ethernet is specified by these fields: <data rate in Mb/s><medium type><maximum segment length (×100m)>  
The definitions for the technologies mentioned in this guide are as follows:

**Table A-1. Physical Layer Definitions**

Term	Definition
10Base-T	IEEE 802.3 Physical Layer specification for a 10 Mb/s CSMA/CD local area network over two pairs of twisted-pair telephone wire.
100Base-T	IEEE 802.3 Physical Layer specification for a 100 Mb/s CSMA/CD local area network over two pairs of Category 5 unshielded twisted-pair (UTP) or shielded twisted-pair (STP) wire.
1000Base-T	IEEE 802.3 Physical Layer specification for a 1000 Mb/s CSMA/CD LAN using four pairs of Category 5 balanced copper cabling.
Twisted pair	A cable element that consists of two insulated conductors twisted together in a regular fashion to form a balanced transmission line.

**Port** — Ethernet device.

**Promiscuous Mode** — EMAC receives frames that do not match its address.

## Appendix B Revision History

Table B-1 lists the changes made since the previous version of this document.

**Table B-1. Document Revision History**

Reference	Additions/Modifications/Deletions
<a href="#">Section 2.1</a>	Changed <a href="#">Section 2.1</a> .
<a href="#">Section 2.1.2</a>	Changed <a href="#">Section 2.1.2</a> .
<a href="#">Section 2.15.4</a>	Changed <a href="#">Step 19</a> .
<a href="#">Figure 3</a>	Changed <a href="#">Figure 3</a> .
<a href="#">Table 3</a>	Changed <a href="#">Table 3</a> and the paragraph directly above <a href="#">Table 3</a> .
<a href="#">Figure 6</a>	Changed <a href="#">Figure 6</a> .
<a href="#">Chapter A</a>	Added the term jumbo packets.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

<b>Products</b>		<b>Applications</b>	
Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>	Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>	Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>	Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>	Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>	Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>	Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>	Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Low Power Wireless	<a href="http://www.ti.com/lpw">www.ti.com/lpw</a>	Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
		Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
		Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

Mailing Address: Texas Instruments  
Post Office Box 655303 Dallas, Texas 75265