TNETX4020
Programmer's Reference Guide

SPWU022
December 1998

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.

Preface

Read This First

About This Manual

This document contains information on the initialization, configuration, and operation of the TNETX4020 Ethernet™ switch from the software designer's perspective. Following each major section, pseudocode, C code, or a summary is included, highlighting the tasks to be performed by the software. Appendix A is a detailed description of the internal registers and statistics.

How to Use This Manual

This document contains the following chapters and appendixes:

- **Chapter 1 – System Overview**
- **Chapter 2 – Direct Input/Output (DIO) Interface**
- **Chapter 3 – System Test**
- **Chapter 4 – System Configuration**
- **Chapter 5 – System Operations**
- **Chapter 6 – Servicing Interrupts**
- **Appendix A – Internal Registers and Statistics**
- **Appendix B – Hardware-Level I/O Interface Definitions**

To aid the readability of this document, the following formatting and naming conventions have been used.

- Uppercase bold type is used for **PINS** and **SIGNALS.**
- Active-low pins and signals are identified by an overbar, e.g., **$\overline{\text{RESET}}$.**
- Initial capital bold italic type is used for ***Register*** names.
- Lowercase bold italic type is used for ***bits*** and ***fields***.
- Initial capital italic type is used for on-chip operations (*Find*).
- Numbers and number system bases:

  100 = 3-digit decimal number (base 10) equal to a decimal quantity of one hundred (100)

  0b100 = 3-digit binary number (base 2) equal to a decimal quantity of four (4)

  0x100 = 3-digit hexadecimal number (base 16) equal to a decimal quantity of two hundred fifty six (256)

Table A–6 gives the key for the fields in the registers.

Trademarks

Ethernet and Etherstat are trademarks of Xerox Corporation.

Rambus is a trademark of Rambus, Inc.

TI, ThunderSWITCH, and ThunderSWITCH II are trademarks of Texas Instruments Incorporated.

Windows95 is a trademark of Microsoft Corporation.

If You Need Assistance. . .

If you want to . . .	Do this . . .
Request more information about Texas Instruments networking products	Call the CRC† hotline: **800-336-5236**
Ask questions about product operation or report suspected problems	Call the CRC hotline **800-336-5236** or send e-mail to: **networks@ti.comj**
Request more information on XACT management boards or software	Contact: XACT, Inc. 1445 MacArthur Drive, Suite 244 Carrollton, TX 75007-4458 **972-242-6252**

† Texas Instruments Customer Response Center

Contents

1 System Overview 1-1

2 Direct Input/Output (DIO) Interface 2-1
2.1 Host Registers 2-2
2.2 Hardware Reset via the DIO 2-5
2.3 DMA Transfer 2-6
2.4 Code Examples 2-7
2.4.1 DIO Write Using DIOData Register 2-7
2.4.2 DIO Read Using DIODataInc Register 2-7
2.4.3 Low-Level Support Routines From TSMAN 2-8
2.4.4 DMA Write – Same DIO Address 2-11
2.4.5 DMA Read – Incrementing DIO Address 2-12

3 System Test 3-1
SysTest (System Test Register) @ 0x080F 3-2
3.1 Internal CAM and RAMs 3-4
3.2 Accessing External RAM (RDRAM) 3-5
RAMAddress (RAM Address Register) @ 0x0810–0x0813 3-6
RAMData (RAM Data Register) @ 0x00814 3-7
3.2.1 Code Example 3-8
3.3 Switch-Functionality Test 3-11
3.3.1 Internal Wrap Test 3-11
3.3.2 Duplex Wrap Test 3-15

4 System Configuration 4-1
4.1 Introduction 4-2
SysControl (System Control Register) @ 0x00FA–0x00FB 4-3
RAMStatus (RAM Status Register) @ 0x00E3 4-6
4.2 Port Configuration 4-7
PortxControl (Control Register) @ 0x0000–0x0003 4-8
4.2.1 Port Awareness 4-12
Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register) @ 0x0380–0x0383 4-12
4.2.2 Maximum Frame Length 4-13
4.2.3 Port Disable 4-13
4.2.4 Port Capability 4-14

4.2.5 Flow Control 4-15
FreeStackLength (Free Stack Length Register) @ 0x080C–0x080E 4-17
FlowThreshold (Flow Control Threshold Register) @ 0x00F0–0x00F2 4-18
McastLimit (Transmit Queue Multicast Limit Register) @ 0x00DC–0x00DF (DIO) 4-19
PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s)) @ 0x00EA–0x00EB 4-20
PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s)) @ 0x00EE–0x00EF 4-21
DevNode (Device Node Register) @ 0x00A4–0x00A9 4-22
4.2.6 Advertisement of Port Capability 4-23
4.2.7 Extended-Port Awareness 4-23
4.2.8 Transmit Pacing 4-23
4.2.9 PCS Function 4-23
PortxStatus (Port x Status Register) @ 0x0400–0x0403 4-24
4.3 Internal Address-Lookup Engine (IALE) Configuration 4-27
4.3.1 Automatic Address Maintenance 4-29
SysControl (System Control Register) @ 0x00FA–0x00FB 4-30
4.3.2 Aging Algorithms 4-33
AgingThreshold (Age Out Time Select Register) @ 0x0044–0x0045 4-33
4.3.3 VLAN Configuration 4-34
VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register) @ 0x0300–0x037F 4-34
VLAN0Ports–VLAN63Ports (VLAN Routing Register) @ 0x0100–0x01FF 4-35
4.3.4 Unknown Addresses 4-36
UnkUniPorts (Unknown Unicast Address Routing Register) @ 0x0060–0x0063 4-36
UnkMultiPorts (Unknown Multicast Address Routing Register) @ 0x0064–0x0067 4-37
UnkSrcPorts (Unknown Source Address Routing Register) @ 0x0068–0x006B 4-38
UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register) @ 0x006C–0x006F 4-39
4.3.5 Mirroring 4-40
UplinkPort (Uplink Routing Register) @ 0x0040 4-40
MirrorPort (Mirrored Port Select Register) @ 0x0041 4-41
4.3.6 Support for the Spanning-Tree Algorithm 4-42
NLearnPorts (Address Learning Disable Register) @ 0x0050–0x0053 4-42
TxBlockPorts (Block Transmit Register) @ 0x0054–0x0057 4-43
RxUniBlockPorts (Block Unicast Receive Register) @ 0x0058–0x005B 4-43
RxMultiBlockPorts (Block Multicast Receive Register) @ 0x005C–0x005F 4-44
4.3.7 Extended Port Awareness 4-45
XMultigroup17–XMultigroup63 (Extended Multicast Group Register) @ 0x0544–0x05FF 4-46

4.4 Cascading-Ring Topology 4-47
RingPorts (Ring Topology Enable Register) @ 0x008C 4-48
UnkSrcPorts (Unknown Source Address Routing Register) @ 0x0068–0x006B 4-49
4.4.1 Summary 4-51
4.5 Statistics Configuration 4-52
StatControl (Statistics Control Register) @ 0x00F8–0x00F9 4-52
4.6 LED Configuration 4-54
LEDControl (LED Control Register) @ 0x00F4–0x00F5 4-54

5 System Operations 5-1
5.1 Accessing PHYs 5-2
SIO (Serial Interface I/O Register) @ 0x00A1 5-2
5.1.1 PHY Reset 5-3
5.1.2 Accessing PHY's Internal Register 5-4
5.1.3 Code Examples 5-6
5.2 Accessing EEPROM 5-13
SIO (Serial Interface I/O Register) @ 0x00A1 5-13
5.2.1 Performing Reads and Writes 5-15
5.2.2 Pseudocode 5-18
5.2.3 Low-Level Support Routines From TSMAN 5-20
5.3 Network Management Port 5-29
5.3.1 Frame Format on the NM Port 5-30
5.3.2 Sending a Frame to the NM Port 5-32
NMRxControl (Network Management Receive Control Register) @ 0x0818–0x081A 5-32
NMData (Network Management Data Register) @ 0x0820 5-34
5.3.3 TSWriteFrame – Flowchart and Sample Code 5-37
5.3.4 Getting a Frame From the NM Port 5-42
NMTxControl (Network Management Transmit Control Register) @ 0x081C–0x081E 5-42
5.3.5 TSReadFrame – Flowchart and Sample Code 5-46
5.4 Internal Address-Lookup Engine (IALE) Management 5-50
5.4.1 Add Operation 5-50
AddNode (Address Register) @ 0x0458–0x045D 5-51
AddVLAN (Add Address VLAN Index Register) @ 0x045F 5-51
AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Unicast Format 5-52
AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Multicast Format 5-54
AddDelControl (Add/Delete Control Register) @ 0x045E 5-55
5.4.2 Delete Operation 5-58
AddDelControl (Add/Delete Control Register) @ 0x045E 5-59
DelNode (Delete Address Register) @ 0x046C–0x0471 5-60
DelPort (Delete Port Register) @ 0x0472 5-61
DelVLAN (Delete VLAN Index Register) @ 0x0473 5-61

5.4.3 Searching the IALE 5-64
FindControl (Search Control Register) @ 0x0446 5-64
FindNode (Address Register) @ 0x0440–0x0445D 5-65
FindVLAN (Find Address VLAN Index Register) @ 0x0447 5-65
FindPort (Search Routing Code Register) @ 0x0448–0x044B – Unicast Format 5-66
FindPort (Search Routing Code Register) @ 0x0448–0x044B – Multicast Format 5-68

6 Servicing Interrupts 6-1
Int (Interrupt Register) @ 0x0804–0x0806 6-2
6.1 Interrupts 6-5
6.1.1 Link-Change Interrupt (link) 6-5
6.1.2 New-Node Interrupt (new) 6-5
6.1.3 Missed New-Node Interrupt (newm) 6-5
6.1.4 Node Port-Change Interrupt (chng) 6-5
6.1.5 Missed-Node Port-Change Interrupt (chngm) 6-6
6.1.6 Security-Violation Interrupt (secvio) 6-6
6.1.7 Age-Out Interrupt (age) 6-6
6.1.8 Missed Age-Out Interrupt (agem) 6-6
6.1.9 Test-Interrupt Request (int) 6-6
6.1.10 Find-Completion Interrupt (fndcplt) 6-7
6.1.11 Unknown VLAN Interrupt (unkvlan) 6-7
6.1.12 Unknown VLAN Member Interrupt (unkmem) 6-8
6.1.13 Network-Management-Transmit Interrupt (nmtx) 6-8
6.1.14 Network-Management-Receive Interrupt (nmrx) 6-8
6.1.15 Address-Lookup-Table-Full Interrupt (full) 6-9

Figures

1–1 TNETX4020 Block Diagram 1-2
3–1 Internal Wrap Example 3-12
3–2 Duplex Wrap Example 3-15
4–1 Frame-Routing Algorithm 4-27
4–2 Cascading-Ring Topology 4-47
5–1 Byte Write 5-15
5–2 Page Write 5-15
5–3 Current Address Read 5-16
5–4 Random Read 5-16
5–5 Sequential Read 5-16
5–6 NM Port Block Diagram 5-29
5–7 Ethernet Frame Format 5-30
5–8 Ethernet Frame With an IEEE Std 802.1q Tag Header 5-30
5–9 IEEE Std 802.1q Tag Header 5-30
5–10 ThunderSwitch Encoding of TPID Field 5-31
5–11 TSWriteFrame Flowchart 5-39
5–12 TSReadFrame Flowchart 5-47

Tables

2–1 Host Register Address Map 2-2
2–2 Numbering of Bits and Bytes Within a 32-Bit Register 2-4
3–1 SysTest (System Test Register) 3-2
3–2 RAMAddress (RAM Address Register) 3-6
3–3 RAMData (RAM Data Register) 3-7
4–1 SysControl (System Control Register) 4-3
4–2 RAMStatus (RAM Status Register) 4-6
4–3 PortxControl (Control Register) 4-8
4–4 Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register) 4-12
4–5 Ingress 4-12
4–6 Egress 4-12
4–7 FreeStackLength (Free Stack Length Register) 4-17
4–8 FlowThreshold (Flow Control Threshold Register) 4-18
4–9 McastLimit (Transmit Queue Mcast Limit Register) 4-19
4–10 PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s)) 4-20
4–11 PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s)) 4-21
4–12 DevNode (Device Node Register) 4-22
4–13 PortxStatus (Port x Status Register) 4-24
4–14 SysControl (System Control Register) 4-30
4–15 AgingThreshold (Age Out Time Select Register) 4-33
4–16 VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register) 4-34
4–17 VLAN0Ports–VLAN63Ports (VLAN Routing Register) 4-35
4–18 UnkUniPorts (Unknown Unicast Address Routing Register) 4-36
4–19 UnkMultiPorts (Unknown Multicast Address Routing Register) 4-37
4–20 UnkSrcPorts (Unknown Source Address Routing Register) 4-38
4–21 UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register) 4-39
4–22 UplinkPort (Uplink Routing Register) 4-40
4–23 MirrorPort (Mirrored Port Select Register) 4-41
4–24 NLearnPorts (Address Learning Disable Register) 4-42
4–25 TxBlockPorts (Block Transmit Register) 4-43
4–26 RxUniBlockPorts (Block Unicast Receive Register) 4-43
4–27 RxMultiBlockPorts (Block Multicast Receive Register) 4-44
4–28 XMultigroup17–XMultigroup63 (Extended Multicast Group Register) 4-46
4–29 RingPorts (Ring Topology Enable Register) 4-48
4–30 UnkSrcPorts (Unknown Source Address Routing Register) 4-49
4–31 StatControl (Statistics Control Register) 4-52
4–32 LEDControl (LED Control Register) 4-54
5–1 SIO (Serial Interface I/O Register) 5-2
5–2 MII Read Frame Format 5-4
5–3 MII Write Frame Format 5-4
5–4 SIO (Serial Interface I/O Register) 5-13
5–5 Slave Address Format (Bits 7:0) 5-17
5–6 NMRxControl (Network Management Receive Control Register) 5-32

5–7 NMData (Network Management Data Register) 5-34
5–8 NMTxControl (Network Management Transmit Control Register) 5-43
5–9 AddNode (Address Register) 5-51
5–10 AddVLAN (Add Address VLAN Index Register) 5-51
5–11 AddPort (Add Routing Code Register) – Unicast Format (Determined by Bit 40 of Address) 5-52
5–12 AddPort (Add Routing Code Register) – Multicast Format (Determined by Bit 40 of Address) 5-54
5–13 AddDelControl (Add/Delete Control Register) 5-55
5–14 AddDelControl (Add/Delete Control Register) 5-59
5–15 DelNode (Delete Address Register) 5-60
5–16 DelPort (Delete Port Register) 5-61
5–17 DelVLAN (Delete VLAN Index Register) 5-61
5–18 FindControl (Search Control Register) 5-64
5–19 FindNode (Address Register) 5-65
5–20 AddVLAN (Add Address VLAN Index Register) 5-65
5–21 FindPort (Search Routing Code Register) – Unicast Format (Determined by Bit 40 of Address) 5-66
5–22 FindPort (Add Routing Code Register) – Multicast Format (Determined by Bit 40 of Address) 5-68
5–23 Supported Find Operations 5-70
6–1 Int (Interrupt Register) 6-2
A–1 DIO Internal Register Address Map A-2
A–2 Register Key A-11
A–3 EEPROM-Loadable Map A-11
A–4 Recurrent Field Code Definition A-12
A–5 PortxControl (Port x Control Register) A-13
A–6 UplinkPort (Uplink Routing Register) A-17
A–7 MirrorPort (Mirrored Port Select Register) A-18
A–8 UnkVLANPort (Unknown VLAN Routing Register) A-19
A–9 AgingThreshold (Age Out Time Select Register) A-20
A–10 NLearnPorts (Address Learning Disable Register) A-21
A–11 TxBlockPorts (Block Transmit Register) A-22
A–12 RxUniBlockPorts (Block Unicast Receive Register) A-23
A–13 RxMultiBlockPorts (Block Multicast Receive Register) A-24
A–14 UnkUniPorts (Unknown Unicast Address Routing Register) A-25
A–15 UnkMultiPorts (Unknown Multicast Address Routing Register) A-26
A–16 UnkSrcPorts (Unknown Source Address Routing Register) A-27
A–17 UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register) A-28
A–18 RxFilterPorts (Receive Filtering Enable Register) A-29
A–19 RingPorts (Ring Topology Enable Register) A-30
A–20 Revision (Revision Register) A-31
A–21 SIO (Serial Interface I/O Register) A-32
A–22 DevCode (Device Code Register) A-34
A–23 DevNode (Device Node Register) A-35
A–24 McastLimit (Transmit Queue Mcast Limit Register) A-36
A–25 RAMControl (RAM Control Register) A-37
A–26 RAMStatus (RAM Status Register) A-38
A–27 PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s)) A-39
A–28 PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s)) A-40
A–29 FlowThreshold (Flow Control Threshold Register) A-41

A–30 LEDControl (LED Control Register) A-42
A–31 StatControl (Statistics Control Register) A-43
A–32 SysControl (System Control Register) A-45
A–33 VLAN0Ports–VLAN63Ports (VLAN Routing Register) A-48
A–34 VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register) A-49
A–35 Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register) A-50
A–36 PortxStatus (Port x Status Register) A-51
A–37 FindNode (Search Node Address Register) A-54
A–38 FindControl (Search Control Register) A-55
A–39 FindVLAN (Search VLAN Index Register) A-56
A–40 FindPort (Search Routing Code Register) – Unicast Format A-57
A–41 FindPort (Search Routing Code Register) – Multicast Format A-59
A–42 NewNode (New Address Register) A-60
A–43 NewPort (New Address Port Register) A-61
A–44 NewVLAN Interrupt (New Address VLAN Register) A-62
A–45 NewVLAN Other Interrupts (New Address VLAN Register) A-62
A–46 AddNode (Address Register) A-63
A–47 AddDelControl (Add/Delete Control Register) A-64
A–48 AddVLAN (Add Address VLAN Index Register) A-65
A–49 AddPort (Add Routing Code Register) – Unicast Format A-66
A–50 AddPort (Add Routing Code Register) – Multicast Format A-68
A–51 AgedNode (Aged Out Address Register) A-69
A–52 AgedPort (Aged Out Address Port Assignment Register) A-70
A–53 AgedVLAN (Aged Out Address VLAN Index Register) A-71
A–54 DelNode (Delete Address Register) A-72
A–55 DelPort (Delete Port Register) A-73
A–56 DelVLAN (Delete VLAN Index Register) A-74
A–57 NumNodes (Node Count Register) A-75
A–58 AgingCounter (Age Out Time Count Register) A-76
A–59 XMultigroup17–XMultigroup63 (Extended Multicast Group Register) A-77
A–60 PCSxControl (Port x PCS Control Register) A-78
A–61 PCSxStatus (Port x PCS Status Register) A-80
A–62 PCSxANAdvert (Port x PCS Autonegotiation Advertisement Register) A-82
A–63 PCSxANLinkP (Port x PCS Autonegotiation Link Partner Ability Register) A-84
A–64 PCSxANExp (Port x PCS Autonegotiation Expansion Register) A-86
A–65 PCSxANNxt (Port x PCS Autonegotiation Next Page Transmit Register) A-87
A–66 PCSxANLinkPNxt
(Port x PCS Autonegotiation Link Partner Next Page Received Register) A-88
A–67 PCSxExtStatus (Port x PCS Extended Status Register) A-89
A–68 DMAAddress (DMA Address Register) A-90
A–69 Int (Interrupt Register) A-91
A–70 IntEnable (Interrupt Enable Register) A-94
A–71 FreeStackLength (Free Stack Length Register) A-96
A–72 SysTest (System Test Register) A-97
A–73 RAMAddress (RAM Address Register) A-98
A–74 RAMData (RAM Data Register) A-99
A–75 NMRxControl (Network Management Receive Control Register) A-100
A–76 NMTxControl (Network Management Transmit Control Register) A-102
A–77 NMData (Network Management Data Register) A-104

Appendixes

A Internal Registers and Statistics A-1

A.1 Detailed Register Map A-2

A.1.1 System and Control Registers A-2

A.1.2 Port and Network Management (NM) Port Statistics A-10

A.2 Register Description A-11

PortxControl (Port x Control Register) @ 0x0000–0x0003 . . . . . A-13

UplinkPort (Uplink Routing Register) @ 0x0040 . . . . . A-17

MirrorPort (Mirrored Port Select Register) @ 0x0041 . . . . . A-18

UnkVLANPort (Unknown VLAN Routing Register) @ 0x0042 . . . . . A-19

AgingThreshold (Age Out Time Select Register) @ 0x0044–0x0045 . . . . . A-20

NLearnPorts (Address Learning Disable Register) @ 0x0050–0x0053 . . . . . A-21

TxBlockPorts (Block Transmit Register) @ 0x0054–0x0057 . . . . . A-22

RxUniBlockPorts (Block Unicast Receive Register) @ 0x0058–0x005B A-23

RxMultiBlockPorts (Block Multicast Receive Register) @ 0x005C–0x005F A-24

UnkUniPorts (Unknown Unicast Address Routing Register) @ 0x0060–0x0063 . . . . . A-25

UnkMultiPorts (Unknown Multicast Address Routing Register) @ 0x0064–0x0067 . . . . . A-26

UnkSrcPorts (Unknown Source Address Routing Register) @ 0x0068–0x006B A-27

UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register) @ 0x006C–0x006F A-28

RxFilterPorts (Receive Filtering Enable Register) @ 0x0070–0x0073 . . . . . A-29

RingPorts (Ring Topology Enable Register) @ 0x008C A-30

Revision (Revision Register) @ 0x00A0 . . . . . A-31

SIO (Serial Interface I/O Register) @ 0x00A1 . . . . . A-32

DevCode (Device Code Register) @ 0x00A3 . . . . . A-34

DevNode (Device Node Register) @ 0x00A4–0x00A9 . . . . . A-35

McastLimit (Transmit Queue Multicast Limit Register) @ 0x00DC–0x00DF (DIO) A-36

RAMControl (RAM Control Register) @ 0x00E2 . . . . . A-37

RAMStatus (RAM Status Register) @ 0x00E3 . . . . . A-38

PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s)) @ 0x00EA–0x00EB A-39

PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s)) @ 0x00EE–0x00EF A-40

FlowThreshold (Flow Control Threshold Register) @ 0x00F0–0x00F2 . . . . . A-41

LEDControl (LED Control Register) @ 0x00F4–0x00F5 . . . . . A-42

StatControl (Statistics Control Register) @ 0x00F8–0x00F9 . . . . . A-43

SysControl (System Control Register) @ 0x00FA–0x00FB A-45

VLAN0Ports–VLAN63Ports (VLAN Routing Register) @ 0x0100–0x01FF A-48

VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register) @ 0x0300–0x037F A-49
Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register) @ 0x0380–0x0383 . . . . . A-50
PortxStatus (Port x Status Register) @ 0x0400–0x0403 . . . . . A-51
FindNode (Search Node Address Register) @ 0x0440–0x0445 . . . . . A-54
FindControl (Search Control Register) @ 0x0446 . . . . . A-55
FindVLAN (Search VLAN Index Register) @ 0x0447 . . . . . A-56
FindPort (Search Routing Code Register) @ 0x0448–0x044B – Unicast Format A-57
FindPort (Search Routing Code Register) @ 0x0448–0x044B – Multicast Format A-59
NewNode (New Address Register) @ 0x044C–0x0451 . . . . . A-60
NewPort (New Address Port Register) @ 0x0454–0x0455 . . . . . A-61
NewVLAN Interrupt (New Address VLAN Register) @ 0x0456–0x0457, unkvlan Int Definition A-62
NewVLAN Other Interrupts (New Address VLAN Register) @ 0x0456–0x0457 . . . . . A-62
AddNode (Address Register) @ 0x0458–0x045D A-63
AddDelControl (Add/Delete Control Register) @ 0x045E A-64
AddVLAN (Add Address VLAN Index Register) @ 0x045F A-65
AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Unicast Format A-66
AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Multicast Format A-68
AgedNode (Aged Out Address Register) @ 0x0464–0x0469 . . . . . A-69
AgedPort (Aged Out Address Port Assignment Register) @ 0x046A A-70
AgedVLAN (Aged Out Address VLAN Index Register) @ 0x046B A-71
DelNode (Delete Address Register) @ 0x046C–0x0471 . . . . . A-72
DelPort (Delete Port Register) @ 0x0472 . . . . . A-73
DelVLAN (Delete VLAN Index Register) @ 0x0473 . . . . . A-74
NumNodes (Node Count Register) @ 0x0474–0x0475 . . . . . A-75
AgingCounter (Age Out Time Count Register) @ 0x0476–0x0477 . . . . . A-76
XMultigroup17–XMultigroup63 (Extended Multicast Group Register) @ 0x0544–0x05FF A-77
PCSxControl (Port x PCS Control Register) @ 0x0600–0x0601, 0x0620–0x0621 . . . . . A-78
PCSxStatus (Port x PCS Status Register) @ 0x0602–0x0603, 0x0622–0x0623 . . . . . A-80
PCSxANAdvert (Port x PCS Autonegotiation Advertisement Register) @ 0x0608–0x0609, 0x0628–0x0629 . . . . . A-82
PCSxANLinkP (Port x PCS Autonegotiation Link Partner Ability Register) @ 0x060A–0x060B, 0x062A–0x062B A-84
PCSxANExp (Port x PCS Autonegotiation Expansion Register) @ 0x060C–0x060D, 0x062C–0x062D A-86
PCSxANNxt (Port x PCS Autonegotiation Next Page Transmit Register) @ 0x060E–0x060F, 0x062E–0x062F A-87
PCSxANLinkPNxt (Port x PCS Autonegotiation Link Partner Next Page Received Register) @ 0x0610–0x0611, 0x0630–0x0631 . . . . . A-88
PCSxExtStatus (Port x PCS Extended Status Register) @ 0x061E–0x061F, 0x063E–0x063F A-89

DMAAddress (DMA Address Register) @ 0x0800–0x0801 . . . . . A-90
Int (Interrupt Register) @ 0x0804–0x0806 . . . . . A-91
IntEnable (Interrupt Enable Register) @ 0x0808–0x080A A-94
FreeStackLength (Free Stack Length Register) @ 0x080C–0x080E A-96
SysTest (System Test Register) @ 0x080F A-97
RAMAddress (RAM Address Register) @ 0x0810–0x0813 . . . . . A-98
RAMData (RAM Data Register) @ 0x00814 . . . . . A-99
NMRxControl (Network Management Receive Control Register) @ 0x0818–0x081A A-100
NMTxControl (Network Management Transmit Control Register) @ 0x081C–0x081D A-102
NMData (Network Management Data Register) @ 0x0820 . . . . . A-104
A.3 Statistic Descriptions A-105

B Hardware-Level I/O Interface Definitions B-1
B.1 TSIFHW.H B-2
B.2 TSIF.H B-3

Chapter 1

System Overview

The ThunderSWITCH™ TNETX4020 is a 2-port 100-/1000-Mbit/s Ethernet™ switch with an on-chip address-lookup engine. A low-cost, high-performance, uplink for slower switches is achieved by combining the TNETX4020 with physical interfaces, high-bandwidth Rambus-based packet memory, and an external CPU capable of receiving and transmitting simple network-management-protocol (SNMP) and bridge-protocol-data-units (BPDU) (spanning tree) frames. A block diagram of the TNETX4020 is shown in Figure 1–1.

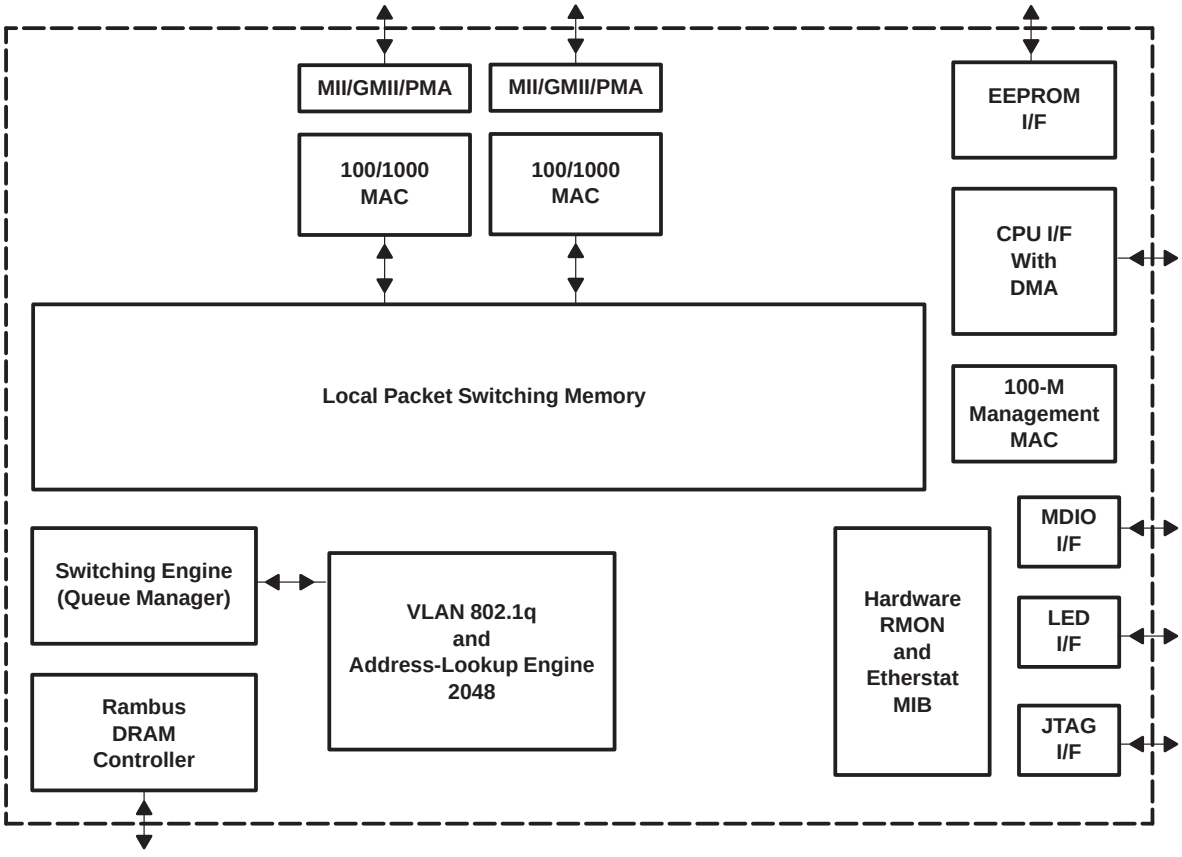
The TNETX4020 provides two 100-/1000-Mbit/s interfaces. Both ports support back-pressure flow control in half-duplex mode to reduce the risk of data loss for a long burst of activity and IEEE Std 802.3x frame-based flow control for the full-duplex mode of operation. With full-duplex capability, ports 0 and 1 support 2-Gbit/s aggregate bandwidth connections to desktops, high-speed servers, hubs, or other switches. The physical coding sublayer (PCS) function is integrated on chip to provide a direct 10-bit interface to a gigabit Ethernet transceiver. In the ring-cascade mode, both ports can be used to connect multiple TNETX4020s in a ring topology, which provides a low-cost, high-port-density desktop switch. Pretagging and extended port awareness on port 1 allows the TNETX4020 to be used as a front end to a router or crossbar matrix switch to build a cost-effective, high-density nonblocking system.

The internal address-lookup engine (IALE) supports up to 2K unicast/multicast and broadcast addresses and up to 64 IEEE Std 802.1q virtual LAN networks (VLANs). For interoperability, each port can be programmed as an access port or nonaccess port to recognize VLAN tags and transmit frames with VLAN tags to other systems that support VLAN tagging. The IALE performs destination and source address compares and forwards unknown source-address and destination-address packets to ports that are specified via programmable masks.

Statistics for the Etherstat™, SNMP, and remote monitoring (RMON) management-information base (MIB) are collected independently for each of the ports and the network-management port. Access to the statistics counters is provided via the direct input/output (DIO) interface. Furthermore, management frames can be received and transmitted via the DIO interface, thereby providing a frame I/O interface to an external CPU without requiring a MAC port interface.

The TNETX4020 memory solution combines low cost and extremely high bandwidth, using 600-Mbit/s/pin concurrent RDRAM. The packet memory has been implemented to maximize efficiency with the RDRAM architecture. Data is buffered internally and transferred to/from packet memory only in 128-byte bursts. An extremely high memory bandwidth is maintained, allowing all ports to be active without bottlenecking at the memory buffer.

Figure 1–1. TNETX4020 Block Diagram



Chapter 2

Direct Input/Output (DIO) Interface

The direct input/output (DIO) interface allows an external CPU to directly access host registers of the TNETX4020. The host registers (see Table 2–1) are one byte (8 bits) wide and are selected according to the value of the **SAD[1:0]** signals (host address).

Topic **Page**

2.1	Host Registers	2-2
2.2	Hardware Reset via the DIO	2-5
2.3	DMA Transfer	2-6
2.4	Code Examples	2-7

2.1 Host Registers

The host registers (see Table 2–2) are used indirectly to access the internal registers and statistics RAM via the DIO interface. The internal registers are used to configure the device itself and allow the external CPU to perform slave accesses to the EEPROM, PHYs, and RDRAM, as well as allow the external CPU to run the built-in self-test (BIST) on the internal RAM and CAM. In addition to DIO reads and writes to transfer data to and from the TNETX4020, the DIO interface supports direct memory access (DMA) transfer of data between the external CPU and internal registers and statistics RAM. A hardware reset can be performed on the TNETX4020 via the DIO interface.

Note:

The DIO interface is an 8-bit-wide interface. All access to the internal structures of the TNETX4020 must be done in byte-wide accesses.

Table 2–1. Host Register Address Map

Bit	Type	Name	Description	Reset Type Value	SAD[1:0] Address
7:0	r/w	***DIOAddrLo***	Lower byte of the 16-bit DIO address The external CPU must write the lower byte of the 16-bit DIO address used on subsequent access(es) to the ***DIOData*** or ***DIODataInc*** registers.	h – 0	00b
15:8	r/w	***DIOAddrHi***	Upper byte of the 16-bit DIO address The external CPU must write the upper byte of the 16-bit DIO address used on subsequent access(es) to the ***DIOData*** or ***DIODataInc*** registers. A hardware reset can be performed by writing any value in the range 0x40–0x5f into the ***DIOAddrHi*** register. This action is equivalent to asserting **RESET** low, except that the DIO interface continues to operate normally and, subsequently, no autoload from the EEPROM occurs.	h – 00h	01b

Table 2–1. Host Register Address Map (Continued)

Bit	Type	Name	Description	Reset Type Value	SAD[1:0] Address
7:0	r/w	***DIOData***	Data byte read from or written to the DIO address formed by the ***DIOAddrLo[7:0]*** and ***DIOAddrHi[15:8]*** registers When the external CPU has written a DIO address into the ***DIOAddrLo[7:0]*** and ***DIOAddrHi[15:8]*** registers, the external CPU can read/write one byte of data from/to that particular location by accessing the ***DIOData*** register.	h – 00h	10b
7:0	r/w	***DIODataInc***	Data byte read or written by the external CPU when moving a block of data When the external CPU wants to read/write a consecutive block of data, the ***DIODataInc*** register is used. The external CPU first must configure the DIO address of the first byte of data in the block to be read from or written to. After each access of the ***DIODataInc***, the TNETX4020 internally increments the DIO address by one; therefore, the external CPU does not have to specify a DIO address for each subsequent access.	h – 00h	11b

Table 2–2 shows how a 32-bit internal register would be labeled. The byte format is least significant byte in lowest address and the bits are similarly laid out – bit 0 is the least significant bit (LSBit) of the lowest byte and bit 31 is the most significant bit (MSBit) of the highest numbered byte. Single bits are labeled vertically to save space. Reserved bits or fields are shaded. For each bit, Table 2–2 gives the value after reset and which reset affects that bit.

Table 2–2. Numbering of Bits and Bytes Within a 32-Bit Register

	Lowest Byte +1								Lowest Byte Address								Byte Address Offset 0x0
Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7 MS Bit**	**6**	**5**	**4**	**3**	**2**	**1**	**0 LS Bit**	
Field Name	***major field[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

	Highest Byte Address†								Lowest Byte Address +2								Byte Address Offset 0x2
Bit	**31‡**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	
Field Name	***bit***	***minor field[3:0]***				reserved	***major field[25:16]***										
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	h	h	h	h	h	h	h	h	
Field Type	rwp	rwp	rwp	rwp	rwp	r	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

† The highest byte address is also the lowest byte address + 3.
‡ Most significant bit of the register.

Several field names recur in many registers, particularly field names for specifying where to deliver a frame that has been received. Appendix A includes a detailed definition of these fields with numerical-order definitions of the internal registers. The destinations for packets include ports 0, 1, management port on this device, and remote ports connected to port 1 via a crossbar-type switching element. If a packet is forwarded to port 1, and pretagging is enabled in the ***Port1Control*** register, then additional information is required to point to the external port(s) that receive the frame. The additional information is used to build the tag prepended to the packet to steer it through the crossbar. The format of the pretag is shown in the TNETX4020 data sheet, literature number SPWS049A. The following definitions are given in summary:

PORTCODE — A portcode specifies exactly one port on this device.

XPORTCODE — An xportcode specifies exactly one port, but may include ports of this switch, as well as other ports on a crossbar connected to port 1.

PORTVECTOR — A portvector can specify any or all of the ports on this switch (one or many) but none of the crossbar ports.

XPORTVECTOR — An xportvector can specify any or all of the crossbar ports (one or many) but none of the ports on this switch.

XROUTECODE — An xroutecode can specify either one crossbar port, or indirectly through a ***Xmultigroupn*** register, which is a xportvector.

2.2 Hardware Reset via the DIO

A hardware reset can be performed by writing any value in the range of 0x40 to 0x5F into the ***DIOAddrHi[15:8]*** register. This operation is equivalent to asserting the **$\overline{\text{RESET}}$** pin, with the following exceptions:

- The DIO interface continues to operate.
- After a hardware reset initiated via the DIO, no EEPROM autoload is performed.

2.3 DMA Transfer

The **$\overline{\text{SDMA}}$** pin of the DIO interface allows the TNETX4020 to support DMA transfers. DMA transfers move data to and from the switch via the DIO interface. Unlike DIO access, DMA transfers do not use the ***DIOAddrLo*** and ***DIOAddrHi*** registers to specify the DIO address. Instead, DMA transfers use a logical 16-bit internal register, ***DMAAddress*** (DIO address: 0x0800–0x0801), that contains the DIO address used with DMA transfers. The DIO address in the ***DMAAddress*** register is totally independent of the DIO address formed by the ***DIOAddrLo[7:0]*** and ***DIOAddrHi[15:8]*** registers. The default value of the ***DMAAddress*** register is 0x0000. The content must be configured by the external CPU using DIO writes unless DIO address 0x0000 is the desired DMA address.

In the default operation of the DMA transfer, the DIO address in the ***DMAAddress*** register does not increment after each access. By setting the ***dmainc*** bit to 1 in the ***SysControl*** register, the DIO address is incremented after each access (see *System Configuration* for more information).

When the DIO interface uses the DMA mode, there is no concept of any host register. Data is read from and written to the DIO interface. The location of the read and write is determined by the content of the ***DMAAddress***.

Note:

Loading the ***DMAAddress*** register with a value in the range of 0x4000 to 0x5F00 does not perform a hardware reset.

When the DIO interface is in DMA mode, it is a slave DMA device. It does not contend for control of the bus, but responds to a block-move operation from the host CPU.

If the ***DMAAddress*** register is moved frequently, there is no advantage to using this mechanism. If the ***DMAAddress*** register is left pointed to the ***NMData*** register, transfers using this mechanism are faster to set up than straight DIO transfers.

2.4 Code Examples

The following subsections show examples of how to access the internal registers using DIO read/write and DMA transfers. The pseudocode examples provide every mode of accessing the internal registers. However, some information, such as bit description of the internal registers used in the pseudocode, is beyond the scope of this section. Refer to Chapter 3 for detailed register descriptions.

2.4.1 DIO Write Using *DIOData* Register

The following pseudocode is an example of a 1-byte DIO write. The ***DIOAddrLo*** and ***DIOAddrHi*** registers are loaded with a 16-bit DIO address. The address loaded in this example is 0x0820, which is the DIO address for the ***NMData*** register. The data written to the ***NMData*** register is 0xFF, which is placed in the ***DIOData*** register. By using the ***DIOData*** register, the DIO address is still pointing to ***NMData*** after the access is completed.

```
{
   *DIO_addr_lo = 0x20;       // Load lower address byte
   *DIO_addr_hi = 0x08;       // Load upper address byte
   *DIO_data = 0xFF;          // Write the value of 0xFF into the NMData register
}
```

2.4.2 DIO Read Using *DIODataInc* Register

The following block of pseudocode is an example of reading a block of registers using the ***DIODataInc*** register. The ***DIOAddrLo*** and ***DIOAddrHi*** registers are loaded with the 16-bit DIO address. The address loaded in this example is 0x0000, which is the DIO address for the ***Port0Control*** register. The external CPU reads the ***DIODataInc*** register four times, i.e., four bytes are read, and both the TNETX4020 port-control registers have been read. Then the pseudocode converts the byte-wide accesses into the port-control registers' native 16-bit format. The port-control registers logically are 16 bits wide. By using the ***DIODataInc*** register, the DIO address contained in the ***DIOAddrLo*** and ***DIOAddrHi*** registers are incremented by one after completing each DIO access.

```
{
      *DIO_addr_lo = 0x00;                    //Load lower address byte
      *DIO_addr_hi = 0x00;                 //Load upper address byte
      for (i=0, i<2, i++)                     // Read 2 Port Control registers
   {
      temp = *DIO_data_inc;                   // read low byte of register
      port[i] = ((*DIO_data_inc) <<8) | temp; // read high byte & create 16 bit
                                              // register value
   }
}
```

2.4.3 Low-Level Support Routines From TSMAN

The low-level register read and write routines used in TSMAN, a program for setting up and observing ThunderSWITCH II™-based Ethernet switches, are presented in this section. A computer printer port is used to access the DIO interface of TNETX4020 devices. Schematics for connecting Texas Instruments (TI™) evaluation modules (EVMs) are available from TI. The hardware-level routines for sending a byte through the printer port, retrieving a byte through the printer port, and handling the control signals are in the header routines in Appendix B. Source code is available from TI.

This is the reference to the header information.

```
//-------------------------------------------------------------------------
// Thunderswitch interface functions
//-------------------------------------------------------------------------
// File: dio.c
//
// This module contains defined the following functions associated with
// accessing Thunderswitch hardware:
//
// TsDioRdByte – Read 1 byte from an internal register
// TsDioWrByte – Write 1 byte to an internal register
// TsDioRd     – Read n bytes from internal registers
// TsDioWr     – Write n bytes to internal registers
//
// This module uses the following hardware specific functions.
// These are dependent on the system specific processor to TSwitch
// interface and must be ported.
//
// InByte      – Read byte from TSWITCH Host Register
// OutByte     – Write byte to TSWITCH Host Register
//-------------------------------------------------------------------------

#include ”tsif.h”
#include ”tsifhw.h”
```

This code supports single-byte and multiple-byte reads.

```
//-------------------------------------------------------------------------
//
//  TsDioRdByte()    Read a byte from a Thunderswitch internal register.
//
//  Parameters:
//   WORD  wAddr     Base Address of DIO registers to read
//   BYTE* pbyteBuf  Data buffer to receive
//
//  Return val:
//   nonzero if success
//-------------------------------------------------------------------------
int TsDioRdByte( WORD wAddr, BYTE* pbyteBuf )
{
   int nError = 0;

   nError =  OutByte( TS_ADR_LO, (BYTE)wAddr );
   nError |= OutByte( TS_ADR_HI, (BYTE)(wAddr>>8) );
   if( nError )
      return( 0 );

   nError = InByte( TS_DATA, pbyteBuf );
   if( nError )
      return( 0 );
   return(1);
}

//-------------------------------------------------------------------------
//
//  TsDioRd()        Read a specified number of bytes from Thunderswitch
//                   internal register.
//
//  Parameters:
//   WORD  wAddr     Base Address of DIO registers to read
//   int   nCount    Number of bytes to read
//   BYTE* pbyteBuf  Data buffer to receive
//
//  Return val:
//   nonzero if success
//-------------------------------------------------------------------------
int TsDioRd( WORD wAddr,int nCount,BYTE* pbyteBuf)
{
   int nError = 0;

   nError =  OutByte( TS_ADR_LO, (BYTE)wAddr );
   nError |= OutByte( TS_ADR_HI, (BYTE)(wAddr>>8) );
   if( nError )
      return( 0 );

   while( (nCount--) && !nError )
      nError = InByte( TS_DATA_INC, pbyteBuf++ );
   if( nError )
      return( 0 );
   return(1);
}
```

This code supports single-byte and multiple-byte writes of information to DIO registers.

```
//---------------------------------------------------------------------------
//
//  TsDioWrByte()    Write a byte to a Thunderswitch internal register.
//
//  Parameters:
//   WORD  wAddr     Base Address of DIO registers to write
//   BYTE* pbyteBuf  Data buffer to write
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------------
int TsDioWrByte( WORD wAddr, BYTE* pbyteBuf )
{
    int nError = 0;

   nError =  OutByte( TS_ADR_LO, (BYTE)wAddr );
   nError |= OutByte( TS_ADR_HI, (BYTE)(wAddr>>8) );
   if( nError )
      return(0);

   nError = OutByte( TS_DATA, *pbyteBuf );
   if( nError )
      return(0);
   return(1);
}

//---------------------------------------------------------------------------
//
//  TsDioWr()        Write a specified number of bytes to Thunderswitch
//                   internal register
//
//  Parameters:
//   WORD  wAddr     Base Address of DIO registers to write
//   int   nCount     Number of bytes to write
//   BYTE* pbyteBuf  Data buffer to write
//
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------------
int TsDioWr( WORD wAddr, int nCount, BYTE* pbyteBuf )
{
   int nError = 0;

   nError =  OutByte( TS_ADR_LO, (BYTE)wAddr );
   nError |= OutByte( TS_ADR_HI, (BYTE)(wAddr>>8) );
   if( nError )
      return(0);

   while( (nCount--) && !nError )
      nError = OutByte( TS_DATA_INC, *pbyteBuf++ );
   if( nError )
      return(0);
   return(1);
}
```

2.4.4 DMA Write – Same DIO Address

DMA transfers can be used instead of DIO accesses. However, for the DMA transfer to operate correctly, certain internal registers must be configured via DIO accesses. This example uses a single DIO address multiple times; therefore, the ***SysControl*** register must be configured not to increment the DIO address after completion of the DMA transfer (see ***SysControl*** register in Chapter 3). The DIO address used by the DMA transfer must be configured in the ***DMAAddress*** register. Each subsequent DMA transfer uses the DIO address configured in the ***DMAAddress*** register (see ***DMAAddress*** register in Chapter 3). The DIO address provided in this example is the DIO address ***RAMData***. This sequence loads 64 of the 128 bytes in the internal buffer RAM for later transfer to external RDRAM.

```
{
   *DIO_addr_lo = 0xFA;          // Load lower address byte
   *DIO_addr_hi = 0x00;          // Load upper address byte (SysControl)

   *DIO_data_inc = 0x00;         // Write the value of 0x0000 into SysControl
   *DIO_data_inc = 0x00;         // register to ensure DMA address is not
                                 // incremented.

   *DIO_addr_lo = 0x00;          // Load lower address byte
   *DIO_addr_hi = 0x08;          // Load upper address byte (DMAAddress)

   *DIO_data_inc = 0x14;         // Provide the DIO address of RAMData
   *DIO_data_inc = 0x08;

   for (i=0, i < 64, i++)
   {
     *dma = 0xFF;                // Write a 64-byte pattern of 0xFF into RAMData.
   }
}
```

2.4.5 DMA Read – Incrementing DIO Address

The following pseudocode is an example of using DMA transfers where the DIO address is incremented after completing each DMA transfer. For the DIO address to be incremented after completing each DMA transfer, bit 6 of the ***SysControl*** register must be set (see ***SysControl*** register for more detail). The DIO address for the DMA transfer is configured in the ***DMAAddress*** register. In this example, DIO address, 0x8000, which is the beginning of the Statistics RAM, is configured. The external CPU reads the first 128 bytes from the statistics RAM, which is the port statistics of port 00. The byte accesses are converted into the statistics native format: 32-bit-wide registers.

```
{

   *DIO_addr_lo = 0xFA;          // Load lower address byte
   *DIO_addr_hi= 0x00;           // Load upper address byte (SysControl)

   *DIO_data_inc = 0x40;         // Write the value of 0x0040 into SysControl
   *DIO_data_inc= 0x00;          // register to ensure DMA address is incremented

   *DIO_addr_lo = 0x00;          // Load lower address byte
   *DIO_addr_hi = 0x08;          // Load upper address byte (DMAAddress)

   *DIO_data_inc = 0x00;         // Provide the DIO address for the start of the
   *DIO_data_inc = 0x80;         // statistics RAM (port 00)

   x = 0;                        // Initialize x to zero

   for (i=0, i < 128, i++)       // Reads 18 bytes starting at DIO
   {                             // address = 8000h
      temp[i] = *dma;
         if ((i mod 4) = 3)      // Converts the byte accesses into 32-bit reg.
                                 // Stop on MS byte and build 32-bit word

      {  stat[x] = temp[i–3] | (temp[i–2] << 8) | (temp[i–1] << 16) | (temp[i] << 24);
          x++;
      }
   }
}
```

Chapter 3

System Test

It is recommended that the external CPU perform a system test/diagnostic on power up and after a hardware reset. The TNETX4020 has several test features that allow the software engineer to easily implement anything from simple power-up diagnostics to an exhaustive system diagnostic (see Table 3–1). These test features can be utilized only when both the ***start*** and ***initd*** bits in the ***SysControl*** register are set to 0. The ***start*** and ***initd*** bits are set to 0 after a hardware reset, after a reset initiated via the DIO, or after the ***stop*** bit in the ***SysControl*** register has been set to 1.

Topic **Page**

3.1 Internal CAM and RAMs 3-4

3.2 Accessing External RAM (RDRAM) 3-5

3.3 Switch-Functionality Test 3-11

SysTest (System Test Register) @ 0x080F

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***passed***	***bist[1:0]***		***rdwrite***	***rdram***	***dpwrap***	***intwrap[1:0]***		
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	
Field Type	r	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Table 3–1. SysTest (System Test Register)

Bit	Name	Description
7	***passed***	BIST passed. A 1 in this bit indicates that the previously initiated internal RAM or CAM BIST passed. It is cleared automatically when 01 or 10 is written to ***bist***.
6:5	***bist***	Initiate BIST. Initiates the internal RAM BIST processes. Upon completion, this field self-clears, and the result of the test is recorded in ***passed***. The coding is: 00 No BIST 01 Enable BIST of all internal RAMs 10 Enable BIST of CAM 11 Reserved
4	***rdwrite***	RDRAM write. This bit determines the transfer direction when a 1 is written to ***rdram***: ❑ ***rdwrite*** = 1. The RDRAM transfer is a write. ❑ ***rdwrite*** = 0. The RDRAM transfer is a read.
3	***rdram***	RDRAM test access. Initiates a 128-byte transfer between an internal buffer and the external RDRAM. The transfer direction is specified by ***rdwrite***. The RDRAM page address is specified by ***ramaddress***. When the 128-byte transfer is complete, this bit is cleared by hardware, allowing completion to be determined by polling.
2	***dpwrap***	Duplex wrap test mode. When high, all ports are forced into full-duplex mode and active link is asserted, so all ports can receive frames they transmit, thus enabling external wrap testing at the physical layer (PHY). If port 0 or 1 is in PMA mode (***pma*** = 1 in ***PortxControl***), this capability is controlled via loopback in ***PCSxControl***.
1:0	***intwrap***	Internal wrap test mode. Ports 00–01 internally wrap back with full duplex and link according to the two-bit coding: 00 No internal wrapping 01 All ports internally wrapped 10 All ports internally wrapped, except port 0 11 All ports internally wrapped, except port 1 The network management port never can wrap. ***intwrap*** takes priority over ***dpwrap*** if both bits are set to 1.

The external CPU controls what system test or diagnostic features to use via the ***SysTest*** register. The features include testing all the internal RAM and CAM using a built-in self-test (BIST), allowing access to the external RAM, testing the functionality of the switch by configuring the device in internal wrap test mode, and testing the MII connections, as well as the functionality of the switch, by configuring the device in duplex wrap-test mode.

3.1 Internal CAM and RAMs

The internal CAM and all the internal RAMs are tested by setting the built-in self-test ***(bist)*** bits in the ***SysTest*** register to 0b10 and 0b01, respectively. Setting the ***bist*** bits to these values invokes the BIST for that particular internal memory. The TNETX4020 clears the value in the ***bist*** bits and sets the ***passed*** bit to 1 to indicate that the test was successful. (If the BIST fails, the ***passed*** bit is set to 0.)

The TNETX4020 uses the MARCH C+ BIST algorithm to test all the internal RAMs. The BIST checks the RAMs for these defects:

- Stuck-at memory cells
- Shorts between memory cells
- Static coupling between memory cells (the state of cell *i* causes cell *j* to appear stuck)
- Dynamic coupling between memory cells (access to cell *i* causes a transition in cell *j*) in different words
- Inaccessible cells (floating or stuck word lines or open across transistor)
- Shorts between adjacent word lines, bit lines, or supply lines to the array

The CAM BIST algorithm checks that all CAM comparators function correctly. The CAM BIST algorithm performs these tests on the CAM:

- Load every location in the CAM with a 1. Compare with a word of all 1s and the CAM should assert the “all match” signal.
- Walk a 0 through every location of the CAM and compare with a word of all 1s. The CAM never should assert the “all match” signal.
- Load every location in the CAM with a 0. Compare with a word of all 0s and the CAM should assert the “all match” signal.
- Walk a 1 through every location of the CAM and compare with a word of all 0s. The CAM never should assert the “all match” signal.

3.2 Accessing External RAM (RDRAM)

The TNETX4020 allows the external CPU to access the external RAM (RDRAM) by using the ***rdwrite*** and ***rdram*** bits of the ***SysTest*** register in conjunction with the ***RAMAddress*** and ***RAMData*** registers (see Tables 3–2 and 3–3). The ***RAMAddress*** register must be configured with the starting address of the access while the ***RAMData*** register is used by the external CPU to transfer the data to and from the RDRAM. The ***rdwrite*** bit determines if the access to the RDRAM is a read or a write access, and the ***rdram*** bit initiates a 128-byte transfer. (The TNETX4020 accesses the RDRAM using 128-byte pages.) The ***rdram*** bit is cleared after the completion of the transfer. That is, for a read operation, the ***rdram*** bit is cleared when the 128 bytes from RDRAM have been buffered on-chip. And, for a write operation, the ***rdram*** bit is cleared when the buffered data has been written to the RDRAM. Hence, the ***rdram*** bit indicates the completion of an RDRAM transfer and can be polled by the external CPU.

Whether performing a read or a write access of the RDRAM, the ***RAMAddress*** register must be the first register that is configured. The external CPU must write the starting address of the access in the ***ramaddress*** bit field. By configuring the ***inc*** bit to 0, the external CPU either reads from or writes to the same address, i.e., individual byte access. When the ***inc*** bit is configured to 1, the address is incremented after each access. When accessing the external RAM, the ***ramsel*** bit field must be configured to 0b000.

When reading or writing the internal 128-byte transfer RAM, only the lower 7 bits of ***RAMAddress*** matter. When the internal RAM is transferred to or from the RDRAM page using the ***SysTest*** register bits, only the address bits above the lower 7 matter. The transfer to or from RDRAM starts with the lower 7 address lines = 0.

After performing a 128-byte access to the internal buffer with ***inc*** = 1 in the ***RAMAddress*** register, the ***ramaddress*** field points to the next page of RDRAM memory. The upper bits of ***RAMAddress*** must be set to the desired page before a transfer to or from external RDRAM is triggered by setting the control bits in the ***SysTest*** register.

Note:

Before accessing the external RAM, the CPU must wait until the RDRAM has been initialized. This is achieved by the CPU polling the ***rdinit*** bit in the ***RAMStatus*** register. The ***rdinit*** bit is set to indicate that the RDRAM has been initialized successfully.

RAMAddress (RAM Address Register) @ 0x0810–0x0813

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***ramaddress[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***inc***	***ramsel[3:0]***				reserved	***ramaddress[25:16]***										
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	h	h	h	h	h	h	h	h	
Field Type	rwp	rwp	rwp	rwp	rwp	r	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Table 3–2. RAMAddress (RAM Address Register)

Bit	Name	Description
31	***inc***	RAM address autoincrement. When set to 1, ***ramaddress*** autoincrements after each access to/from ***RAMData***. If ***ramaddress*** is pointing at a reserved address, the increment does not occur.
30:27	***ramsel***	RAM select. This field selects the RDRAM to be accessed. 0000 External RDRAM 0001–1111 Reserved
26	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
25:0	***ramaddress***	RAM address. Indicates the byte within external RDRAM to be accessed. ❑ ***RAMData***, in conjunction with ***rdram*** and ***rdwrite*** in ***SysTest***, initiates reads or writes to the RDRAM.

When performing a read or a write access, the external CPU must access the 128-byte block of data via the ***RAMData*** register.

RAMData (RAM Data Register) @ 0x00814

Bit	7	6	5	4	3	2	1	0	DIO Address Offset
Field Name	***ramdata[7:0]***								
Reset Value	–	–	–	–	–	–	–	–	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Table 3–3. RAMData (RAM Data Register)

Bit	Name	Description
7:0	***ramdata***	RAM test data. In conjunction with ***RAMAddress*** and ***SysTest***, performs test accesses to the external RDRAM memory. ❑ Before initiating an RDRAM write, the 128 bytes to be transferred to RDRAM should be written to this address. ❑ After initiating an RDRAM read and waiting for it to complete, the 128 bytes to be transferred from RDRAM should be read from this address. RDRAM accesses can be performed only when ***start*** = 0 and ***initd*** = 0 in ***SysControl***.

When reading a 128-byte block from the external RAM, the external CPU must configure the starting address in the ***RAMAddress*** register, the lower 7 bits are assumed = 0, regardless of the actual pattern in these bits. To initiate the read, the external CPU must set the ***rdwrite*** bit to 0 and, at the same time, set the ***rdram*** bit to 1 in the ***SysTest*** Register. The external CPU can poll the ***rdram*** bit, because it clears when the data is available to be read out via the ***RAMData*** register. The ***rdram*** bit is cleared to indicate the 128-byte block of data is now buffered internally. The host CPU reads data from the internal buffer by supplying the address in the lower 7 bits of ***RAMAddress*** and reading ***RAMData***. Blocks of reads take place if the ***inc*** bit is set in ***RAMAddress***. If more than 128 reads or writes to the internal buffer take place with the ***inc*** bit in ***RAMAddress*** set, the 7-bit address pointer wraps.

To write a 128-byte block to the external RAM, the external CPU must configure the starting address in the internal RAM in the ***RAMAddress*** register. The external CPU then must write as much data as needed into the 128-byte block data to the ***RAMData*** register. To initiate the write access, the external CPU must set the ***rdwrite*** and ***rdram*** bits to 1 in the ***SysTest*** register. The external CPU can poll the ***rdram*** bit as it clears at the completion of the transfer. The entire 128-byte internal RAM is written to the external page specified in ***RAMAddress*** and is page aligned. That is, the transfer starts at the address in ***RAMAddress*** with the 7 least significant bits masked to zero and sequences until the 7 least significant bits are 0x7F. The upper bits do not change.

3.2.1 Code Example

```
// RDAM    Read + write overhead, header, and assumptions:
//-------------------------------------------------------------------------
// Thunderswitch interface functions
//-------------------------------------------------------------------------
// File: rdram.c
//
// This module contains defined the following functions associated with
// accessing RDRAM:
//
// TsRdramRd   – Read data from RDRAM (up to 128 bytes)
// TsRdramWr   – Write data to RDRAM (up to 128 bytes)
//
// This module uses the following hardware specific functions.
// These are dependent on the system specific processor to TSwitch
// interface and must be ported.
//
// InByte      – Read byte from TSWITCH Host Register
// OutByte     – Write byte to TSWITCH Host Register
//
//-------------------------------------------------------------------------

#include ”tsif.h”             // See Appendix B
#include ”tsifhw.h”           // See Appendix B

#define TS_SYSTEST   0x080F   // DIO address of TSwitch SysTest register
#define TS_RAM_ADDR  0x0810   // DIO address of TSwitch RAMAddress register
#define TS_RAM_DATA  0x0814   // DIO address of TSwitch RAMData register

#define RDWRITE      0x10        // rdwrite bit in SysTest register
#define RDRAM        0x08        // rdram bit in SysTest register
#define RAM_AUTO_INC 0x80000000  // inc bit in RAMAddress register
#define RAM_SEL      0x70000000  // ramsel field in RAMAddress register
```

```
//---------------------------------------------------------------------------
//
//  TsRdramRd        Read a specified number of bytes from RDRAM. Note that
//                   this function will only read up to 128 bytes
//                   correctly.
//


//  Parameters:
//   DWORD dwAddr    Starting byte address of RDRAM to read
//   int   nCount    Number of bytes to read
//   BYTE* pbyteBuf  Data buffer to receive
//
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------------
int TsRdramRd( DWORD dwAddr, int nCount, BYTE *pbyteBuf )
{
   BYTE  bData;

   //-------------------------------------------------------------------------
   // Set up RAMAddress register:
   //    ramaddress field is set to byte address of RDRAM to read from
   //    ramsel field is set to 000b (to select external RDRAM)
   //    inc field is set to enable autoincrement
   //-------------------------------------------------------------------------
   dwAddr &= ~RAM_SEL;     // Clear bits 28–30 to select external RDRAM
   dwAddr |= RAM_AUTO_INC; // Set inc bit to autoincrement ramaddress after
                           //   each access to/from RAMData
   TsDioWr( TS_RAM_ADDR, 4, (BYTE*)&dwAddr ); // Set up RAMAddress register

   //-------------------------------------------------------------------------
   // Initiate transfer of 128 bytes of data from RDRAM to internal
   // transfer buffer.
   //-------------------------------------------------------------------------
   TsDioRdByte( TS_SYSTEST, &bData );
   bData &= ~RDWRITE;                  // Specify a read operation
   bData |= RDRAM;                     // Initiate transfer
   TsDioWrByte( TS_SYSTEST, &bData );  // Go!
   do                   // Wait for transfer to be complete (RDRAM=0)
      TsDioRdByte( TS_SYSTEST, &bData );
   while( bData & RDRAM );

   //-------------------------------------------------------------------------
   // Read data from the transfer buffer.
   //-------------------------------------------------------------------------
   dwAddr = 0;             // Point to start of transfer buffer
   dwAddr |= RAM_AUTO_INC; // Set inc bit to autoincrement ramaddress after
                           //   each access from transfer buffer
   TsDioWr( TS_RAM_ADDR, 4, (BYTE*)&dwAddr ); // Set up RAMAddress register

   //-------------------------------------------------------------------------
   // Read data from transfer buffer to destination
   //-------------------------------------------------------------------------
   return TsDioRd( TS_RAM_DATA, nCount, pbyteBuf );
}
```

```
//---------------------------------------------------------------------------
//
//  TsRdramWr        Write a specified number of bytes to RDRAM. Note that
//                   this function will only write up to 128 bytes
//                   correctly.
//


//  Parameters:
//   DWORD dwAddr    Starting byte address of RDRAM to write
//   int   nCount    Number of bytes to write
//   BYTE* pbyteBuf  Data buffer containing data to write
//
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------------
int TsRdramWr( DWORD dwAddr, int nCount, BYTE *pbyteBuf )
{
   BYTE  bData;

   //-------------------------------------------------------------------------
   // Write data from the source address to the transfer buffer.
   //-------------------------------------------------------------------------
   dwAddr = 0;             // Point to start of transfer buffer
   dwAddr |= RAM_AUTO_INC; // Set inc bit to autoincrement ramaddress after
                           //   each access from transfer buffer
   TsDioWr( TS_RAM_ADDR, 4, (BYTE*)&dwAddr ); // Set up RAMAddress register
   if( TsDioWr( TS_RAM_DATA, nCount, pbyteBuf ) == 0 )
      return 0;

   //-------------------------------------------------------------------------
   // Set up RAMAddress register:
   //    ramaddress field is set to byte address of RDRAM to write to
   //    ramsel field is set to 000b (to select external RDRAM)
   //    inc field is set to enable autoincrement
   //-------------------------------------------------------------------------
   dwAddr &= ~RAM_SEL;     // Clear bits 28-30 to select external RDRAM
   dwAddr |= RAM_AUTO_INC; // Set inc bit to autoincrement ramaddress after
                           //   each access to/from RAMData
   TsDioWr( TS_RAM_ADDR, 4, (BYTE*)&dwAddr ); // Set up RAMAddress register

   //-------------------------------------------------------------------------
   // Initiate transfer of 128 bytes of data from internal transfer buffer
   // to RDRAM.
   //-------------------------------------------------------------------------
   TsDioRdByte( TS_SYSTEST, &bData );
   bData |= RDWRITE;                   // Specify a write operation
   bData |= RDRAM;                     // Initiate transfer
   TsDioWrByte( TS_SYSTEST, &bData );  // Go!
   do                   // Wait for transfer to be complete (RDRAM=0)
      TsDioRdByte( TS_SYSTEST, &bData );
   while( bData & RDRAM );

   return 1;
}
```

3.3 Switch-Functionality Test

The TNETX4020 can be configured to operate in internal wrap test mode and duplex wrap test mode by setting the ***intwrap*** bits or the ***dpwrap*** bit, respectively, in the ***SysTest*** (0x80F) register to be used during a power-up diagnostic as well as a full system diagnostic. The purpose of these modes is to test the functionality of the switch. The internal wrap test forces some or all of the Ethernet MACs in a loopback mode. (The management port cannot be configured in a loopback mode.) Frames forwarded to a MAC, i.e., put in the Tx buffer, are looped back into the receive path, i.e., received in the Rx buffer. The duplex wrap test is similar to the internal wrap test, except that the frames are wrapped (looped back) at the PHY. The duplex wrap mode requires that the system configure the PHYs in loopback mode via the MDIO interface *(*refer to Chapter 5, *System Operations*). Note that none of the bits in the ***SysTest*** register can be changed once the TNETX4020 is started. One must:

1) Set ***reset*** in ***SysControl*** to 1.
2) Set ***reset*** in ***SysControl*** to 0.
3) Wait for the RDRAM initialization to complete (***rdinit*** = 1 in ***RAMStatus***).
4) Set any ***SysTest*** bits that need to be changed from reset values.
5) Set the ***start*** bit in ***SysControl*** to 1 to start the switch.

3.3.1 Internal Wrap Test

The ***intwrap*** bits in the ***SysTest*** register determine which ports loop back. Either port can be configured to not loop back, allowing them to be used as the start/end port for the test. Alternatively, the network management port (accessed via DIO) can be used for this purpose, with all MII ports configured to loop back.

For a frame to be forwarded from one port to another (see Figure 3–1), the switch must be programmed as follows:

- ❑ Assign a unique IEEE Std 802.1q VLAN ID to each of the ***PortxQTag*** registers, and program these tags into the ***VLANnQID*** registers.
- ❑ The ***VLANnPorts*** register associated with each of the ***VLANnQID*** registers should have the bits set to 1 for the inbound port and the destination port, indicating to which port frames containing that IEEE Std 802.1q VLAN ID should be routed.
- ❑ ***rxacc*** and ***txacc*** for each port must be 1. This causes the port to add the VLAN ID from its ***PortxQTag*** to the frame upon reception, and strip the tag before transmission.
- ❑ The destination address of the frames to be applied should not be known or learned, and ***UnkUniPorts*** and ***UnkMultiPorts*** should be all 1s, and the ***portvector*** of ***NlearnPorts*** should be all 1s as well.
- ❑ Receive a frame with an unknown destination address on the access port. The access port is the port that is not configured in a loopback mode, i.e., port 0, port 1, or network management port, depending on how the ***intwrap*** bit field is configured.

For detailed information on how to configure the TNETX4020, refer to Chapter 4, *System Configuration*.

After the switch is programmed, the following occurs:

1) The VLAN ID from the source port's ***PortxQTag*** register is added to the frame upon reception. Because the address of the frame is unknown, the frame is forwarded to the logical AND of the ***portvector*** fields of the appropriate ***VLANnPorts*** and ***UnkUniPorts*** (unicast) or ***UnkMultiPorts*** (multicast) registers. ***VLANnPorts*** should contain two bits set to 1, indicating the source port and destination port.

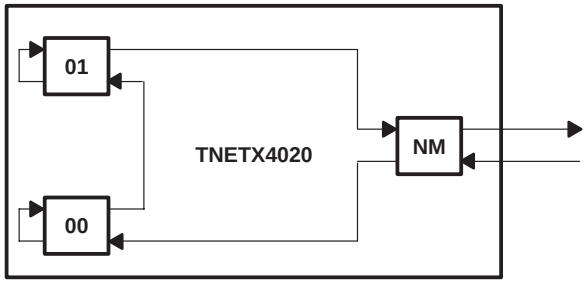
2) The frame is transmitted from the destination port selected in Step 1. Its VLAN ID is stripped beforehand, and the frame loops back to the receive path.

3) Steps 1 and 2 are repeated, but the VLAN ID added upon reception is different from the one previously stripped off at transmission. This means a different ***VLANnPorts*** register is used to determine the destination.

   The port order shown here is sequential, but the actual order depends on how ports are paired in the ***VLANnPorts*** registers, and how the ***PortxQTag*** registers are assigned.

4) Eventually, the frame is sent to a port that is not configured for loopback and the frame leaves the switch.

Figure 3–1. Internal-Wrap-Test Example



The operational status of the PHYs or external connections to the device do not have to be considered or assumed good when in internal-loopback mode.

3.3.1.1 Pseudocode

The following pseudocode procedure configures the TNETX4020 for the internal-wrap-test example described in Figure 3–1. It is assumed that both the ***start*** and ***initd*** bits in the ***SysControl*** register are set to 0.

```
// Internal wrap test according to figure 3–1
{
    *DIO_addr_lo = 0x0F;
    *DIO_addr_hi = 0x08;       // DIO address: SysTest register

    *DIO_data = 0x01;          // intwrap = 1, enables Internal Wrap Mode, all
                               // ports. Device is not started yet

    *DIO_addr_lo = 0x60;
    *DIO_addr_hi = 0x00;       // DIO address: UnkUniPorts register

    *DIO_data_inc = 0xFF;      // Port vector for unknown unicast is set to all 1s
    *DIO_data_inc = 0xFF;

    *DIO_addr_lo = 0x64;
    *DIO_addr_hi = 0x00;       // DIO address: UniMultiPorts register

    *DIO_data_inc = 0xFF;      // Port vector for unknown multicast is set to
    *DIO_data_inc = 0xFF;      // all 1s

    *DIO_addr_lo = 0x00;
    *DIO_addr_hi = 0x03;       // DIO address: VLAN0QID register

    *DIO_data_inc = 0x09;      // VLAN 0 Q ID is set to 0x009. When a frame
    *DIO_data_inc = 0x00;      // contains this VLAN tag, then the port members are
                               // in VLAN0Ports.

    *DIO_data_inc = 0x05;      // VLAN 1 Q ID is set to 0x005. When a frame
    *DIO_data_inc = 0x00;      // contains this VLAN tag, then the port members
                               // are in VLAN1Ports.

    *DIO_data_inc = 0x07;      // VLAN 2 Q ID is set to 0x007. When a frame
    *DIO_data_inc = 0x00;      // contains this VLAN tag, then the port members
                               // are in VLAN2Ports.

    *DIO_addr_lo = 0x00;
    *DIO_addr_hi = 0x01;       // DIO address: VLAN0Ports register

    *DIO_data_inc = 0x05;      // Port members of VLAN 0: Port 0 and NM port
    *DIO_data_inc = 0x00;
    *DIO_data_inc = 0x00;
    *DIO_data_inc = 0x00;

    *DIO_data_inc = 0x03;      // Port members of VLAN 1: Ports 1 and 0
    *DIO_data_inc = 0x00;
    *DIO_data_inc = 0x00;
    *DIO_data_inc = 0x00;

    *DIO_data_inc = 0x06;      // Port members of VLAN 2: Port 1 and NM port
    *DIO_data_inc = 0x00;
    *DIO_data_inc = 0x00;
    *DIO_data_inc = 0x00;
```

```
    *DIO_addr_lo = 0x80;
    *DIO_addr_hi = 0x03;       // DIO address: Port0QTag register

    *DIO_data_inc = 0x05;      // Any frame received on port 0 gets a VLAN
    *DIO_data_inc = 0x00;      // tag of 0x005. This frame is then associated
                               // with VLAN 1

    *DIO_addr_lo = 0x82;
    *DIO_addr_hi = 0x03;       // DIO address: Port1QTag register

    *DIO_data_inc = 0x07;      // Any frame received on port 1 gets a VLAN
    *DIO_data_inc = 0x00;      // tag of 0x007. This frame is then associated
                               // with VLAN 2

    *DIO_addr_lo = 0xFA;
    *DIO_addr_hi = 0x00;       // DIO address: SysControl register

    *DIO_data_inc = 0x00;      // start bit = 1. Cause the device to begin
    *DIO_data_inc = 0x02;      // operation.

    while (!(initd))           // Poll the initd bit of the SysControl register.
                               // When this bit is is set to one the initialization
                               // of the device is done.

    rx_unknown_frame();        // Via the DIO interface make the switch receive a
                               // frame with an unknown destination address
                               // (mulitcast or unicast) with a VLAN tag of 0x009.
                               // Refer to section 5 for information on how to
                               // transmit (section 5.3.2) and receive
                               (section 5.3.4) frames with the NM port.

End
```

Because the NM port does not have a ***PortxQTag*** register, the packet sent into the NM port in this example must have a tag, and the value must be of a VLAN already defined or the packet will be filtered.

3.3.2 Duplex Wrap Test

The duplex wrap test is similar to the internal-wrap mode. The ports are set to accept frame data that is wrapped at the PHY (or SERDES device if ports 00 or 01 are in PMA mode). This permits network connections between the device and the PHY to be verified. Any port can be the source port, not just the network management port, as shown in Figure 3–2. By using multicast/broadcast frames, traffic is routed selectively between ports involved in the test or return the frame directly before retransmission on the uplink.

Software control of the external PHYs is required to configure them for loopback (see Chapter 5, *System Operations* for details on how to configure the PHYs via the MDIO interface). If the internal PCS is in use on either gigabit port (port 00 or 01 configured in PMA mode), ***loopback*** in ***PCSxControl*** also must be asserted.

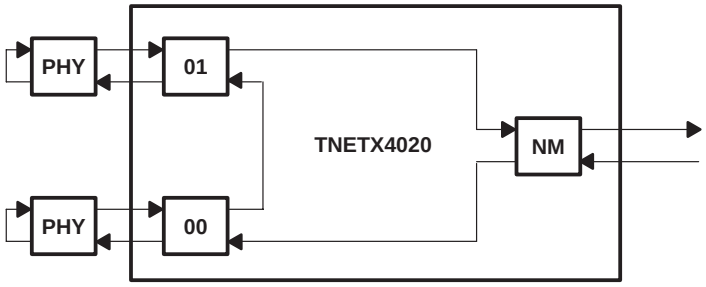
The duplex frame-wrap test mode is selected by setting ***dpwrap*** in ***SysTest***. When selected, each port is forced into full duplex, allowing it to receive frames it transmits.

Note:

Most external PHYs do not assert DUPLEX in wrap mode.

Configure the switch as in the internal wrap test.

Figure 3–2. Duplex-Wrap-Test Example



3.3.2.1 Pseudocode

The pseudocode to configure the device for the duplex wrap-mode test is exactly the same as the pseudocode for the internal-wrap-mode test, with the following exceptions:

- The ***dpwrap*** bit must be set to 1 in the ***SysTest*** register while ***intwrap*** is set to 0. If both ***dpwrap*** and ***intwrap*** bits are set to 1, unpredictable behavior can result.
- The PHYs must be configured for loopback. This must be done by the external CPU via the MDIO interface. Before setting the ***start*** bit in the ***SysControl*** register, it is recommended that the PHYs be configured in loopback mode. Refer to Chapter 5, *System Operations,* for details on how to configure the PHYs via the MDIO interface.
- If Port 00 or 01 is configured in PMA mode, the ***loopback*** bit must be configured in the ***PCSxControl*** register.

Chapter 4

System Configuration

Immediately after the TNETX4020 exits a hardware reset, the device automatically attempts to download configuration and initialization information from the EEPROM. The EEPROM, depending on its size, configures and initializes the port control, address lookup, VLAN, and systems registers. (For detailed information regarding the EEPROM download, refer to the EEPROM application report.) The TNETX4020 does not allow any DIO accesses during the reset or the EEPROM download.

When the external CPU is allowed to perform DIO accesses, the system can be initialized and configured as desired. This chapter describes, from a register and bit-level point of view, how to configure the entire system (see Tables 4–1 through 4–32).

Note:

This chapter addresses how to configure the TNETX4020 via the DIO interface, i.e., an EEPROM-less system. The discussion focuses on a system with an external CPU. However, this chapter is valuable to vendors using an unmanaged system and configuring the TNETX4020. This information also is applicable when programming the EEPROM.

Topic **Page**

4.1	Introduction	4-2
4.2	Port Configuration	4-7
4.3	Internal Address-Lookup Engine (IALE) Configuration	4-27
4.4	Cascading-Ring Topology	4-47
4.5	Statistics Configuration	4-52
4.6	LED Configuration	4-54

4.1 Introduction

Before discussing registers that control individual ports, two global control registers should be considered: the ***SysControl*** register and the ***RAMStatus*** register. Bits in these registers provide the following control:

- ❏ Stopping operation of the TNETX4020
- ❏ Causing a reload of the EEPROM to internal registers
- ❏ Starting normal operation after a stop or reset
- ❏ Enabling internal control of address aging
- ❏ Enabling a reset on internal error
- ❏ Enabling flow control (pause or collision depends on port mode) and others

Some global status information also is available in these registers, such as:

- ❏ Whether the RDRAM initialized successfully after reset
- ❏ Whether the RDRAM is operating in parity mode

The following ***SysControl*** information defines these bits.

SysControl (System Control Register) @ 0x00FA–0x00FB

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***stop***	***load***	***start***	***initd***	***reboot***	***holb***	reserved	***cnotify***	***nage***	***dmainc***	reserved	***unkvlan***	***mirr***	***nauto***	***ncrc***	***flow***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	hs	h	h	–	h	h	h	–	h	h	h	h	h	
Field Type	rw	rw	rwl	r	rwl	rwl	r	rwl	rwl	rwl	r	rwl	rwl	rwl	rwl	rwl	

Table 4–1. SysControl (System Control Register)

Bit	Name	Description
15	***stop***	Stop system. When ***stop*** = 1, device operation is halted, and remains so until this bit is cleared to 0. All internal state machines, FIFOs, and MACs are reset. Register bits are cleared as per the register definition tables in this section. The host registers are not affected. Any data in the device is lost. ❑ This bit is set to 1 if an EEPROM load fails because of a bad cyclic-redundancy check (CRC) word. ❑ This bit is set to 1 if a parity error is detected on the external memory interface.
14	***load***	Load system. Writing a 1 to this bit causes the DIO registers to be autoloaded from an external EEPROM (if present). Writing a 0 to this bit has no effect. Writing a 1 to this bit also clears the CRC-error indication from the fault LED. All DIO operations are inhibited (**SRDY** is held inactive high) while ***load*** is a 1 once the EEPROM state machine has finished checking that an EEPROM is present and begins to download data. ***load*** clears to 0 once the download is completed. When this bit is written with a 1, the ***stop*** bit simultaneously should be written with a 0.
13	***start***	Start system. Writing a 1 to this bit causes the device to begin operation, following a reset or stop. This bit is read as a 1 until buffer and address-lookup memory initialization is complete. This causes any previous frames or address records to be erased. While memory is being initialized, all ports are disabled. All DIO writes are inhibited (**SRDY** inactive high) while ***start*** is a 1. Writing a 1 to this bit also clears the parity-error indication from the fault LED. Writing a 0 to this bit has no effect; nor does writing a 1 when ***initd*** = 1. When this bit is written with a 1, the ***stop*** bit simultaneously should be written with a 0.
12	***initd***	Initialization done. This bit becomes a 1 after ***start*** has been set, and buffer and address-lookup-memory initialization is complete (i.e., at the same time ***start*** is cleared). Once set, this bit remains so until a hard reset (pin or DIO), or until ***stop*** is set to 1. Until this bit is a 1, the ports ignore any frames they receive.
11	***reboot***	Reboot. When set to 1, this bit enables unmanaged systems to recover from fatal errors without manual intervention. ❑ ***reboot*** = 0. A fault causes ***stop*** to be set. ❑ ***reboot*** = 1. A fault causes a hard reset, thereby allowing the device configuration to be reloaded automatically from EEPROM and the device to be restarted. A fault occurs if an RDRAM parity error is detected or if any internal state-consistency check fails.

Table 4–1. SysControl (System Control Register) (Continued)

Bit	Name	Description
10	***holb***	Head of line blocking. When set, ***holb*** is enabled. This function prevents switch congestion on a channel from consuming all the memory resources, causing other channels to be affected. ❑ If ***flow*** is disabled with ***holb*** enabled, the switch discards frames after the ***holb*** threshold has been exceeded (with the exception of the NMPORT). ❑ If ***flow*** is enabled with ***holb***, the switch attempts to preserve frames already in the fabric and prevent further submission on RX ports, contributing to the congestion. The ***FLOW*** LED illuminates to indicate that a flow request was made. Where it is not possible to use flow control, frames are discarded from the Rx FIFO and counted as an overrun. When the NMPort enters ***holb***, the ***NumFreeBufs*** field in ***NMRxControl***, is held at zero after the next ***EOF*** bit write, until the condition clears, preventing further frame submission from the host computer. No frames are discarded that are received from the NMPORT. When the ***holb*** threshold is exceeded and the ***flow*** bit is set, the appropriate ***Mxx_FLOW*** pin on the port that entered ***holb*** is asserted.
9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***cnotify***	Change notification. Determines what action is taken when a known source address is received on a port other than the one currently associated with the address: ❑ ***cnotify*** = 1. Change notification is enabled. The frame additionally is forwarded to the ports specified by ***UnkSrcPorts*** ANDed with the appropriate ***VLANnPorts*** register, provided that address learning is enabled for the port to which the address has moved (appropriate bit = 0 in ***NLearnPorts***). This allows the address-lookup tables of other switches to be updated in accordance with the change. ❑ ***cnotify*** = 0. Change notification is disabled.
7	***nage***	No aging. Allows automatic address-record aging to be disabled independent of learning. (***nauto*** = 1 disables both learning and aging). ❑ ***nage*** = 1. Aging is disabled even if ***nauto*** = 0. ❑ ***nage*** = 0. Aging is enabled unless ***nauto*** = 1. If ***nauto*** = 1, the value of ***nage*** is irrelevant – aging is disabled. This bit has no effect on address learning or time-stamp updating.
6	***dmainc***	DMA address autoincrement. When accessing the DIO interface with a DMA controller (**$\overline{\text{SDMA}}$** pin low), this bit determines whether the address held in ***dmaaddress*** should be incremented between successive accesses. ❑ ***dmainc*** = 1. The address increments between accesses. ❑ ***dmainc*** = 0. The address does not increment between accesses.
5	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
4	***unkvlan***	Unknown VLAN forwarding. This bit enables forwarding of multicast frames that have an unknown IEEE Std 802.1q VLAN Identifier (i.e., one that matches none of the VLAN IDs registered in ***VLAN0QID***–***VLAN63QID***). ❑ ***unkvlan*** = 1. All multicast frames belonging to unknown VLANs are forwarded to the port indicated in ***UnkVLANPort***. ❑ ***unkvlan*** = 0. All multicast frames belonging to unknown VLANs are discarded.

Table 4–1. SysControl (System Control Register) (Continued)

Bit	Name	Description
3	***mirr***	Port mirroring. This bit enables port mirroring. ❑ ***mirr*** = 1. All frames received on or sent to the port specified in ***MirrorPort*** are copied to the port specified in ***UplinkPort.*** ❑ ***mirr*** = 0. No mirroring is performed.
2	***nauto***	Not automatically add address mode. This bit selects the manner in which addresses are added to the address records. ❑ ***nauto*** = 1. Addresses can be added only to records using DIO adds. The aging-state machine is disabled; it is the external CPU's responsibility to manage the address records. ❑ ***nauto*** = 0. New addresses are learned from the wire and added to the address tables automatically. (They also can be added using DIO adds). However, if ***nage*** = 1 and the address tables become full, further addresses are not added unless space is created first using DIO deletes.
1	***ncrc***	No CRC check. This bit determines whether addresses are learned if the frame containing them contains an invalid CRC. ❑ ***ncrc*** = 1. Unknown addresses are added to the address-lookup table without regard to the validity of the CRC. ❑ ***ncrc*** = 0. Unknown addresses are added to the address-lookup table only if the frame containing the address has a valid CRC.
0	***flow***	Flow control enable. This bit enables flow control. ❑ ***flow*** = 1. The device implements flow control. Ports that are configured for IEEE Std 802.3x use pause frames. Ports that are not configured for IEEE Std 802.3x use collisions. ❑ ***flow*** = 0. No flow control is implemented.

It is strongly recommended that the ***start*** bit in ***SysControl*** be set to 1 after the external CPU has properly configured the TNETX4020 internal registers. The device initializes the buffer and address-lookup memory after the ***start*** bit is set to 1. During this time, the TNETX4020 holds off the external CPU, i.e., no DIO accesses are allowed. As an indication that the initialization buffer and address-lookup memory are initialized successfully, the ***initd*** bit is set to 1 and the ***start*** bit is set to 0.

Before setting the start bit to 1, the external CPU must wait for the RDRAM to be initialized successfully. The ***rdinit*** = 1 in the ***RAMStatus*** register indicates that the initialization of the RDRAM was successful.

RAMStatus (RAM Status Register) @ 0x00E3

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved						***parity***	***rdinit***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	h	h	
Field Type	r	r	r	r	r	r	r	r	

Table 4–2. RAMStatus (RAM Status Register)

Bit	Name	Description
7:2	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
1	***parity***	RDRAM parity enabled. This bit indicates whether the RDRAM supports data protection via setting and checking parity: ❑ ***parity*** = 1. The RDRAMs in the system have 9-bit bytes, and data is provided by the setting and checking of parity. ❑ ***parity*** = 0. The RDRAMs in the system have 8-bit bytes; therefore, no data protection is provided. This bit is not valid unless ***rdinit*** is set.
0	***rdinit***	RDRAM initialized. This bit becomes set when the RDRAM initialization sequence (which occurs after a hard reset) completes successfully.

4.2 Port Configuration

The TNETX4020 consists of two 100-/1000-Mbit/s ports. The ports can have the following parameters, configured on a per-port basis:

- ***maxlen*** – maximum frame length (1518 bytes or 1535 bytes)
- ***rxacc*** and ***txacc*** – port awareness on ingress and egress
- ***disable*** – disabling a port and flushing the frames from the port queue
- ***speed*** – configuring the ports to operate either at 100 Mbit/s, or 1000 Mbit/s
- ***mode*** – PCS or (G)MII
- ***pause*** – configuring a port in full duplex to support IEEE Std 802.3x flow control
- ***neg*** – A port can be configured to advertise the port capability to the PHY via hardware or the port's capability can be forced.
- ***txpace*** – transmit pacing
- ***reqhd*** – selecting duplex
- ***tolerant*** – some flexibility on minimum packet size at 1000 Mbit/s

Port 1 can also be configured to support preframe tagging (see *Pretag*).

PortxControl (Control Register) @ 0x0000–0x0003

Ports 0–1 are 100-/1000-Mbit/s ports. x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	reserved		*tolerant*	*maxlen*	*txacc*	*rxacc*	*pretag*	*neg*	*disable*	*reqntxp*	*lckref*	*reqnrxp*	*reqpma*	*req100*	*txpace*	*reqhd*	DIO Address 0x0000 +0x2*x
Reset Value	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–3. PortxControl (Control Register)

Bit	Name	Description
15:14	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
13	***tolerant***	Frame-length tolerance. Determines the minimum frame length that is considered as a valid frame when operating in half-duplex mode at 1000 Mbit/s (***duplex*** = 0 and ***speed*** = 1 in ***PortxStatus***, to provide tolerance for slightly noncompliant IEEE Std 802.3z systems: ❑ ***tolerant*** = 0. Frames less than 512 bytes in length (including carrier extension) are discarded. ❑ ***tolerant*** = 1. Frames less than 510 bytes in length (including carrier extension) are discarded. This bit has no effect when ***duplex*** = 1 or ***speed*** = 0.
12	***maxlen***	Maximum frame length. Determines the maximum frame length at which a received frame is discarded. ❑ ***maxlen*** = 1. The maximum received frame length is 1518 bytes, as specified by IEEE Std 802.3. This is the maximum length on the wire. If a VLAN header is inserted into a 1518-byte frame within the MAC, the frame is stored as a 1522-byte frame within the switch. ❑ ***maxlen*** = 0. The maximum received frame length is 1535 bytes if no VLAN header is inserted, or 1531 bytes if a VLAN header is inserted. When stored within the switch, a frame never can be longer than 1535 bytes. This bit has no effect on the maximum frame size that can be transmitted by the port.
11	***txacc***	Transmit access. Identifies whether the port is connected to the network beyond the switching system (an access port), or to another similarly capable switch device within the system (an intra-switch port). In turn, this determines how the port handles VLANs during transmission. ❑ ***txacc*** = 1. The first IEEE Std 802.1q tag header in the frame is stripped from all frames transmitted from this port. ❑ ***txacc*** = 0. The first IEEE Std 802.1q tag header in the frame is stripped from the frame only if the VLAN ID within it is identical to ***vlanqid*** in the port's ***PortxQTag*** register.

Table 4–3. PortxControl (Control Register) (Continued)

Bit	Name	Description
10	***rxacc***	Receive access. Identifies whether the port is connected to the network beyond the switching system (an access port), or to another similarly capable switch device within the system (an intraswitch port). In turn, this determines how the port handles VLANs during reception. ❑ ***rxacc*** = 1. An IEEE Std 802.1q tag header equal to the port's ***PortxQTag*** register is added to all frames received on this port, even if they already include one. ❑ ***rxacc*** = 0. An IEEE Std 802.1q tag header equal to the port's ***PortxQTag*** register is added to all untagged frames received on this port. If the frame already contains an IEEE Std 802.1q tag header but its VLAN ID is 0x000, then the VLAN ID in the header is replaced by ***vlanqid*** from the port's ***PortxQTag*** register.
9	***pretag***	Preframe tagging. Controls support for frame tagging on this port. ❑ In MII and PMA mode, this bit always reads 0, and writes have no effect. ❑ ***pretag*** = 1. The device expects frames received on this port to have a pretag and adds a tag to all frames transmitted from this port. ❑ ***pretag*** = 0. Frames received on this port should not have a pretag. No pretag is added to transmitted frames. **Pretagging is supported only on port 1. Do not set this bit on port 0.** See the description of 100-/1000-Mbit/s PHY interface for details of tag formats and interpretation. This bit should not be set while the corresponding ***portvector*** bit in ***RingPorts*** also is a 1; otherwise, unpredictable behavior can result.
8	***neg***	Negotiation. In PMA mode: ❑ ***neg*** = 0. Autonegotiation is disabled and the mode of operation described by ***reqntxp, reqhd,*** and ***reqnrxp*** bits is adopted by the port. ❑ ***neg*** = 1. The on-chip PCS autonegotiates the pause and duplex capabilities described by ***reqntxp, reqhd,*** and ***reqnrxp*** bits across the PMA interface. ❑ Writing a 0 followed by a 1 to this bit forces the on-chip PCS to recommence autonegotiation even if a link already has been established and is active. This bit also can be accessed via ***enaneg*** in ***PCSxControl***. See the ***PCSxControl*** description. When not in PMA mode, this bit has no effect.
7	***disable***	Port disable. Writing a 1 to this bit disables the port. Frames are not forwarded from or to a disabled port. No learning occurs, although statistics pertaining to frame reception continue to operate normally. Any previously queued frames are taken off the transmit queue as if they were being transmitted, but they are not sent to the PHY, i.e., they are discarded. **Mxx_TXEN** is not asserted. The state of the MAC-PHY interface signals does not impede this queue-clearing process.

Table 4–3. PortxControl (Control Register) (Continued)

Bit	Name	Description
6	***reqntxp***	Request no transmit pause. The effect of this bit depends on the mode of operation for which the port is configured, as determined by ***speed*** and ***pma*** in ***PortxStatus***: ❑ ***reqntxp*** = 1. The PCS layer advertises an inability to transmit IEEE Std 802.3x pause frames during autonegotiation. ❑ ***reqntxp*** = 0. The PCS layer advertises an ability to transmit IEEE Std 802.3x pause frames during autonegotiation. When configured for GMII mode (***speed*** = 1, ***pma*** = 0): ❑ ***reqntxp*** = 1. Transmission of IEEE Std 802.3x pause frames is disabled for this port. ❑ ***reqntxp*** = 0. Transmission of IEEE Std 802.3x pause frames is supported for this port. When configured for MII mode (***speed*** = 0): ❑ ***reqntxp*** = 1. Transmission of IEEE Std 802.3x pause frames is disabled for this port, and pause frames received on this port are treated the same as any other multicast frame. The open-drain **Mxx_RXD5** pin is asserted low. ❑ ***reqntxp*** = 0. IEEE Std 802.3x pause frames are supported for this port, unless **Mxx_RXD5** is pulled low by the attached PHY or PMI device. Alternatively, ***reqntxp*** can be read or written via ***pause*** and ***asym*** in ***PCSxANAdvert***, but the encoding is different. See the ***PCSxANAdvert*** definition for details.
5	***lckref***	Lock to reference. The inverse value of this bit directly controls the **$\overline{\text{Mxx\_LREF}}$** pin, when hardware reset is not being applied. (**$\overline{\text{Mxx\_LREF}}$** is active during hardware reset.) The pin is attached to an external PMA sublayer device and causes it to lock to its reference clock.
4	***reqnrxp***	Request no receive pause. The effect of this bit depends on the mode of operation for which the port is configured, as determined by ***speed*** and ***pma*** in ***PortxStatus***. When configured for PMA mode (***pma*** = 1): ❑ ***reqnrxp*** = 1. The PCS layer advertises an inability to respond to IEEE Std 802.3x pause frames during autonegotiation. ❑ ***reqnrxp*** = 0. The PCS layer advertises an ability to respond to IEEE Std 802.3x pause frames during autonegotiation. When configured for GMII mode (***speed*** = 1, ***pma*** = 0): ❑ ***reqnrxp*** = 1. IEEE Std 802.3x pause frames received on this port are ignored, but these pause frames are absorbed by the MAC. ❑ ***reqnrxp*** = 0. Reception of IEEE Std 802.3x pause frames is supported for this port. When configured for 100 Mbit/s (***speed*** = 0): ❑ ***reqnrxp*** = 1. Transmission of IEEE Std 802.3x pause frames is disabled for this port. Pause frames received on this port are ignored and absorbed by the MAC. The open-drain **Mxx_RXD5** pin is asserted low. ❑ ***reqnrxp*** = 0. IEEE Std 802.3x pause frames are supported for this port, unless **Mxx_RXD5** is pulled low by the attached PHY or PMI device. Alternatively, ***reqnrxp*** can be read or written via ***pause*** and ***asym*** in ***PCSxANAdvert***, but the encoding is different. See the ***PCSxANAdvert*** definition for details.

Table 4–3. PortxControl (Control Register) (Continued)

Bit	Name	Description
3	***reqpma***	Request PMA mode. When high, the port is configured to interface to an external PMA device and the open-drain **Mxx_PMA** pin is pulled low. When low, the port is configured to interface only to an external PMA device if the **Mxx_PMA** pin is pulled low by external hardware.
2	***req100***	Request 100 Mbit/s. Controls the speed of the port when not in PMA mode. ❑ ***req100*** = 1. The port is configured for 100-Mbit/s operation, and **Mxx_MII** is pulled low. ❑ ***req100*** = 0. The port is configured for 1000-Mbit/s operation unless **Mxx_MII** is pulled low externally. This bit has no effect if the port is configured in PMA mode.
1	***txpace***	Transmit pacing. When high, the port uses transmission pacing to enhance performance.
0	***reqhd***	Request half duplex. When high, this bit puts the port in half-duplex mode. In MII mode, **Mxx_RXD4** also is asserted low. When low, the effect of this bit depends on the mode of operation for which the port is configured: ❑ When in GMII mode and ***reqhd*** = 0, the port is configured in full-duplex mode. ❑ When in MII mode and ***reqhd*** = 0, the port is configured for full-duplex mode unless **Mxx_RXD4** is pulled low externally. ❑ When in PMA mode and ***reqhd*** = 1, the PCS layer attempts to negotiate half-duplex operation with the attached PMD device. The result of this negotiation can be determined by reading the ***duplex*** bit in the ***PortxStatus*** register. Under certain conditions, this bit can be modified by writing to ***pcsdp*** in ***PCSxControl*** and ***canfdp*** in ***PCSxANAdvert***, and writes to those bits can change the value of ***reqhd***. See the ***PCSxControl*** and ***PCSxANAdvert*** descriptions.

4.2.1 Port Awareness

The ***PortxControl*** register access bits, ***rxacc*** and ***txacc***, allow the system to configure ports to be aware that they are a part of a larger switch (intraswitch port) or connected to the network (access port). The access bits, ***rxacc*** and ***txacc***, determine if a VLAN tag is added, modified, or retained on ingress and stripped or retained on egress.

The VLAN tag that the TNETX4020 adds (or modifies) on ingress is a user-programmable VLAN tag contained in the ***PortxQTag*** registers.

Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register) @ 0x0380–0x0383

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0380 + 0x2*x
Field Name	reserved				***vlanqid[11:0]***												
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	–	–	–	–	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–4. Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register)

Bit	Name	Description
15:12	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
11:0	***vlanqid***	IEEE Std 802.1q VLAN identifier. Identifies the default IEEE Std 802.1q VLAN ID associated with each port. How it is used depends on the value of ***rxacc*** in ***PortxControl***: ❑ ***rxacc*** = 1. An IEEE Std 802.1q tag header always is added to the frame (even if the frame already is tagged). The default source port VLAN ID contained in this field is used. ❑ ***rxacc*** = 0. If a received frame contains no IEEE Std 802.1q tag header, a tag header is added using the default source port VLAN ID contained in this field. If the frame has a tag header with an ID of 0x000, the VLAN tag is replaced by this field.

Tables 4–5 and 4–6 describe how a port can be configured to operate when adding, retaining, or modifying a frame's IEEE Std 802.1q header.

Table 4–5. Ingress

IEEE Std 802.1q Frame Used	***rxacc*** = 0	***rxacc*** = 1
No	Header added from ***PortxQTag***	Header added from ***PortxQTag***
Yes VLAN ID = 0x000	Header modified with ***PortxQTag***	Header added from ***PortxQTag***
Yes VLAN ID ≠ 0x000	No header added or modified	Header added from ***PortxQTag***

Table 4–6. Egress

Frame VLAN ID = ***PortxQTag***	***txacc*** = 0	***txacc*** = 1
Yes	Strip header before transmit	Strip header before transmit
No	Retain header	Strip header before transmit

4.2.2 Maximum Frame Length

The maximum frame-length bit, ***maxlen*** in each port's ***PortxControl*** register, affects the limit on the longest frame that the switch port accepts without discarding the frame. The default state, 0, allows the switch to use an integer number of internal buffers, and to set the maximum internal frame size to 1535 bytes. This translates to 1535 bytes if the incoming frame does not have a VLAN tag inserted, or 1531 bytes if it does have a VLAN tag inserted. At this ***maxlen*** setting, the port accepts frames larger than that allowed by IEEE Std 802.1q.

If ***maxlen*** is set to 1, the port discards frames larger than 1518 bytes. The frame is allowed to be 1522 bytes internally, a concession for the insertion of a VLAN tag into the frame. The internal representation of frames in this device always has a VLAN tag; if a valid one is not present in a frame on ingress, one is created using the default tag information programmed into the port (see description of the ***raxcc*** bit, ***PortxControl*** register). If VLAN tags are not being supported on the egress port, there is provision for always stripping out a tag (see description of the ***txacc*** bit, ***PortxControl*** register).

If this device is expected to transport maximum length frames with VLAN tags already inserted, ***maxlen*** should be set to 0 to avoid discarding these packets, which can exceed 1518 bytes.

If frames are output onto a network that does not support VLAN tags, that port's ***txacc*** bit should be set to 1, and the other port's ***rxacc*** bit bit set to 0 to make sure each packet only has one VLAN tag, and it is stripped on transmit from the switch. This supports the legacy limit of a 1518-byte frame length.

4.2.3 Port Disable

Configuring the port disable bit ***(disable)*** to 1, disables the port. Frames are not forwarded from or to a disabled port. Frames queued on a particular port that is disabled are discarded.

The ***disable*** bit can be used to remove head-of-line blocking for a particular port by flushing the port's frame queue. When setting the ***disable*** bit to 1 in a port's control register, the frames queued up on that port are discarded. The ***txqueue*** bit in the port's status register are set to 0 when all frames have been removed from the queue. Hence, to flush the queue of a port, the external CPU must set the ***disable*** bit in the port's control register. The external CPU can then poll the ***txqueue*** bit in the port's status register, which is set to 0 when the queue has been flushed. Frames destined for a port that is disabled are discarded automatically.

4.2.4 Port Capability

The two gigabit ports, when configured in 100-Mbit/s mode, can be configured to have the following capabilities:

Speed	Duplex	Pause
100 Mbit/s	HD	N/A
100 Mbit/s	FD	No pause
100 Mbit/s	FD	Pause

The 100-/1000-Mbit/s port in gigabit-mode configurations:

Speed	Duplex	Tx Pause	Rx Pause
1000 Mbit/s	HD	N/A	N/A
1000 Mbit/s	FD	No Tx pause	No Rx pause
1000 Mbit/s	FD	Tx pause	No Rx Pause
1000 Mbit/s	FD	No Tx pause	Rx pause
1000 Mbit/s	FD	Tx pause	Rx pause

When configured in gigabit mode, the 100-/1000-Mbit/s port flow control is configured asymmetrically. That is, the port can be configured to respond only to received pause frames, while not generating Tx pause frames and vice versa. The port also can be configured for symmetrical pause by configuring the port to respond to pause frames and to transmit pause frames when the TNETX4020 is congested.

Note:

When the 100-/1000-Mbit/s port is configured for 100-Mbit/s mode, IEEE Std 802.3x flow control can only be configured symmetrically.

4.2.5 Flow Control

The TNETX4020 supports collision-based flow control for ports in half-duplex mode and IEEE Std 802.3x flow control for ports in full-duplex mode. The ***flow*** bit in the ***SysControl*** register determines the action to be taken when back pressure is needed, that is, when there are insufficient resources to handle an inbound packet. The ***holb*** bit in the ***SysControl*** register determines when back pressure is needed.

If ***flow*** = 0, packets are discarded at the ingress port when insufficient resources are available to handle them.

If ***flow*** = 1, ports in half-duplex mode cause collisions to avoid accepting packets, and ports in full-duplex mode whose link partners negotiated to accept pause packets will send them, otherwise, packets are dropped. If Port 0 or 1 is in MII mode, the pause-frame transmission/reception is required to be symmetric. If in GMII or PMA mode, transmit and receive pause capabilities are negotiated independently.

With ***holb*** = 0, back pressure is applied to all ports when the number of buffers in the global pool is down to the value in the ***FlowThreshold*** register. (The external pins for a gigabit-capable port in any mode, **Mxx_FLOW**, go active when the port gets down to half ***FlowThreshold***. This feature was intended to let a high-speed ringed-port run after other slower ports were flow controlled on the TNETX4090.) This prevents the reception of more frames at any port until the frame backlog is reduced and the number of free buffers has risen above the threshold. When this happens, back pressure is removed from all ports and packets can be received. The value in ***FlowThreshold*** should be set so that all ports can complete reception of a maximum-size frame, i.e., each port should have enough time to activate the flow mechanisms without dumping a frame for which reception has started.

If ***holb*** = 1, back pressure is applied as when ***holb*** = 0, or to an individual port when the buffers held in memory for data that arrived on that port is greater than the available pool remaining assuming that ***FlowThreshold*** is set small enough that this mechanism does not affect the back pressure in this mode. An example is:

> When port A's traffic begins to backlog in memory (no matter what port(s) it is destined to), back pressure is applied when the amount of data backed up is greater than the available pool (about half the buffers are assigned to data from port A). If A's data stays backlogged and if data arriving at port B also begins to backlog in memory, back pressure is applied to port B when its data amounts to one-fourth of the buffer pool, or one-half of the half left after port A had back pressure applied. When port A's traffic begins to exit the switch, port A stays back pressured until its data is equal to one-third of the total. As buffers become available, port B is allowed to consume up to one-third of the buffer pool (each backlogged total is compared with the buffers available). In this mode, only the stations that have caused their fair share of buffers to be removed from the available pool are back pressured.

Setting ***holb*** = 1 activates circuitry that attempts to prevent a backlogged conversation stalling other port traffic by using all the memory buffers. Because the number of buffers charged to a particular port always is compared with the number of buffers remaining, there is no threshold register for this mode.

4.2.5.1 Other Flow-Control Mechanisms

4.2.5.1.1 Hardware Flow Control

If a port were in MII or GMII mode and full duplex, normally, its **Mxx_COL** would not be needed. Hardware flow control has been added by preventing the start of transmission of a frame if the **Mxx_COL** is high. This is useful in ring mode; the **Mxx_COL** can be tied to the **Mxx_FLOW** of the upstream neighbor for hardware flow control.

4.2.5.1.2 Multicast Limit

Because buffer resources for multicast (or broadcast) frames are released only when the last port has transmitted the frame, multicast packets to fast and slow ports are released at the slow-port rate. Multicast traffic from a fast port to a slow port could cause back pressure to be applied to the source port, blocking input from that port. This occurs even if at least one of the ports in the multicast could have kept pace with the inbound multicast packets. The ***McastLimit*** register can limit the number of multicast packets allowed to be pending at any output port. When the limit is reached, that port is not added to the routing vector of the next multicast packet for which it was eligible. Eligibility is restored when the number of backlogged packets is again below the limit. The ***McastLimit*** register allows the user to set the ports subject to this restriction, and set the limit in 8 binary steps from 2 to 256. The spanning-tree BPDU multicast packet is exempt from this limit.

4.2.5.2 Other Behavior Changes Resulting From Flow-Control Bits or Pins

Clearing (the default) ***holbrm*** in the ***RingPorts*** register includes the test for buffer use controlled by the ***holb*** bit in ***SysControl*** into the action of the **Mxx_FLOW** pin. **Mxx_FLOW** goes high if either the number of buffers left is fewer than the ***FlowThreshold*** value or a ring-mode port receive operation has exceeded its fair share of buffers. Setting ***holbrm*** = 1 makes **Mxx_FLOW** respond only to ***FlowThreshold*** value violations.

FreeStackLength (Free Stack Length Register) @ 0x080C–0x080E

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***freestacklength[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved								
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table 4–7. FreeStackLength (Free Stack Length Register)

Bit	Name	Description
23:16	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
15:0	***freestacklength***	Free stack length. This register contains the number of 1K-byte buffers currently on the free stack, i.e., the number of buffers not currently used for storing backlogged frames. The free stack itself consumes 1K bytes per 512K bytes of memory. This register always should be read least significant byte first; reading this location causes the register contents to be transferred to a holding latch to prevent the value changing while it is being read.

FlowThreshold (Flow Control Threshold Register) @ 0x00F0–0x00F2

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***flowthreshold[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved								
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table 4–8. FlowThreshold (Flow Control Threshold Register)

Bit	Name	Description
23:16	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
15:0	***flowthreshold***	Flow-control threshold value. The ***flow*** bit must be set to 1 in ***SysControl*** to enable this feature. ❑ When the number of free buffers in external memory that are available for frame reception drops below this value, flow control will be initiated in the ports (both Ethernet and NM port). ❑ When the number of free buffers in external memory that are available for frame reception drops below one-half of this value, both Mxx_FLOW pins are asserted. The **Mxx_FLOW** pins cease to be asserted when the free-buffer count equals or exceeds one-half the flowthreshold value, the MAC/NMPORT flow control ceases when the free-buffer count equals or exceeds the flowthreshold value. Depending on the amount of memory in the system, it may be possible to program this value to be greater than the total number of buffers in the system, thereby permanently inducing flow control.

Note:

The purpose of flow control is to reduce the risk of data loss due to long bursts of traffic. However, there is no way to prevent frame reception on those ports operating in full-duplex mode that have not negotiated IEEE Std 802.3x flow control. Ports that have not negotiated IEEE Std 802.3x flow control may exhaust the free-buffer queue with subsequent data loss. Moreover, even if a port is configured for IEEE Std 802.3x flow control, the port continues to receive frames, even during congestion, if the other port does not respond to the transmitted pause frames.

McastLimit (Transmit Queue Multicast Limit Register) @ 0x00DC–0x00DF (DIO)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***enable[2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***threshold [2:0]***			reserved													
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	rwl	rwl	rwl	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–9. McastLimit (Transmit Queue Mcast Limit Register)

Bit	Name	Description
31:29	***threshold***	Multicast limit threshold. This value is used to calculate the number of queued multicast frames any port can have queued waiting for transmission. The limit is calculated as $2^{(n+1)}$ multicast frames (where n = ***threshold***). ***threshold [2:0]*** / **Multicast Frame Limit** 000 — 2 001 — 4 010 — 8 011 — 16 100 — 32 101 — 64 110 — 128 111 — 256 Each port maintains a count of the number of outstanding multicast frames queued. When this count exceeds the limit set, and the multicast limit function is enabled, subsequent multicast frames are not queued to that port. BPDU multicast frames recognized by the IALE are not affected.
28:10	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
9:3	reserved	Reserved. These bits may be read and written. These bits have no effect.
2:0	***enable***	Multicast limit enable portvector. The ***enable*** portvector determines the transmit ports that take part in transmit-queue multicast limiting.

4.2.5.3 Collision-Based Flow Control

Collision-based flow control is initiated on all ports configured in half duplex when the buffer threshold is broken and flow control has been enabled. Ports that are not transmitting currently generate a collision by sending a jam signal when they start to receive a frame. The collisions generated by the flow control logic are not limited to a maximum of 16 consecutive collisions and are independent of the normal backoff algorithm. If the backlog lasts more than 16 collisions, IEEE Std 802.3 requires the transmitting port to abandon that packet and move to the next.

4.2.5.4 IEEE Std 802.3x Flow Control

IEEE Std 802.3x flow control is initiated on all ports configured for pause and full-duplex mode when the buffer threshold is broken and flow control has been enabled. These ports transmit a pause frame, which requests the peer node to pause for a user-programmable amount of time. If the numbers of free buffers remain below the threshold after 80% of pause time has elapsed, another pause frame is transmitted at the earliest opportunity.

The user-programmable pause time contained in the transmitted pause frame is configured in ***PauseTime100*** and ***PauseTime1000*** registers. Depending on the speed configured for the port, the corresponding value from the pause-time register is included in the pause frame. For more information about pause frames, refer to IEEE Std 802.3x.

PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s)) @ 0x00EA–0x00EB

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***pausetime[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–10. PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s))

Bit	Name	Description
15:0	***pausetime***	IEEE Std 802.3x pause_time. This value defines the 16-bit unsigned integer containing the length of time for which the receiving station is requested to inhibit data-frame transmission. The pause time is measured in units of pause quanta, equal to 512-bit times of the particular implementation. The range of possible pause time is 0 to 65535 pause quanta. If this value is set to 0, pause frames never will be issued, even if flow control is enabled and the switch is congested. The default value of 0x1 is for device testing; it needs to be initialized by the host CPU or by EEPROM load to 0x0 or greater than 0x3F.

PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s)) @ 0x00EE–0x00EF

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***pausetime[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–11. PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s))

Bit	Name	Description
15:0	***pausetime***	IEEE Std 802.3x pause_time. This value defines the 16-bit unsigned integer containing the length of time for which the receiving station is requested to inhibit data-frame transmission. The pause time is measured in units of pause quanta, equal to 512-bit times of the particular implementation. The range of possible pause time is 0 to 65535 pause quanta. If this value is set to 0, pause frames never will be issued, even if flow control is enabled and the switch is congested. The default value of 0x1 is for device testing; it needs to be initialized by the host CPU or by EEPROM load to 0x0 or greater than 0x3F.

It is recommended that ***pausetime*** values be configured according to the speed, such that the pause times for 100 Mbit/s and 1000 Mbit/s are the same. The ratios between the ***pausetime*** of the two registers would be 1:10, using 100 Mbit/s as the base.

The source address of a pause frame is that of the 48-bit address field in the ***DevNode*** register. The external CPU should configure the MAC address of the system in the ***DevNode*** register. If an address is not configured, the default value of 0x00.00.00.00.00.00 is used as the source address in IEEE Std 802.3x pause frames generated by the MAC of a particular port.

DevNode (Device Node Register) @ 0x00A4–0x00A9

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	–	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	r	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	–	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–12. DevNode (Device Node Register)

Bit	Name	Description
47:0	***address***	Device node address. This register is used to specify the 48-bit address that is used by the MAC control sublayer as: ❑ the source address in transmitted pause frames, and ❑ one of the two valid destination addresses for pause frames (the other address being the reserved multicast address 01.80.c2.00.00.01). This register is not used for any other purpose. The address is stored in native Ethernet format. Bit 40 of this address, bit 0 of this register, is hardwired to 0 to prevent a multicast address from being entered. If the value contained in this register is desired to be used as the destination address for data frames, for example to cause frames to be routed to the NM port, this value must be added explicitly to the IALE's address records via a DIO add, or by learning from a frame received by the NM port. Until this is done, IALE treats this address as an unknown unicast address, i.e., the contents of this register are not implicitly known by the IALE.

Note:

The transmitted pause frames are only a request to the peer node to stop transmitting. Frames that are received during the pause interval are received and forwarded as during normal operation, provided the external memory (packet memory) is not full.

4.2.6 Advertisement of Port Capability

The negotiation ***(neg)*** bit of the ***PortxControl*** registers indicates if the port's capability as configured in the ***PortxControl*** register is advertised to the attached PHY via hardware duplex and pause pins or the RXD and TXD pins when in PMA mode. This bit has no effect in MII or GMII modes.

4.2.7 Extended-Port Awareness

By setting the ***pretag*** bit in the ***Port1Control*** register to 1, the gigabit port supports an in-band pretag that allows the IALE to become aware of extended ports in the system. This feature allows this gigabit port to be connected to a crossbar-matrix switch with up 17 1000-Mbit/s ports. By making the TNETX4020 aware of the ports on the crossbar-matrix switch, the crossbar-matrix switch need not make any forwarding or filtering decisions, and can be relatively simple and inexpensive to design. See the *Extended Port Awareness* application report, literature number SPWA018, for details.

Note:

The preframe tagging supported when ***pretag*** is set to 1 and the preframe tagging used in cascading-ring topology must not be confused. The TNETX4020 cannot be configured to support both preframe tagging schemes at the same time. The features are mutually exclusive.

The ***pretag*** for extended-port awareness can be set only on Port 1. Preframe tagging for ring mode, enabled via the ***RingPorts*** register at 0x8C, is available on both ports.

4.2.8 Transmit Pacing

Transmit pacing or adaptive performance optimization (APO) can be enabled for each port by configuring the ***txpace*** bit in the ***PortxControl*** register to 1. When APO is enabled, the MAC uses an extra delay in the IPG after transmitting a packet to reduce collisions in a busy network. The transmit-pacing feature is most effective when connected on networks using transmit-pacing-capable MACs.

4.2.9 PCS Function

Configuring the ***reqpma*** bit in the ***PortxControl*** register to a 1 alters the port's interface from GMII to PCS. The PCS function is performed on chip and the port can connect to a gigabit transceiver, e.g., SERDES.

When the PCS function is performed on-chip, it can be configured via the following PCS-specific registers: ***PCSxControl***, ***PCSxStatus***, ***PCSxANAdvert***, ***PCSxANLinkP***, ***PCSxANExp***, ***PCSxANNxt***, ***PCSxANLinkPNxt***, and ***PCSxExtStatus*** registers. (See Chapter 3 for detailed descriptions of the registers.) The ***PortxControl*** register still must be used to configure the gigabit MAC of port 0 or 1. The gigabit MAC advertises its configured capability internally to the PCS function, which performs the autonegotiation. The result of the autonegotiation is reflected in the ***PCSxStatus*** and ***PCSxExtStatus*** registers, as well as in the ***PortxStatus*** register.

PortxStatus (Port x Status Register) @ 0x0400–0x0403

Port x is a 100-/1000-Mbit/s port, x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	reserved							***pma***	***txpactv***	***nlink***	***pausetx***	***speed***	***duplex***	***pauserx***	***txqueue***	***linkchng***	DIO Address 0x0400 + 0x2*x
Reset Value	0	0	0	0	0	0	0	0	0†	–	–	–	–	–	–	0	
Reset Type	–	–	–	–	–	–	–	–	hs	–	–	–	–	–	–	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rc	

† A synchronized reset. The TNETX4020 clocks must be running for this value to appear in the register.

Table 4–13. PortxStatus (Port x Status Register)

Bit	Name	Description
15:9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***pma***	PMA mode. This bit indicates whether the on-chip PCS layer is enabled. ❑ ***pma*** = 1. It indicates enabled. ❑ ***pma*** = 0. It indicates disabled. The value of this bit reflects the inverse of the **$\overline{\text{Mxx\_PMA}}$** pin.
7	***txpactv***	Transmit pause active. When this bit is 1, the port has received a pause frame and its pause-time period is being observed. The port does not start transmitting a new data frame while this bit is a 1. Any frame in midtransmission when this bit becomes a 1 is transmitted fully. The transmission of pause frames is unaffected by this bit's value.
6	***nlink***	No link. This bit indicates the link state of the network port. ❑ ***nlink*** = 1. The link is inactive. ❑ ***nlink*** = 0. The link is active. The value of this bit is determined in one of two ways: ❑ ***pma*** = 1. This bit reports the inverse of ***lkstatus*** in ***PCSxStatus***. ❑ ***pma*** = 0. This bit reports the inverse of the state of the port's **Mxx_LINK** pin.

Table 4–13. PortxStatus (Port x Status Register) (Continued)

Bit	Name	Description
5	***pausetx***	Transmit pause frame support. This bit indicates whether the port can generate IEEE Std 802.3x pause frames. ❑ ***pausetx*** = 1. IEEE Std 802.3x pause frames can be generated, provided ***flow*** = 1 in ***SysControl***. ❑ ***pausetx*** = 0. IEEE Std 802.3x pause frames cannot be generated. The value of this bit is determined in one of several ways. ❑ ***speed*** = 1 and ***pma*** = 0. This bit reflects the inverse of ***reqntxp*** in ***PortxControl***. ❑ ***speed*** = 0. This bit reflects the continuously sampled value of the **Mxx_RXD5** pin. ❑ ***pma*** = 1. This bit and ***pauserx*** are set according to the capabilities negotiated (***neg*** = 1) or advertised (***neg*** = 0) by the PCS layer. This is determined by ANDing ***pause*** and ***asym*** in ***PCSxANAdvert*** with ***lppause*** and ***lpasym,*** respectively, in ***PCSxANLinkP***. ■ When ***neg*** = 1, this is determined by ANDing ***pause*** and ***asym*** in ***PCSxANAdvert*** with ***lppause*** and ***lpasym***, respectively, in ***PCSxANLinkP***. The mapping is as follows: ***pause*** AND ***lppause*** → ***pausetx***, ***pauserx*** 0 0 → 0 0 0 1 → 1 0 1 0 → 1 1 1 1 → 0 1 ■ When ***neg*** = 0, this is determined by ***pause*** and ***asym*** only: ***pause*** ***asym*** → ***pausetx*** ***pauserx*** 0 0 → 0 0 0 1 → 1 0 1 0 → 1 1 1 1 → 0 1
4	***speed***	Network speed. This bit indicates the speed of the network port: ❑ ***speed*** = 1. Indicates 1000 Mbit/s. ❑ ***speed*** = 0. Indicates 100 Mbit/s. The value of this bit is determined in one of two ways: ❑ ***pma*** = 1. This bit is 1. ❑ ***pma*** = 0. This bit reflects the value of the **$\overline{\text{Mxx\_MII}}$** pin.
3	***duplex***	Full duplex network. This bit indicates the duplex state of the network port. ❑ ***duplex*** = 1. Indicates full duplex. ❑ ***duplex*** = 0. Indicates half duplex. The value of this bit is determined in one of several ways: ❑ ***speed*** = 1 and ***pma*** = 0. This bit reflects the inverse of ***reqhd*** in ***PortxControl***. ❑ ***speed*** = 0. This bit reflects the continuously sampled value of the **Mxx_RXD4** pin. ❑ ***pma*** = 1. This bit reflects the value of ***fulldpx*** in ***PCSxExtStatus***.

Table 4–13. PortxStatus (Port x Status Register) (Continued)

Bit	Name	Description
2	***pauserx***	Pause-frame Rx support. This bit indicates whether the port can respond to IEEE Std 802.3x pause frames. In 100-Mbit/s mode (***speed*** = 0), this bit is identical to ***pausetx***. In 1000-Mbit/s mode (***speed*** = 1), this bit indicates whether the port will respond to IEEE Std 802.3x pause frames. The value of this bit is determined in one of two ways: ❑ ***pauserx*** = 1. IEEE Std 802.3x pause frames will be responded to. ❑ ***pauserx*** = 0. IEEE Std 802.3x pause frames will be ignored and absorbed by the MAC. The value of this bit is determined in one of several ways: ❑ ***speed*** = 1 and ***pma*** = 0. This bit reflects the inverse of ***reqnrxp*** in ***PortxControl***. ❑ ***speed*** = 0. This bit reflects the continuously sampled value of the **Mxx_RXD5** pin. ❑ ***pma*** = 1. This bit and ***pausetx*** are set according to the capabilities negotiated by the PCS layer. Refer to the ***pausetx*** description.
1	***txqueue***	Transmit data queued. This bit indicates whether any frame data remains to be transmitted on this port. ❑ ***txqueue*** = 1. The port has untransmitted frame data queued. ❑ ***txqueue*** = 0. The transmit queue is empty.
0	***linkchng***	Link change. This bit is set to a 1 if the port's ***nlink*** bit changes from a 1 to a 0 or from a 0 to a 1, indicating that the link has gone down or up, respectively. This bit is cleared to 0 when this register is read. If an ***nlink*** change attempts to set the bit at the same time as a read tries to clear it, the bit is set. This bit is logically ORed with the ***linkchng*** bits of all the ***PortxStatus*** registers to produce the ***link*** interrupt bit in the ***Int*** register.

Configuring the PCS function on-chip is much like configuring an MII PHY, except that, in this case, the registers are internal to the TNETX4020 and the registers are in compliance with IEEE Std 802.3z.

4.3 Internal Address-Lookup Engine (IALE) Configuration

The IALE performs the frame-routing algorithm shown in the flowchart in Figure 4–1. The registers listed in the flowchart can be used to configure the IALE to produce the desired frame-routing results. These registers support, among other things, spanning-tree algorithm, cascading-ring topology, VLAN configuration, and extended port awareness.

Figure 4–1. Frame-Routing Algorithm

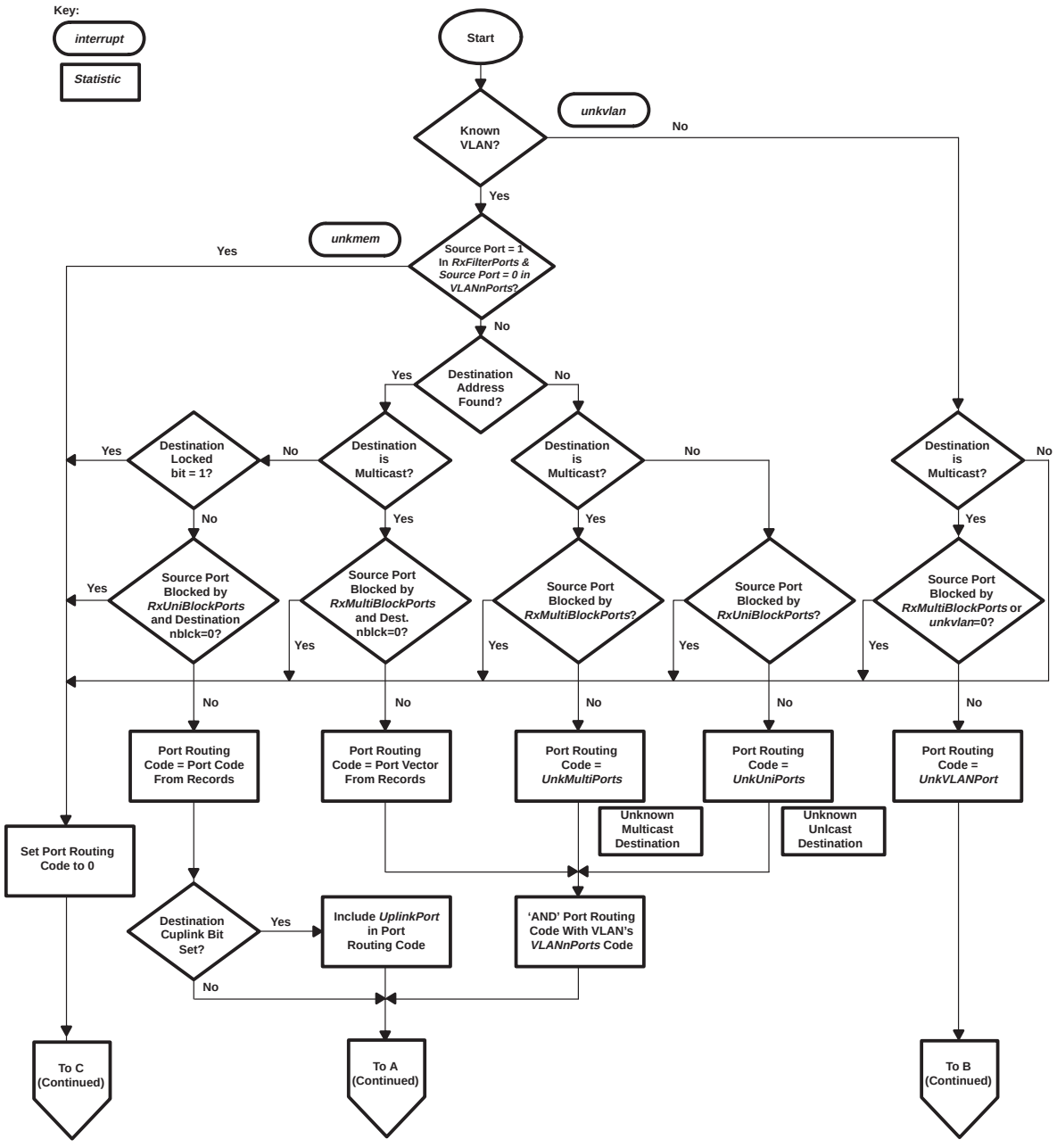


Figure 4–1. Frame-Routing Algorithm (Continued)

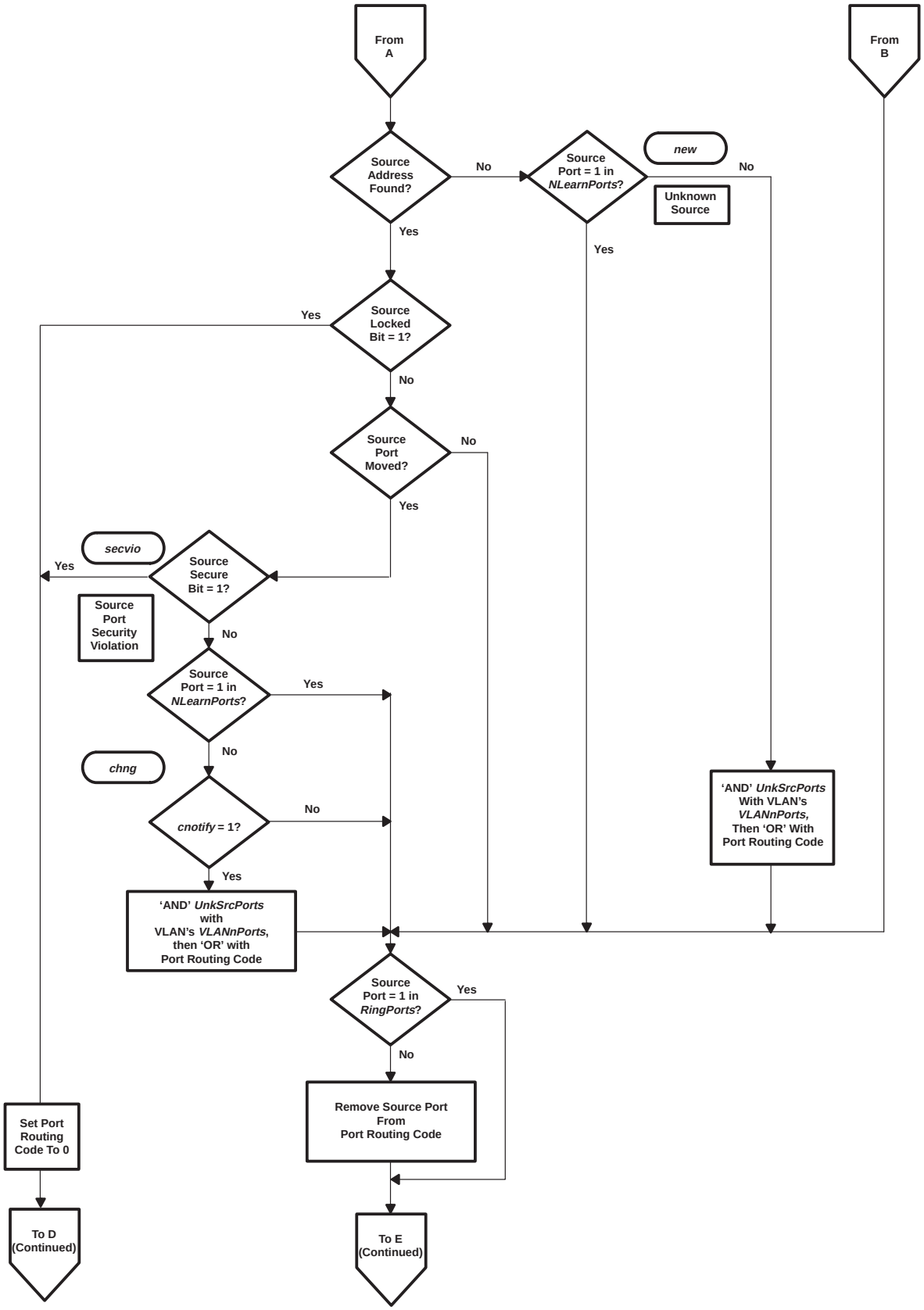
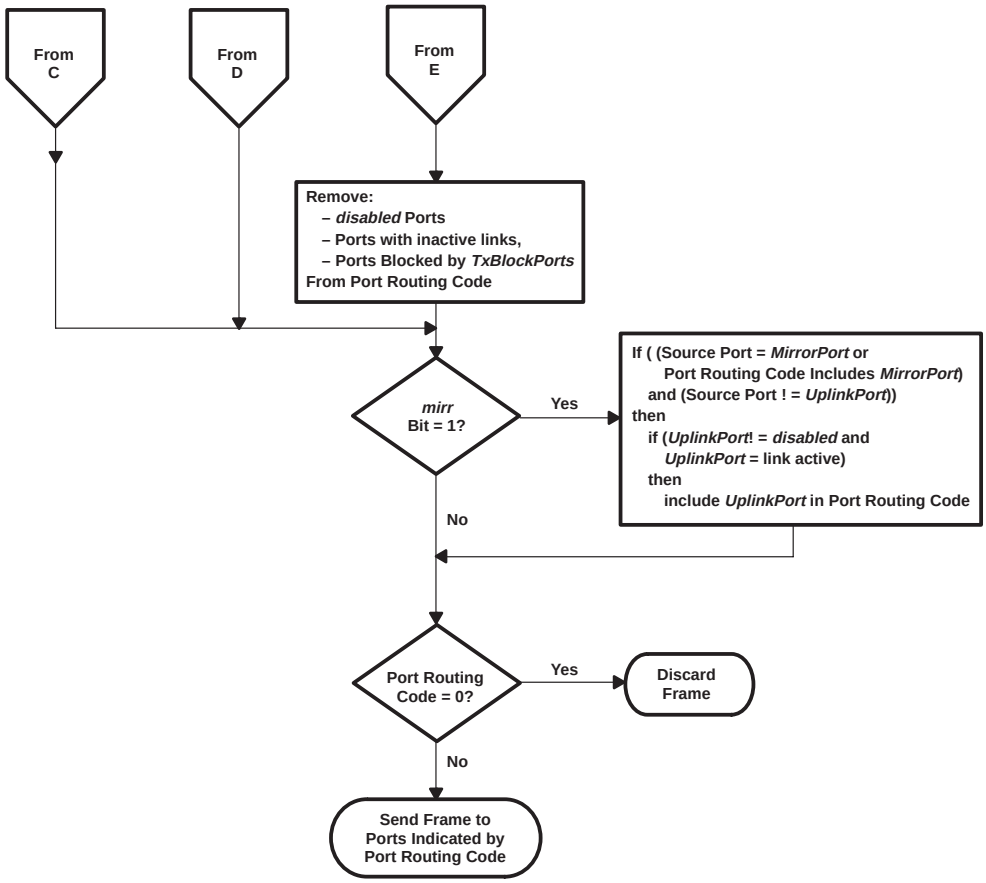


Figure 4–1. Frame-Routing Algorithm (Continued)



The IALE can be configured to automatically maintain the address records by learning addresses from the wire, updating addresses contained in the address records, and deleting addresses from the address records using one of two aging algorithms.

4.3.1 Automatic Address Maintenance

The default operation of TNETX4020 is automatic address maintenance. That is, the device learns and updates unicast addresses from the wire and deletes addresses from its address records by aging the perform address lookups for frame routing determined. Via the ***nauto*** and ***nage*** bits in the ***SysControl*** register, the external CPU can alter how addresses are learned and aged out.

The ***SysControl*** register is shown again, with focus on bits 2 and 7.

SysControl (System Control Register) @ 0x00FA–0x00FB

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***stop***	***load***	***start***	***initd***	***reboot***	***holb***	reserved	***cnotify***	***nage***	***dmainc***	reserved	***unkvlan***	***mirr***	***nauto***	***ncrc***	***flow***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	hs	h	h	–	h	h	h	–	h	h	h	h	h	
Field Type	rw	rw	rwl	r	rwl	rwl	r	rwl	rwl	rwl	r	rwl	rwl	rwl	rwl	rwl	

Table 4–14. SysControl (System Control Register)

Bit	Name	Description
15	***stop***	Stop system. When ***stop*** = 1, device operation is halted, and remains so until this bit is cleared to 0. All internal state machines, FIFOs, and MACs are reset. Register bits are cleared as per the register definition tables in this section. The host registers are not affected. Any data in the device is lost. ❑ This bit is set to 1 if an EEPROM load fails because of a bad CRC word. ❑ This bit is set to 1 if a parity error is detected on the external memory interface or an internal error.
14	***load***	Load system. Writing a 1 to this bit causes the DIO registers to be autoloaded from an external EEPROM (if present). Writing a 0 to this bit has no effect. Writing a 1 to this bit also clears the CRC-error indication from the fault LED. All DIO operations are inhibited (**SRDY** is held inactive high) while ***load*** is a 1 once the EEPROM state machine has finished checking that an EEPROM is present and begins to download data. ***load*** clears to 0 once the download is completed. When this bit is written with a 1, the ***stop*** bit simultaneously should be written with a 0.
13	***start***	Start system. Writing a 1 to this bit causes the device to begin operation, following a reset or stop. This bit is read as a 1 until buffer and address-lookup memory initialization is complete. This causes any previous frames or address records to be erased. While memory is being initialized, all ports are disabled. All DIO writes are inhibited (**SRDY** inactive high) while ***start*** is a 1. Writing a 1 to this bit also clears the parity-error indication from the fault LED. Writing a 0 to this bit has no effect; nor does writing a 1 when ***initd*** = 1. When this bit is written with a 1, the ***stop*** bit simultaneously should be written with a 0.
12	***initd***	Initialization done. This bit becomes a 1 after ***start*** has been set, and buffer and address-lookup memory initialization is complete (i.e., at the same time ***start*** is cleared). Once set, it remains so until a hard reset (pin or DIO), or until ***stop*** is set to 1. Until this bit is a 1, the ports ignore any frames they receive.
11	***reboot***	Reboot. When set to 1, reboot enables unmanaged systems to recover from fatal errors without manual intervention. ❑ ***reboot*** = 0. A fault causes ***stop*** to be set. ❑ ***reboot*** = 1. A fault causes a hard reset, thereby allowing the device configuration to be reloaded automatically from EEPROM and the device to be restarted. A fault occurs if an RDRAM parity error is detected or any of a variety of internal state consistency checks fails.

Table 4–14. SysControl (System Control Register) (Continued)

Bit	Name	Description
10	***holb***	Head-of-line blocking. When set, ***holb*** is enabled. This function prevents switch congestion on a channel from consuming all the memory resources, causing other channels to be affected. ❑ If ***flow*** is disabled with ***holb*** enabled, the switch discards frames after the ***holb*** threshold has been exceeded (with the exception of the NMPORT). ❑ If ***flow*** is enabled with ***holb***, the switch attempts to preserve frames already in the fabric and prevent further submission on RX ports, contributing to the congestion. The ***FLOW*** LED illuminates to indicate that a flow request was made. Where it is not possible to use flow control, frames are discarded from the Rx FIFO and counted as an overrun. When the NMPort enters ***holb***, the ***NumFreeBufs*** field in ***NMRxControl***, is held at zero after the next ***eof*** bit write, until the condition clears, preventing further frame submission from the host computer. No frames are discarded that are received from the NMPORT. When the ***holb*** threshold is exceeded and the ***flow*** bit is set, the appropriate ***Mxx_FLOW*** pin on the port that entered ***holb*** is asserted.
9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***cnotify***	Change notification. Determines what action is taken when a known source address is received on a port other than the one currently associated with the address: ❑ ***cnotify*** = 1. Change notification is enabled. The frame additionally is forwarded to the ports specified by ***UnkSrcPorts*** ANDed with the appropriate ***VLANnPorts*** register, provided that address learning is enabled for the port to which the address has moved (appropriate bit = 0 in ***NLearnPorts***). This allows the address-lookup tables of other switches to be updated in accordance with the change. ❑ ***cnotify*** = 0. Change notification is disabled.
7	***nage***	No aging. Allows automatic address-record aging to be disabled independent of learning. (***nauto*** = 1disables both learning and aging). ❑ ***nage*** = 1. Aging is disabled even if ***nauto*** = 0. ❑ ***nage*** = 0. Aging is enabled unless ***nauto*** = 1. If ***nauto*** = 1, the value of ***nage*** is irrelevant – aging is disabled. This bit has no effect on address learning or time-stamp updating.
6	***dmainc***	DMA address autoincrement. When accessing the DIO interface with a DMA controller (**$\overline{\text{SDMA}}$** pin low), this bit determines whether the address held in ***dmaaddress*** should be incremented between successive accesses. ❑ ***dmainc*** = 1. The address increments between accesses. ❑ ***dmainc*** = 0. The address does not increment between accesses.
5	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
4	***unkvlan***	Unknown VLAN forwarding. This bit enables forwarding of multicast frames that have an unknown IEEE Std 802.1q VLAN Identifier (i.e., one that matches none of the VLAN IDs registered in ***VLAN0QID***–***VLAN63QID***). ❑ ***unkvlan*** = 1. All multicast frames belonging to unknown VLANs are forwarded to the port indicated in ***UnkVLANPort***. ❑ ***unkvlan*** = 0. All multicast frames belonging to unknown VLANs are discarded.

Table 4–14. SysControl (System Control Register) (Continued)

Bit	Name	Description
3	***mirr***	Port mirroring. This bit enables port mirroring. ❑ ***mirr*** = 1. All frames received on or sent to the port specified in ***MirrorPort*** are copied to the port specified in ***UplinkPort.*** ❑ ***mirr*** = 0. No mirroring is performed.
2	***nauto***	Not automatically add address mode. This bit selects the manner in which addresses are added to the address records. ❑ ***nauto*** = 1. Addresses can be added only to records using DIO adds. The aging-state machine is disabled; it is the external CPU's responsibility to manage the address records. ❑ ***nauto*** = 0. New addresses are learned from the wire and added to the address tables automatically. (They also can be added using DIO adds). However, if ***nage*** = 1 and the address tables become full, further addresses are not added unless space is created first using DIO deletes.
1	***ncrc***	No CRC check. This bit determines whether addresses are learned if the frame containing them contains an invalid CRC. ❑ ***ncrc*** = 1. Unknown addresses are added to the address-lookup table without regard to the validity of the CRC. ❑ ***ncrc*** = 0. Unknown addresses are added to the address-lookup table only if the frame containing the address has a valid CRC.
0	***flow***	Flow control enable. This bit enables flow control. ❑ ***flow*** = 1. The device implements flow control. Ports that are configured for IEEE Std 802.3x will use pause frames. Ports that are not configured for IEEE Std 802.3x use collisions. ❑ ***flow*** = 0. No flow control is implemented.

When the ***nauto*** bit is set to 1, the TNETX4020 does not learn addresses of the wire. Instead, all addresses must be added via the DIO interface. The ***nage*** bit is a don't care when the ***nauto*** bit is set to 1.

Aging is disabled by setting the ***nage*** bit to 1 while ***nauto*** is set to 0. The external CPU can delete addresses via the DIO interface.

4.3.2 Aging Algorithms

The TNETX4020 supports two automatic aging algorithms, in addition to letting a host CPU provide these functions: time threshold and table full. Both algorithms determine the age of an address record in the same manner. In both algorithms, the oldest address record is deleted. The difference is that when a delete occurs using the time-threshold algorithm, the table might not be full. This is because the address (node) has not transmitted a frame (address record updated in the address-lookup table) before the time expires.

AgingThreshold (Age Out Time Select Register) @ 0x0044–0x0045

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address Offset 0x0
Field Name	***threshold[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–15. AgingThreshold (Age Out Time Select Register)

Bit	Name	Description
15:0	***threshold***	Aging threshold. There are two aging modes that are selected by the value of this register. ❑ When ***threshold*** is 0x0000 or 0xFFFF, table-full aging is performed. The oldest address is aged out only when the lookup table becomes full. The ***AgingCounter*** increments by one only after a new address is added to the address records. ❑ When ***threshold*** is not 0x0000 or 0xFFFF, threshold aging is performed. The value in this field is the time threshold in multiples of 8 seconds. (i.e., 0x0001 = 8 seconds). All address records that are older than this time are aged out. The ***AgingCounter*** increments by one every 8 seconds. Aging does not delete unicast addresses that have been locked or secured, or multicast addresses. ❑ Aging is disabled when ***nauto*** bit and/or the ***nage*** bit is set to 1 in ***SysControl***. It then becomes the external CPU's responsibility to delete unwanted address records. The ***AgingCounter*** continues to increment in the manner selected by this ***threshold*** value, i.e., only after address adds, or once every 8 seconds, allowing the address-deletion criteria to be determined by the user.

The switch is configured for the table-full aging algorithm when the external CPU writes 0xFFFF or 0x0000 into the ***AgingThreshold*** register. The oldest address record is deleted when the address table becomes full. The table-full algorithm is the default operation of the device.

If any other value is written into the ***AgingTreshold*** register, the switch utilizes the time threshold aging algorithm. The value contained in the ***AgingThreshold*** register is a multiple of 8 seconds. The ***AgingCounter***, which is used to compare the age of an address, is updated or incremented once every 8 seconds. When an address becomes older than the value in the ***AgingThreshold***, the address is deleted.

Note:

If the address table becomes full using the time-threshold algorithm and the switch needs to add (learn) a new address either from the DIO or wire, the oldest address record is deleted, even though the age is below the value in the ***AgingThreshold*** register.

4.3.3 VLAN Configuration

The TNETX4020 supports up to 64 VLANs. The VLAN ID can be any of the 4096 valid VLAN IDs. VLANs are configured using the ***VLANxPorts*** and the ***VLANxQID*** registers. The ***VLANxQID*** registers contain the VLAN ID, while the ***VLANxPorts*** registers contain the port members of that particular VLAN.

VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register) @ 0x0300–0x037F

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0300 + 0x2*n
Field Name	reserved				***vlanqid[11:0]***												
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Value†	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	–	–	–	–	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

† ***VLAN0QID*** is initialized to 000000000001; all other ***VLANxQID*** registers are initialized to 000000000000.

Table 4–16. VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register)

Bit	Name	Description
15:12	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
11:0	***vlanqid***	IEEE Std 802.1q VLAN identifier. Specifies the IEEE Std 802.1q VLAN identifier associated with each of the device's VLANs. The IEEE Std 802.1q ID associated with a received frame is converted to an internal VLAN index using these registers. Frames are associated with an IEEE Std 802.1q VLAN ID in one of two ways, depending on the value of the ***rxacc*** bit at the ingress ***PortxControl*** register: ❑ If the frame contains an IEEE Std 802.1q VLAN ID, that ID is used. ❑ If the frame contains no IEEE Std 802.1q VLAN ID or has an ID of 0x000, the default source-port VLAN contained in ***PortxQTag*** is used. Unicast frames that contain an IEEE Std 802.1q VLAN ID that is not recognized are discarded. Multicast frames that contain an IEEE Std 802.1q ID that is not recognized are forwarded according to ***UnkVLANPort*** if ***unkvlan*** in ***SysControl*** = 1; otherwise, the frames are discarded. The least significant byte of a ***VLANxQID*** register must be written first because it goes into a holding latch. Only when the most significant byte is written is the whole 12-bit ***vlanqid*** entered into the ***VLANxQID*** register.

VLAN0Ports–VLAN63Ports (VLAN Routing Register) @ 0x0100–0x01FF

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0100 + 0x4*n
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	DIO Address 0x0102 + 0x4*n
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–17. VLAN0Ports–VLAN63Ports (VLAN Routing Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	VLAN port vector. Frames with multicast destination addresses that are associated with VLAN index n are forwarded to the ports determined by logically ANDing the port vector associated with the address in the address-lookup table with this field in ***VLANnPorts***. If the address is unknown, ***VLANnPorts*** are ANDed with ***UnkUniPorts*** or ***UnkMultiPorts***, depending on whether the address is unicast or multicast, to determine the ports to which the frame is forwarded. Destination addresses are associated with a particular VLAN index in one of several ways: ❑ If the frame contains an IEEE Std 802.1q VLAN ID that matches the entry in ***VLANnQID***, the frame is associated with VLAN index *n.* ❑ If the frame contains no IEEE Std 802.1q VLAN ID or contains a VLAN ID of 0x000, the frame is associated with the VLAN in the source port's ***PortxQTag*** register.

The TNETX4020 default VLAN has a VLAN ID of 0x001, of which every port is a member. (***VLAN0Ports*** and ***VLAN0QID*** registers contain the default values.) Frames that contain no VLAN tag or a VLAN ID of 0x000 have a tag added or VLAN ID substituted with the value contained in the ***PortxQTag*** registers. The default value of the ***PortxQTag*** registers is 0x001, hence every port is configured to be a part of VLAN 0x001.

The external CPU can configure the switch to support a particular VLAN by specifying the port members in a ***VLANxPorts*** register and the VLAN ID in a ***VLANxQID*** register. Note that the ***VLANxQID*** registers and the ***VLANxPorts*** registers are on a one-to-one basis. That is, if the value of ***VLAN12QID*** register matches the VLAN ID in the received frame, the switch uses the ***portvector*** in ***VLAN12Ports*** register as the port members of that particular VLAN.

Note:

It is strongly recommended that when a VLAN is configured, the ***VLANxPorts*** register is configured before the ***VLANxQID*** register.

4.3.4 Unknown Addresses

Frames that are received with an unknown destination address, unknown source address, as well as multicast frames with an unknown VLAN ID, can be broadcast on the network using the ***UnkUniPorts***, ***UnkMultiPorts***, ***UnkSrcPorts***, or ***UnkVLANPort*** registers.

If a unicast frame is received with an unknown destination address, the frame is forwarded according to the ***UnkUniPorts*** register.

Note:

A frame is never forwarded to the same port on which it was received using the registers in Section 4.3.4, *Unknown Addresses*.

UnkUniPorts (Unknown Unicast Address Routing Register) @ 0x0060–0x0063

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table 4–18. UnkUniPorts (Unknown Unicast Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown unicast destination address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown unicast destination port vector. When a frame has a unicast destination address that is not contained within the address-lookup table, this field determines to which ports the frame should be forwarded.

A multicast frame with an unknown destination address is forwarded according to the ***UnkMultiPorts*** register.

UnkMultiPorts (Unknown Multicast Address Routing Register) @ 0x0064–0x0067

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table 4–19. UnkMultiPorts (Unknown Multicast Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown multicast destination address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown multicast destination port vector. When a frame has a multicast destination address that is not contained within the address-lookup table, this field determines to which ports the frame should be forwarded.

Frames with an unknown source address are forwarded according to the ***UnkSrcPorts*** register.

UnkSrcPorts (Unknown Source Address Routing Register) @ 0x0068–0x006B

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table 4–20. UnkSrcPorts (Unknown Source Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown source address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown source port vector. When a frame is received from an address that is not contained within the address-lookup table, this field determines to which ports the frame should be forwarded.

When the ***unkvlan*** bit is set to 1 in the ***SysControl*** register, multicast frames received with an unknown VLAN ID are forwarded to the port specified in ***UnkVLANPort*** register. Note that unicast frames with an unknown VLAN ID always are discarded.

UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register) @ 0x006C–0x006F

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–21. UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Enable unknown VLAN interrupts port vector. Enables generation of unknown VLAN and unknown VLAN member interrupts for the ports whose corresponding bit is set, provided the appropriate bits in ***IntEnable*** are set also. This vector has no effect on how frames are forwarded or filtered.

4.3.5 Mirroring

As a debug tool, the TNETX4020 allows two types of mirroring: destination-address mirroring and port mirroring. Destination-address mirroring mirrors all frames whose destination address has its ***cuplnk*** bit set in the address records. The frames are mirrored to the port specified in the ***UpLinkPort*** register. Port mirroring sends all frames that are received or transmitted on the ***MirrorPort*** to the ***UplinkPort***.

Note:

A frame is never forwarded to the same port on which it was received using the registers in Section 4.3.5, *Mirroring*.

UplinkPort (Uplink Routing Register) @ 0x0040

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–22. UplinkPort (Uplink Routing Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Uplink extended port code. Identifies the port to which all mirrored frames are destined. Two types of mirroring are possible: ❑ Destination-address mirroring. Individual unicast addresses can be mirrored if the ***cuplnk*** bit for that address is set. In this case, all frames destined to this address are forwarded to the port specified in this field. ❑ Port mirroring. A port selected through ***MirrorPort*** can be selected for port mirroring. In this case, all frames received by or destined for this port are copied to the port specified in this field. This includes frames that normally would not be routed through the device, such as when both destination and source are assigned to the same port. Port mirroring is possible only if ***mirr*** in ***SysControl*** is set. If this field specifies an unimplemented port number, that port is not included in the port routing code. Selection of a reserved crossbar-switch port is interpreted as an ***xroutecode***, and the pretag is derived by indexing into the ***XMultiGroup*** registers. Both of these mechanisms can be active at the same time. While mirroring may pass some additional frames, this mechanism does not cause a second copy of a packet to appear on any port. Mirrored traffic is subject to the same rules regarding buffer usage with respect to flow control as normal packets. The use of an extended port code allows mirrored traffic to be sent through a crossbar, which can only be connected to port 1.

Port mirroring is achieved by using the ***MirrorPort*** register in conjunction with the ***UpLinkPort*** register. The ***MirrorPort*** register must be configured to contain the port to be mirrored (mirror port). The ***UplinkPort*** register must be configured as the port that receives all the mirrored frames. All frames sourced from or destined to the mirror port are forwarded to the port configured in the ***UpLinkPort*** register. The port-mirroring feature is enabled only when the ***mirr*** bit in the ***SysControl*** register is set to 1.

MirrorPort (Mirrored Port Select Register) @ 0x0041

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved						***portcode[1:0]***		
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	h	h	
Field Type	r	r	r	r	r	r	rwl	rwl	

Table 4–23. MirrorPort (Mirrored Port Select Register)

Bit	Name	Description
7:2	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
1:0	***portcode***	Mirror extended port code. Identifies the local port that is to be monitored when ***mirr*** in ***SysControl*** is set. All frames received by or destined for this port are copied to the port identified by the ***portcode*** field of ***UplinkPort*****,** unless ***UplinkPort*** contains the same port number as the source port. If the port code names an unimplemented port, this option has no effect on the port routing code.

4.3.6 Support for the Spanning-Tree Algorithm

The TNETX4020 supports the spanning-tree algorithm in that the ports can be configured for blocking, listening, learning, forwarding, or disabled mode. These port states, except disabled, are achieved by using ***NLearnPorts***, ***TxBlockPorts***, ***RxUniBlockPorts***, and ***RxMultiBlockPorts*** registers (see the following register descriptions).

NLearnPorts (Address Learning Disable Register) @ 0x0050–0x0053

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–24. NLearnPorts (Address Learning Disable Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Disable learning port vector. Disables learning for the ports whose corresponding bit is set. Frames received on a port where learning is disabled are forwarded, but new source addresses are not added to the address-lookup table.

TxBlockPorts (Block Transmit Register) @ 0x0054–0x0057

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved													***portvector [2:0]***			0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset
Field Name	reserved																0x2
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–25. TxBlockPorts (Block Transmit Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Block transmit port vector. Blocks forwarding of frames, depending on the port to which the frame is destined. Frames destined to ports whose corresponding bit is set are discarded.

RxUniBlockPorts (Block Unicast Receive Register) @ 0x0058–0x005B

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved													***portvector [2:0]***			0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset
Field Name	reserved																0x2
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–26. RxUniBlockPorts (Block Unicast Receive Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Block unicast receive port vector. Blocks forwarding of unicast frames, depending on the port from which the frame originated. Unicast frames received on ports whose corresponding bit is set are discarded. ***RxUniBlockPorts*** can be overridden if the destination address in the address-lookup table has the ***nblck*** bit set.

RxMultiBlockPorts (Block Multicast Receive Register) @ 0x005C–0x005F

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 4–27. RxMultiBlockPorts (Block Multicast Receive Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Block multicast receive port vector. Blocks forwarding of multicast frames, depending on the port from which the frame originated. Multicast frames received on ports whose corresponding bit is set are discarded. ***RxMultiBlockPorts*** can be overridden if the destination address in the address-lookup table has the ***nblck*** bit set.

The states for a specific port are configured as follows:

Blocking

1) Port bit in ***portvector*** set to 1 in ***NLearnPorts*** register
2) Port bit in ***portvector*** set to 1 in ***TxBlockPorts*** register
3) Port bit in ***portvector*** set to 1 in ***RxUniBlockPorts*** register
4) Port bit in ***portvector*** set to 1 in ***RxMultiBlockPorts*** register

Listening

1) Port bit in ***portvector*** set to 1 in ***NLearnPorts*** register
2) Port bit in ***portvector*** set to 1 in ***TxBlockPorts*** register
3) Port bit in ***portvector*** set to 1 in ***RxUniBlockPorts*** register
4) Port bit in ***portvector*** set to 1 in ***RxMultiBlockPorts*** register

Learning

1) Port bit in ***portvector*** set to 0 in ***NLearnPorts*** register
2) Port bit in ***portvector*** set to 1 in ***TxBlockPorts*** register
3) Port bit in ***portvector*** set to 1 in ***RxUniBlockPorts*** register
4) Port bit in ***portvector*** set to 1 in ***RxMultiBlockPorts*** register

Forwarding

1) Port bit in ***portvector*** set to 0 in ***NLearnPorts*** register
2) Port bit in ***portvector*** set to 0 in ***TxBlockPorts*** register
3) Port bit in ***portvector*** set to 0 in ***RxUniBlockPorts*** register
4) Port bit in ***portvector*** set to 0 in ***RxMultiBlockPorts*** register

Disabled

The disable bit in the port's control register (***PortxControl***) must be set to 1.

Note:

The external CPU must add the bridge protocol data unit's (BPDU) MAC address, via the DIO, to the IALE's address records with the ***nblk*** bit set to 1 for BPDUs to be received on ports in blocking, listening, or learning states. Refer to Chapter 5 for more detail on adding addresses via the DIO interface.

4.3.7 Extended Port Awareness

The extended-port-awareness feature allows the TNETX4020 to be connected to a "satellite" switch via the uplink port and be aware of the ports on this satellite switch and forward frames to these ports. This feature is operational only when the ***pretag*** bit in ***Port1Control*** register is set to 1. The extended-port-awareness feature makes use of the 47 ***XMultiGroup*** registers (***XMultiGroup17***–***XMultiGroup63*** registers) and the ***xroutecode***. A definition of the extended-port specifying codes is in Appendix A.

XMultigroup17–XMultigroup63 (Extended Multicast Group Register) @ 0x0544–0x05FF

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address Offset 0x0500 + 4*n
Field Name	***xportvector[15:0]***																
Reset Value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	DIO Address Offset 0x0502 + 4*n
Field Name	reserved															***xportvector***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw	

Table 4–28. XMultigroup17–XMultigroup63 (Extended Multicast Group Register)

Bit	Name	Description
31:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16:0	***xportvector***	Extended multicast group port vector. This field determines the extended port vector to be inserted into the preframe tag added to frames routed to port 1 when ***pretag*** = 1 in ***Port1Control***. ***XMultiGroupn*** is associated with ***xroutecode*** = *n*, where *n* is in the range 17–63. If ***xroutecode*** is less than 17, it is interpreted as a unicast extended port vector, and only bit *n* of the vector is set.

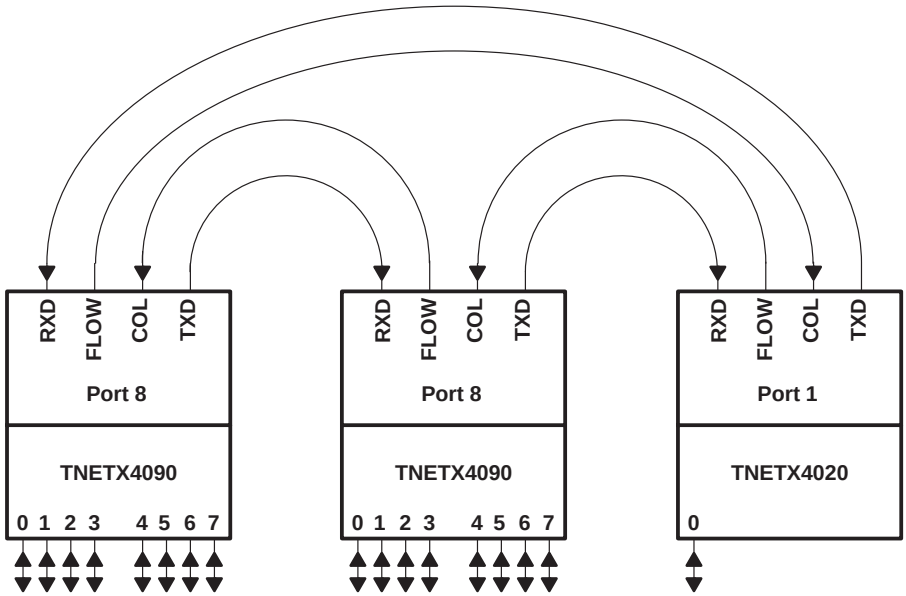
The ***xroutecode*** is found only in ***UnkUniPorts***, ***UnkMultiPorts***, ***UnkSrcPorts***, ***FindPort*** (multicast form), and ***AddPort*** (multicast form) registers. The ***xroutecode*** indicates a single port on the satellite switch when the value is between 0 and 16. When the value of the xroutecode is between 17 and 63, it serves as an index into the ***XMultiGroup*** registers. The content of the ***XMultiGroup*** registers is the ***xportvector*** (or simply the port vector for the satellite switch) that will be part of the pretag (see the TNETX4020 data sheet for detailed information on pretag format).

The ***UnkUniPorts***, ***UnkMultiPorts***, ***UnkSrcPorts*** registers may use the ***xroutecode*** to flood the entire switch, including the satellite switch, when an unknown frame is received. Moreover, a multicast address can be added by the external CPU to broadcast frame with this address to ports on the satellite switch.

4.4 Cascading-Ring Topology

The cascading-ring-topology feature of the TNETX4020 provides a way to construct a high 10-/100-Mbit/s port-density system using a combination of TNETX4090s and/or TNETX4020s (see Figure 4–2). This feature allows vendors to build a high-port-density, low-cost system.

Figure 4–2. Cascading-Ring Topology



To configure a port of the TNETX4020 to operate in ring mode, the corresponding bit in the ***portvector*** in the ***RingPorts*** register must be set to 1. This allows port 0 and/or port 1 to be connected to other ThunderSWITCH devices with gigabit ring ports. Furthermore, the ***ringid*** of each switching device in the ring must be configured to a unique value to prevent frames from circumnavigating the ring topology more than one complete loop.

RingPorts (Ring Topology Enable Register) @ 0x008C

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***ringid[3:0]***				***holbrm***	reserved	***portvector[1:0]***		
Reset Value	0000	0000	0000	0000	0	0	00	00	
Reset Type	h	h	h	h	h	–	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	r	rwl	rwl	

Table 4–29. RingPorts (Ring Topology Enable Register)

Bit	Name	Description
7:4	***ringid***	Ring ID. Specifies the ID of this device in ring-topology mode. Each device on the ring should be configured with a different nonzero value of this field. If more than one port is configured for ring mode, all will use the same ***ringid***.
3	***holbrm***	HOLB ring mode. The ***holbrm*** bit works only in conjunction with the ***flow*** and ***holb*** bits. ❑ When ***holbrm***, ***flow***, and ***holb*** bits are set, ports enabled for ring mode assert the **Mxx_FLOW** pin when half of the ***FlowThreshold*** value is exceeded. Setting ***holbrm*** has no effect on the ***FLOW*** led threshold. ❑ When ***holbrm*** is reset, and ***flow*** and ***holb*** bits are set, ports enabled for ring mode assert the **Mxx_FLOW** pin when the HOLB threshold or one-half of the ***FlowThreshold*** value is exceeded, whichever occurs first. ❑ If either ***flow*** or ***holb*** are reset, setting ***holbrm*** has no effect.
2	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
1:0	***portvector***	Ring topology port vector. Determines which ports are configured for operation in a ring topology. When set, ring topology is enabled. Frames received on a nonring port, then transmitted from a ring port use ***ringid*** to form the out-of-band pretag transmitted with the frame. In this way, frames are tagged with the ID of the device that introduced them to the ring. Frames received on a ring port that have a pretag that matches ***ringid*** are filtered. This ensures that frames circumnavigate the ring only once. In ring-topology mode, frames received on the port can be transmitted back out the same port. When both ports are configured for ring mode, where the switch is acting as a bridge between two rings, frames have the pretag stripped on reception and replaced with ***ringid*** on transmission. If any ports are configured for ring topology, the ***pretag*** bit in the ***Port1Control*** must be clear.

The normal behavior of the switch is modified in two ways when a port is configured for ring mode:

❑ Frames received can be transmitted back out on the same port. This allows frames to be passed along the ring from device to device.

- Frames transmitted from a ring port contain an out-of-band pretag consisting of the ***ringid*** configured in the ***RingPorts*** register. This pretag is used by the receiving ring port to decide if the frame should be forwarded or discarded. The out-of-band pretag prevents frames from circumnavigating the ring topology more than once.

For the ring topology to operate correctly, the IALEs of all the devices in the ring must learn the source address of frames destined to other switches. If a frame is received on switch A and is destined to a nonring port on switch C, the IALE must know to route the frame to the ring port and not broadcast the frame. Broadcasting the frame works, but the performance of the system may suffer dramatically. To keep all the IALEs updated correctly, the ***UnkSrcPorts*** register must be configured to forward frames with an unknown source address to the ring port.

UnkSrcPorts (Unknown Source Address Routing Register) @ 0x0068–0x006B

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table 4–30. UnkSrcPorts (Unknown Source Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown source address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown source port vector. When a frame is received from an address that is not contained within the address-lookup table, this field determines to which local ports the frame should be forwarded.

Setting bit 0 in the ***UnkSrcPorts*** register makes the switch route all frames with an unknown source address to port 0. In doing so, the frame travels around the ring and all the device's IALEs are updated. When the frame has made its way around the ring, the out-of-band pretag indicates to the switching device that the frame has circumnavigated the switch and the frame is discarded.

Because the ring ports are like backplanes, transporting the frames to the proper switching devices, those ports must be configured not to add or to strip the VLAN tags in the frames. This is accomplished by correctly configuring the ***txacc*** and ***rxacc*** bits of the ***PortxControl*** registers of the device. Setting the ***rxacc*** and ***txacc*** bits to 0 ensures that the VLAN tag is retained on transmit (except when the VLAN tag equals the value contained in the ***PortxQtag*** register) and no extra VLAN tag is added when a frame is received. Configuring the ***PortxQtag*** register to all 0s is strongly recommended to prevent the frame's VLAN tag from accidentally being stripped on transmit.

Ring-mode ports are assumed to carry traffic from many stations, some of which only passes through. If the ***holb*** and ***flow*** bits in the ***SysControl*** register are both active, the ***holb*** mechanism could throttle ring traffic earlier than if only ***flow*** is set, if traffic is only passing through the switch. With ***holb***, less total memory can be used if only one port is in use. The effects of ***holb*** in ***SysControl*** can be removed from ports in ring mode by setting the ***holbrm*** in the ***RingPorts*** register. If ***holb*** and ***flow*** are both on, and ***holbrm*** also is set, the ring port's **Mxx_FLOW** pin is asserted only when the ***FlowThreshold*** limit of buffers left is reached. This could cause other traffic in the switch to become buffer starved, but allocates the maximum amount of resources to the ringed port.

The ring port must be configured for full-duplex operation. Full duplex is essential because these ports must be able to transmit to one switching device while receiving from another. Ring-mode ports must not be configured to respond to or transmit IEEE Std 802.3x pause frames. Instead, they use the **Mxx_FLOW** and **Mxx_COLL** pins to provide flow control between the devices (see Figure 4–1 for a system overview). For the **Mxx_FLOW** pin to provide flow control when the buffer threshold has been broken, the ***flow*** bit in the ***SysControl*** register must be set to 1. The switching device whose **Mxx_FLOW** pin has been asserted holds off the transmitting switching device until the congested condition has been eliminated.

4.4.1 Summary

This is a brief summary of the register configuration that allows the TNETX4020 to operate correctly in a ring topology.

1) Set corresponding ***portvector*** bit in the ***RingPorts*** register to 1.

2) Configure the ***ringid*** field in the ***RingPorts*** register to a unique value for every switching device in the ring topology.

3) Set the ***flow*** bit in the ***SysControl*** register to 1.

4) Set the ring port's bit of the ***UnkSrcPorts*** register.

5) Set the ring port's ***PortxQtag*** register to all 0s.

6) Configure the bits in ***PortxControl*** register accordingly:

   - ***rxacc*** = 0
   - ***txacc*** = 0
   - ***reqhd*** = 0
   - ***reqnp*** = 1

Also, attention needs to be paid to the ***holbrm*** bit with respect to product performance requirements.

4.5 Statistics Configuration

The ***StatControl*** register allows the external CPU to configure certain aspects of the statistics RAM, as well as clearing statistics.

StatControl (Statistics Control Register) @ 0x00F8–0x00F9

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved					***stmap***	***long***	***bigend***	***clra***	***clrp***	reserved				***portcode [1:0]***		
Reset Value	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	
Reset Type	–	–	–	–	–	h	h	h	h	h	–	–	–	–	h	h	
Field Type	r	r	r	r	r	rwl	rwl	rwl	rwl	rwl	r	r	r	r	rwl	rwl	

Table 4–31. StatControl (Statistics Control Register)

Bit	Name	Description
15:11	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
10	***stmap***	Statistic mapping. Selects which statistic is recorded in multiple-function statistic counters (currently only Portx Tx Data Errors). ❑ ***stmap*** = 1. The number of frames that have been sent to the transmit queue are recorded. ❑ ***stmap*** = 0. The number of data errors at transmit are recorded.
9	***long***	Long frame handling. Determines how statistics are kept on frames greater than 1518 bytes in length. If ***long*** = 0 or ***maxlen*** = 1 in the port's ***PortxControl*** register, the following statistics do not record frames if they are over 1518 bytes long: ❑ Portx Good Rx frames ❑ Portx Broadcast Rx frames ❑ Portx Multicast Rx frames ❑ Portx Pause Rx frames ❑ Portx Rx CRC errors ❑ Portx Broadcast Rx frames ❑ Portx 1024–1518 octet frames (Rx frames only) ❑ Portx Rx octets The following statistics record frames that are more than 1518 bytes long: ❑ Portx Rx jabbers ❑ Portx oversized Rx frames If ***long*** = 1 and ***maxlen*** = 0 in the port's ***PortxControl*** register, the boundary at which the above statistics stop or start to record frames moves from 1518 bytes to 1531 bytes. The ***long*** bit has no effect upon any Tx statistics. This bit has no effect upon the longest frame size that will be forwarded. The ***long*** bit solely affects the statistics.

Table 4–31. StatControl (Statistics Control Register) (Continued)

Bit	Name	Description
8	***bigend***	Big endian. Determines the order in which bytes within each 32-bit counter are read: ❑ ***bigend*** = 1. A byte address of 0 references the numerically most significant byte of a counter, and a byte address of 3 references the numerically least significant byte of a counter. ❑ ***bigend*** = 0. A byte address of 0 references the numerically least significant byte of a counter, and a byte address of 3 references the numerically most significant byte of a counter. This bit affects accesses to DIO addresses only in the range 0x8000 and above.
7	***clra***	Clear address lookup statistics. Writing a 1 to this bit causes all address-lookup statistics (DIO addresses in the range 0xA000–0xA00B) to be cleared. (Writing a 0 has no effect.) This bit then becomes write protected and autoclears when the statistics clear is complete. As the statistics are maintained in RAM and updated in a cyclic manner, clearing is not instantaneous. After a hardware reset, this bit is set to 1 so that all address-lookup statistics are cleared upon power up. This does not occur following a stop.
6	***clrp***	Clear port statistics. Writing a 1 to this bit causes all statistics counters associated with the port indicated by ***portcode*** to be cleared. (Writing a 0 has no effect.) This bit then becomes write protected and autoclears when the statistics clear is complete. As the statistics are maintained in RAM and updated in a cyclic manner, clearing is not instantaneous. After a hardware reset, this bit is set to 1 (and portcode will be all 1s) so that all port statistics are cleared upon power up. This does not occur following a stop.
5:2	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
1:0	***portcode***	Clear port code. When ***clrp*** is set to 1, this field indicates which port's statistics counters are to be cleared. If ***portcode*** is all 1s, all ports are cleared. Leave these bits unchanged until ***clrp*** becomes 0. If an unimplemented port is specified in this field and ***clrp*** is set to 1, ***clrp*** eventually clears without clearing any port statistics.

The external CPU can configure the statistics for big or little endian by appropriately setting the ***bigend*** bit. The default mode of the statistics RAM is in little endian.

Note:

Setting the ***bigend*** bit to 1 affects only the statistics RAM (DIO address 0x8000 and above).

Setting the ***long*** and ***stmap*** bits affects how certain statistics are kept. See *register description* for detail.

The ***clra***, ***clrp***, and ***portcode*** bit fields in the ***StatControl*** register allow the external CPU to selectively reset port and address-lookup statistics for the port specified in ***portcode***. A ***portcode*** of 0x3 clears the statistics for all ports.

4.6 LED Configuration

The ***LEDControl*** register allows the external CPU to configure the LED interface. The external CPU can configure the LED interface to include multicast traffic as port activity, control the fault LED, configure the order of the LEDs shifted out from the LED interface, and turn on or off the software-controlled LEDs.

LEDControl (LED Control Register) @ 0x00F4–0x00F5

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	reserved	***txais***	***fltled***	***slast***	***swled[11:0]***												Byte Address Offset 0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table 4–32. LEDControl (LED Control Register)

Bit	Name	Description
15	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
14	***txais***	Transmit activity indication select. Allows the transmit content of the port-activity LEDs to be controlled: ❑ ***txais*** = 1. The port-activity indication includes transmit multicast traffic. ❑ ***txais*** = 0. The port-activity indication does not include transmit multicast traffic. Receive traffic indication is not dependent upon this bit value or the type of frame being received.
13	***fltled***	Fault LED. Controls the Fault LED as follows: ❑ ***fltled*** = 1. The Fault LED is turned on. ❑ ***fltled*** = 0. The Fault LED is on only if: ■ An invalid CRC was detected during the EEPROM load. This condition is cleared by hardware reset or by setting the ***load*** bit to 1 in ***SysControl***, or ■ A parity error has occurred in the external memory system. This condition is cleared by hardware reset or by setting the ***start*** bit to 1 in ***SysControl***.
12	***slast***	Software LEDs last. Determines the position of the software-controlled LEDs (as controlled by ***swled*** and ***pmled***) in the shift-out sequence. ❑ ***slast*** = 1. The software LEDs are shifted out after the port LEDs. ❑ ***slast*** = 0. The software LEDs are shifted out before the port LEDs.

Table 4–32. LEDControl (LED Control Register) (Continued)

Bit	Name	Description
11:0	***swled***	Software controlled LEDs. These 12 LEDs are available for general software control. ❑ ***swled*** = 1. The corresponding LED is turned on. ❑ ***swled*** = 0. The corresponding LED is turned off. There is no additional hardware control of these LEDs. The writing of these bits is not synchronized to the display of the LEDs. It is possible for the LED interface to read these 12 bits between a write to the least significant byte and a write to the most significant byte of this register (or vice versa). If the quantities being displayed on the LEDs straddle the byte boundary, the bytes may appear on the LEDs 1/16 of a second apart.

The default operation of the switch is to not indicate multicast as port activity on the port status LED. By setting the ***txais*** bit to 1 in the ***LEDControl*** register, multicast traffic is included as activity.

The external CPU can control the fault LED by setting or clearing the ***fltled*** bit.

The external CPU can control 12 LEDs that are shifted out on the LED interface. A 1 in a bit position turns the corresponding LED on and, likewise, a 0 in a bit position turns the corresponding LED off. The value of the ***slast*** bit in the ***LEDControl*** register determines the order of the LEDs shifted out on the LED interface (see the following table).

Order†			
***slast* = 0**	***slast* = 1**	**Name**	**Function**
1st–6th	1st–6th	**0**	Zero. Dummy data for first 6 of 24 **LED_CLK** cycles.
7th–18th	11th–22nd	**SW0–SW11**	Software LEDs 0–11. These LEDs are intended to allow additional software-controlled status to be displayed. These 12 LEDs reflect the values of bits 0 through 11 of the ***swled*** field in ***LEDControl*** at the moment that the LED interface samples them. If this is between writes to the most significant and least significant bytes of ***LEDControl***, these values appear on the LEDs separated by 1/16 of a second.
19th–20th	7th–8th	**P00–P01**	Port status LEDs 00–01. These two LEDs indicate the status of ports 00 and 01, in this order (port 00 is output first). ***Port 02 (the management port) does not have an LED.*** The transmit multicast content of these bits can be controlled by the ***txais*** bit in ***LEDControl***. IEEE Std 802.3x pause frames never appear on the LEDs as port activity. The port's LED toggles each 1/16 of a second if there is any frame traffic (other then pause frames) on the port during the previous 1/16 of a second.
21st–22nd	9th–10th	**C00–C01**	Ports 00 and 01 collision LEDs. These two LEDs indicate the status of ports 00 and 01, in this order (port 00 is output first). ***Port 02 (the management port) does not have an LED.*** This LED is extinguished if the port is not in PMA mode. It indicates that the collisions toggle each 1/16 of a second if there a collision on the port during the previous 1/16 of a second.
23rd	23rd	**FLOW**	Flow control. This LED is on when the internal flow control is enabled and active. ''Active'' means that flow control was asserted during the previous 1/16 of a second. Flow control is activated by the HOLB logic when the HOLB threshold on a port is exceeded and both ***flow*** and ***holb*** bits are set. If only the ***holb*** bit is set, the FLOW led is not illuminated if the HOLB threshold on a port is exceeded.
24th	24th	**FAULT**	Fault. This LED indicates: ❑ The EEPROM CRC was invalid, or ❑ An external DRAM parity error has occurred, or ❑ The ***fltled*** in ***LEDControl*** has been set. The CRC and parity error indications are cleared by hardware reset (pin or DIO), or by setting the ***reboot*** bit in the ***SysControl*** register. The CRC error indication also is cleared by setting ***load*** to 1. The parity error indication also is cleared by setting ***start*** to 1.

† Sequence in which the bits are shifted out on the LED interface

Chapter 5

System Operations

During operation of the device, the external CPU can perform several system operations, such as accessing and configuring the PHYs via the MDIO interface, accessing the EEPROM, transmitting and receiving frames via the network management (NM) port, and managing the internal address-lookup engine (IALE). All these system operations are performed using the TNETX4020 internal registers (see Tables 5–1 through 5–23).

Topic **Page**

5.1 Accessing PHYs 5-2

5.2 Accessing EEPROM 5-13

5.3 Network Management Port 5-29

5.4 Internal Address-Lookup Engine (IALE) Management 5-50

5.1 Accessing PHYs

The external CPU can access the PHYs via the MDIO interface, which is controlled through the ***SIO*** register. The external CPU can read and write to the PHYs' internal registers, as well as perform a hardware reset on each PHY connected to the **MRESET** pin. The PHY MDIO I/F can be put into the high-impedance state to accommodate another master. This register also contains the control bits to read and program the EEPROMS. The EEPROM I/F cannot be put into the high-impedance state.

SIO (Serial Interface I/O Register) @ 0x00A1

Note:

This register is not loaded during an EEPROM load.

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***nmrst***	***mclk***	***mtxen***	***mdata***	***mdioen***	***eclk***	***etxen***	***edata***	
Reset Value	1	0	0	**MDIO**	0	0	0	**EDIO**	
Reset Type	h	h	h	–	h	h	h	–	
Field Type	rw	rw	rw	rw†	rw	rw	rw	rw	

† ***mdata*** is read only when ***mtxen*** = 0.

Table 5–1. SIO (Serial Interface I/O Register)

Bit	Name	Description
7	***nmrst***	MII NOT reset. The state of this bit directly controls the state of the **MRESET** pin (MII reset). ❑ ***nmrst*** = 1. The external pin **MRESET** is set high. ❑ ***nmrst*** = 0. **MRESET** is set low. This bit is not self-clearing and must be deasserted manually. It can be set low, then immediately set high. Because every PHY attached to the MII may not have a reset pin, one might need to do ***nmrst*** and also individually reset each PHY. The ***mdioen*** bit must be set high to drive **MRESET**.
6	***mclk***	MII SIO clock. This bit controls the state of the **MDCLK** pin. ❑ ***mclk*** = 1. **MDCLK** is set high. ❑ ***mclk*** = 0. **MDCLK** is set low. The ***mdioen*** bit must be set high to drive **MDCLK**.
5	***mtxen***	MII SIO transmit enable. This bit is used in conjunction with the **mdata** bit to read/write information from/to the **MDIO** pin. ❑ ***mtxen*** = 1. **MDIO** is driven with the value in the ***mdata*** bit. ❑ ***mtxen*** = 0. ***mdata*** is loaded with the value on the **MDIO** pin. The ***mdioen*** bit must be set high to drive **MDIO**.

Table 5–1. SIO (Serial Interface I/O Register) (Continued)

Bit	Name	Description
4	***mdata***	MII SIO data. This bit is used in conjunction with **mtxen** to read/write information from/to the **MDIO** pin. ❑ ***mtxen*** = 1. **MDIO** is driven with the value in this bit. ❑ ***mtxen*** = 0. This bit is loaded with the value on the **MDIO** pin. The ***mdioen*** bit must be set high to drive **MDIO**.
3	***mdioen***	MII SIO data pin enable. This bit enables the high-impedance control of the **MDIO**, **MDCLK,** and **MRESET** pins. ❑ ***mdioen*** = 1. The MII SIO interface is enabled. ❑ ***mdioen*** = 0. The MII SIO interface is in a high-impedance state.
2	***eclk***	EEPROM SIO clock. This bit controls the state of the **ECLK** pin. ❑ ***eclk*** = 1. **ECLK** is set high. ❑ ***eclk*** = 0. **ECLK** is set low. Also, this bit is used to indicate if no EEPROM is found. If no EEPROM is detected when an EEPROM load is attempted, this bit is held at 0 and cannot be set to a 1. When this condition occurs, setting ***stop*** in ***SysControl*** unlocks this bit, allowing software to interrogate the EEPROM. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **ECLK** pin.
1	***etxen***	EEPROM SIO transmit enable. This bit controls the direction of the **EDIO** pin. ❑ ***etxen*** = 1. **EDIO** is driven with the value in the ***edata*** bit. ❑ ***etxen*** = 0. ***edata*** is loaded with the value on **EDIO**. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **EDIO** pin.
0	***edata***	EEPROM SIO data. This bit is used to read or write the state of the **EDIO** pin. ❑ ***etxen*** = 1. **EDIO** is driven with the value in this bit. ❑ ***etxen*** = 0. This bit is loaded with the value on **EDIO**. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **EDIO** pin.

5.1.1 PHY Reset

If the PHYs used in a TNETX4020-based system have their reset pins connected to the TNETX4020's **MRESET** pin, the external CPU can perform a hardware reset on the PHYs. A reset is performed on the PHYs when the **MRESET** pin is asserted low. This is accomplished by the external CPU setting the ***nmrst*** bit to 0. The ***mdioen*** bit must be set to 1 when ***nmrst*** is set to 0 (this can be done in the same DIO access). The PHYs are held in reset until the external CPU sets the ***nmrst*** bit to 1. The reset can be performed using two consecutive accesses of the SIO register. The time it takes the external CPU to perform consecutive access of the ***SIO*** register is sufficient time for a hardware reset to have been performed on the PHYs.

For the external CPU to control the MDIO interface and generate a PHY hardware reset, the ***mdioen*** bit in the ***SIO*** register must be set to 1.

5.1.2 Accessing PHY's Internal Register

Fields in the MII frame format (see Tables 5–2 and 5–3) allow the external CPU to determine which register of a particular PHY to access. Each character in the frame format represents one bit.

Table 5–2. MII Read Frame Format

Start Delimiter	Operation Code	PHY Address	Register Format	Turnaround	Data
01	10	AAAAA	RRRRR	Z0	DDDD.DDDD.DDDD.DDDD

Table 5–3. MII Write Frame Format

Start Delimiter	Operation Code	PHY Address	Register Format	Turnaround	Data
01	01	AAAAA	RRRRR	10	DDDD.DDDD.DDDD.DDDD

Start Delimiter

The start of an MII frame is indicated by a 01 bit pattern.

Operation Code

Operation code of 10: read
Operation code of 01: write

PHY Address

The 5-bit PHY address allows for 32 unique PHY addresses. The first PHY address bit transferred is the most significant bit (MSB) of the address. The upper three bits determine the PHY device, while the lower two bits indicate the channel number within the device.

Register Address

The register address is 5 bits, allowing 32 individual registers to be accessed within each PHY. The first register address bit transferred is the MSB of the address. Refer to the PHY data sheet for the address map of the individual register addresses.

Turnaround

For an MII write frame, the external CPU must write a 1 followed by a 0. For an MII received frame, the first bit is idle (time during which no device is actively driving the MDIO interface) followed by a 0. The idle bit is added in the turnaround field to avoid bus contention during a read.

Data

The data field is 16 bits. The first bit transferred (transmitted or received) is the MSB of the data payload.

The external CPU can transfer bits to and from the MDIO interface using the ***mclk***, ***mtxen***, and ***mdata*** bits of the ***SIO*** register. The MDIO interface is a serial interface totally controlled by the external CPU. The external CPU determines whether the access is a write access (transmitted) or a received access (received) via ***mtxen***, the bit read or written (***mdata***), and the clock of the interface (***mclk***).

Note:

The ***mdioen*** bit must be set to 1 for the MDIO interface to be enabled.

Data is transferred on the rising edge of the clock (***mclk***). When writing to a PHY (***mtxen*** = 1) the external CPU must load ***mdata*** with the bit to be written while ***mclk*** is set to 0. The bit contained in ***mdata*** is clocked out on the MDIO interface by setting ***mclk*** to 1. For a read access (***mtxen*** = 0), the ***mdata*** is loaded with the value from the MDIO interface after ***mclk*** is set to 1. The external CPU must clock in the data by setting the mclk to 1, then read the bit contained in ***mdata*** on the next DIO access.

5.1.3 Code Examples

The following three procedures of pseudocode describe the external DCPU performing a write access on the MDIO, a read access on the MDIO, and performing a hardware reset on the PHYs. Since there are more details involved in reading and writing to the MDIO registers through the serial register, more detailed code is provided.

This is the header information. See Appendix B for the routine prototypes.

```
//-----------------------------------------------------------------------------
// Thunderswitch interface functions
//-----------------------------------------------------------------------------
// File: mii.c
//
// This module contains defined the following functions associated with
// accessing Thunderswitch hardware:
//
// MiiRdWord   – Read one word from MII registers
// MiiWrWord   – write one word to MII registers
//
// This module uses the following hardware specific functions.
// These are dependent on the system specific processor to TSwitch
// interface and must be ported.
//
// InMii       – Read MII signal MDATA from the wire
// OutMii      – Write the MII signals MDATA, MDCLK, MTXEN to the wire
//
//-----------------------------------------------------------------------------

#include ”tsif.h”
#include ”tsifhw.h”
```

This is a callable routine to read a 16-bit register in an MII-compliant physical-interface device connected to the MDIO port of the switch. When performing a read, the external CPU must write the start delimiter, operation code, PHY address, and register format as specified in an MII frame. After the external CPU has written the register format, it must relinquish the MDIO interface to the PHY, i.e., the external CPU must start to do reads. For this reason, there is an idle bit in the turnaround to prevent the PHY and the external CPU from driving the MDIO simultaneously. Hence, the external CPU must read the turnaround, followed by the data.

```
//---------------------------------------------------------------------
//  MiiRdWord() – Read a word from Phy MII, place at given ptr, return status
//
//  Parameters:
//   ts_del     int  Fast start delimiter option flag
//   dev        int  device to read from
//   addr       int  register on dev to read from
//   pval       WORD* storage for data read
//
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------
int MiiRdWord(int fs_del, int dev, int addr, WORD *pval)
{
    WORD i,tmp, ret = 1;
    BYTE b, ack;

    InMii( Port_Control, &b);

    // if phy does not do fast start delimiter, synchronize MII
    // by giving it 32 clocks
    b &= ~MTXEN; OutMii(b);
    b &= ~MCLK;  OutMii(b);
    b |= MCLK;   OutMii(b);
    if( !fs_del )
    {
        for (i = 0;i < 32;i++)
        {
            b &= ~MCLK; OutMii(b);
            b |= MCLK;  OutMii(b);
        }
    }

    // Fast start delimiter
    //b &= ~MDATA; b |= MCLK; OutMii(b); //1 (we're high impedance still)
    //b |= MTXEN; OutMii(b);             //0 onto MDIO

    b &= ~(MCLK); OutMii(b);            // disable Phy interrupt
    b |= MTXEN; OutMii(b);              // Enable drive direction

    // Start Delimiter sequence 01

    b &= ~MDATA; OutMii(b);             // 0
    b &= ~MCLK; OutMii(b);              // 0 (–ve egde)
    b |=  MCLK; OutMii(b);              // 1 (+ve edge)

    b |=  MDATA; OutMii(b);             // 1
    b &= ~MCLK; OutMii(b);              // 0 (–ve edge)
    b |=  MCLK;  OutMii(b);             // 1 (+ve edge)
```

```
// Read Command: 10

b |=  MDATA; OutMii(b);              // 1
b &= ~MCLK; OutMii(b);               // 0 (–ve edge)
b |=  MCLK;  OutMii(b);              // 1 (+ve edge)

b &= ~MDATA; OutMii(b);              // 0
b &= ~MCLK; OutMii(b);               // 0 (–ve egde)
b |=  MCLK; OutMii(b);               // 1 (+ve edge)

// Send the 5 bit device address, MSB first
for (i = 0x10;i;i >>= 1)
{
    if( i & dev )
    {
       b |= MDATA; OutMii(b);
    }
    else
    {
       b &= ~MDATA; OutMii(b);
    }
    b &= ~MCLK; OutMii(b);
    b |= MCLK; OutMii(b);
}

// Send the 5 bit register address, MSB first
for (i = 0x10;i;i >>= 1)
{
    if( i & addr )
    {
       b |= MDATA; OutMii(b);
    }
    else
    {
       b &= ~MDATA; OutMii(b);
    }
    b &= ~MCLK; OutMii(b);
    b |= MCLK; OutMii(b);
}

//-----------------------------------------------------------------
// 802.3u specifies a 2 bit spacing between register and
// address is sent, referred to as ”Turn–around” cycles.
// For a read, bus is high impedance on 1st bit time, and on 2nd
// phy drives bus to 0
//-----------------------------------------------------------------

b &= ~MTXEN; OutMii(b);
//togLH
b &= ~MCLK; OutMii(b);
b |= MCLK; OutMii(b);     //high impedance
```

```
    // Get PHY Ack
    InMii(Port_Status, &ack);

    b &= ~MCLK; OutMii(b);
    b |= MCLK; OutMii(b);

    if (!(ack & MDATA))
    {
        for (tmp = 0,i = 0x8000;i;i >>= 1)
        {
            InMii(Port_Status, &ack);
            if (ack & MDATA) tmp |= i;
            b &= ~MCLK; OutMii(b);
            b |= MCLK; OutMii(b);
        }
    }
    else
    {
        for (i = 0;i < 16;i++)
        {
            //togLH
            b &= ~MCLK; OutMii(b);
            b |= MCLK; OutMii(b);     //high impedance
            tmp = 0xffff;
            ret = 0;
        }
    }
    // Reenable the PHY interrupt
    b &= ~MCLK; OutMii(b);
    b |= MCLK;  OutMii(b);

    *pval = tmp;
    return(ret);
}
```

This is a callable routine to write a 16-bit register in an MII-compliant physical-interface device connected to the MDIO port of the switch. In this case the external CPU, via the bits in the SIO register and the external pins they control, drives the MDIO interface for the whole time. There is no mode change when the timing gets down to the turnaround cycle.

```
//---------------------------------------------------------------------
//  MiiWrWord() – Write a word to Phy MII, return status
//
//  Parameters:
//   fs_del     int   fast start delimiter flag
//   dev        int   device to write to
//   addr       int   register on dev to write to
//   pval       WORD  storage for data write
//
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------
int MiiWrWord(int fs_del, int dev, int addr, WORD data )
{
    WORD i;
    BYTE b;

    InMii( Port_Control, &b);

    // if phy does not do fast start delimiter, synchronize MII
    // by giving it 32 clocks
    b &= ~MTXEN; OutMii(b);
    b &= ~MCLK;  OutMii(b);
    b |= MCLK;   OutMii(b);
    if( !fs_del )
    {
        for (i = 0;i < 32;i++)
        {
            b &= ~MCLK; OutMii(b);
            b |= MCLK;  OutMii(b);
        }
    }

    // Fast start delimiter
    // b &= ~MDATA; OutMii(b);
    // b |= MCLK; OutMii(b);              //1 (we're high impedance still)
    // b |= MTXEN; OutMii(b);             //0 onto MDIO

    b &= ~MCLK; OutMii(b);                // disable Phy interrupt
    b |= MTXEN; OutMii(b);                // Enable drive direction

    // Start Delimiter sequence 01

    b &= ~MDATA; OutMii(b);               // 0
    b &= ~MCLK; OutMii(b);                // 0 (–ve egde)
    b |=  MCLK; OutMii(b);                // 1 (+ve edge)

    b |=  MDATA; OutMii(b);               // 1
    b &= ~MCLK; OutMii(b);                // 0 (–ve egde)
    b |=  MCLK; OutMii(b);                // 1 (+ve edge)

    // Write Command: 01

    b &= ~MDATA; OutMii(b);               // 0
    b &= ~MCLK; OutMii(b);                // 0 (–ve edge)
    b |=  MCLK;  OutMii(b);               // 1 (+ve edge)
```

```
b |=  MDATA; OutMii(b);              // 1
b &= ~MCLK; OutMii(b);               // 0 (–ve egde)
b |=  MCLK; OutMii(b);               // 1 (+ve edge)

// Send the device number Internal=31(0x1f), External=0(0x00)
for (i = 0x10;i;i >>= 1)
{
    if( i & dev )
    {
       b |= MDATA; OutMii(b);
    }
    else
    {
       b &= ~MDATA; OutMii(b);
    }
    b &= ~MCLK; OutMii(b);
    b |= MCLK; OutMii(b);
}

// Send the 5 bit register address, MSB first
for (i = 0x10;i;i >>= 1)
{
    if( i & addr )
    {
       b |= MDATA; OutMii(b);
    }
    else
    {
       b &= ~MDATA; OutMii(b);
    }
    b &= ~MCLK; OutMii(b);
    b |= MCLK; OutMii(b);
}

// 802.3u says during a write we drive the 2 bit turnaround time
// with 10

b |= MDATA; OutMii(b);             // 1
b &= ~MCLK; OutMii(b);
b |= MCLK; OutMii(b);

b &= ~MDATA; OutMii(b);            // 0
b &= ~MCLK; OutMii(b);
b |= MCLK; OutMii(b);

// send register data, MSB first
for (i = 0x8000;i;i >>= 1)
{
    if( i & data )
    {
       b |= MDATA; OutMii(b);
    }
    else
    {
       b &= ~MDATA; OutMii(b);
    }
    b &= ~MCLK; OutMii(b);
    b |= MCLK; OutMii(b);
}
b &= ~MTXEN; OutMii(b);
```

```
    b &= ~MCLK; OutMii(b);              // Quiescent cycle
    b |= MCLK; //OutMii(b);             // Re–enable PHY interrupt
    return 1;
}
```

Conceptually, the following code is all that is necessary to execute a reset on MII-compliant PHYs that support a hardware reset.

```
//
// Performing a hardware reset on the PHYs via the MDIO interface.
//
{
*DIO_addr_lo = 0xA1;
*DIO_addr_hi = 0x00;  // DIO address: SIO register
*DIO_data = 0x88;     // MDIO is enabled and MRESET pin
// asserted (i.e., a hardware reset is
// performed on the PHYs).
*DIO_data = 0x88;     // MDIO is enabled and MRESET pin
// deasserted (i.e., the PHYs are
// taken out of reset).
}
```

5.2 Accessing EEPROM

A 24C02 or 24C08 EEPROM can be used to autoconfigure the TNETX4020 (see the EEPROM application report for details). The external CPU can download the content of the EEPROM or reprogram it. The interface between the EEPROM and the TNETX4020 is a serial interface that can be controlled via the SIO register in a manner similar to the way the external CPU controls the MDIO interface.

SIO (Serial Interface I/O Register) @ 0x00A1

Note:

This register is not loaded during an EEPROM load.

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***nmrst***	***mclk***	***mtxen***	***mdata***	***mdioen***	***eclk***	***etxen***	***edata***	
Reset Value	1	0	0	**MDIO**	0	0	0	**EDIO**	
Reset Type	h	h	h	–	h	h	h	–	
Field Type	rw	rw	rw	rw†	rw	rw	rw	rw	

† ***mdata*** is read only when ***mtxen*** = 0.

Table 5–4. SIO (Serial Interface I/O Register)

Bit	Name	Description
7	***nmrst***	MII NOT reset. The state of this bit directly controls the state of the **$\overline{\text{MRESET}}$** pin (MII reset). ❑ ***nmrst*** = 1. **$\overline{\text{MRESET}}$** is set high. ❑ ***nmrst*** = 0. **$\overline{\text{MRESET}}$** is set low. This bit is not self-clearing and must be deasserted manually. It can be set low, then immediately set high. Because every PHY attached to the MII may not have a reset pin, one must do ***nmrst*** and also individually reset each PHY. The ***mdioen*** bit must be set to drive **$\overline{\text{MRESET}}$**.
6	***mclk***	MII SIO clock. This bit controls the state of the **MDCLK** pin. ❑ ***mclk*** = 1. **MDCLK** is set high. ❑ ***mclk*** = 0. **MDCLK** is set low. The ***mdioen*** bit must be set high to drive **MDCLK**.
5	***mtxen***	MII SIO transmit enable. This bit is used in conjunction with the **mdata** bit to read/write information from/to the **MDIO** pin. ❑ ***mtxen*** = 1. **MDIO** is driven with the value in the ***mdata*** bit. ❑ ***mtxen*** = 0. ***mdata*** is loaded with the value on the **MDIO** pin. The ***mdioen*** bit must be set high to drive **MDIO**.

Table 5–4. SIO (Serial Interface I/O Register) (Continued)

Bit	Name	Description
4	***mdata***	MII SIO data. This bit is used in conjunction with **mtxen** to read/write information from/to the **MDIO** pin. ❑ ***mtxen*** = 1. **MDIO** is driven with the value in this bit. ❑ ***mtxen*** = 0. This bit is loaded with the value on the **MDIO** pin. The ***mdioen*** bit must be set high to drive **MDIO**.
3	***mdioen***	MII SIO data pin enable. This bit enables the high-impedance control of the **MDIO**, **MDCLK** and **MRESET** pins. ❑ ***mdioen*** = 1. The MII SIO interface is enabled. ❑ ***mdioen*** = 0. The MII SIO interface is in a high-impedance state.
2	***eclk***	EEPROM SIO clock. This bit controls the state of the **ECLK** pin. ❑ ***eclk*** = 1. **ECLK** is set high. ❑ ***eclk*** = 0. **ECLK** is set low. Also, this bit is used to indicate if no EEPROM is found. If no EEPROM is detected when an EEPROM load is attempted, this bit is held at 0 and cannot be set to a 1. When this condition occurs, setting ***stop*** in ***SysControl*** unlocks this bit, allowing software to interrogate the EEPROM. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **ECLK** pin.
1	***etxen***	EEPROM SIO transmit enable. This bit controls the direction of the **EDIO** pin. ❑ ***etxen*** = 1. **EDIO** is driven with the value in the ***edata*** bit. ❑ ***etxen*** = 0. ***edata*** is loaded with the value on **EDIO**. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **EDIO** pin.
0	***edata***	EEPROM SIO data. This bit is used to read or write the state of the **EDIO** pin. ❑ ***etxen*** = 1. **EDIO** is driven with the value in this bit. ❑ ***etxen*** = 0. This bit is loaded with the value on **EDIO**. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **EDIO** pin.

The external CPU must use ***eclk***, ***etxen***, and ***edata*** bits in the SIO register to read from and write to the **EEPROM**. The ***etxen*** bit determines whether the access is a read or write. The ***edata*** contains the bit being written or read on the serial interface and ***eclk*** is the clock. Data is written or read on the rising edge of ***eclk***.

Note:

The first bit written to or read from the EEPROM is the most significant bit of the byte, i.e., bit 7.

5.2.1 Performing Reads and Writes

All operations initiated between the external CPU and EEPROM must be preceded by a start condition and terminated by a stop condition. The external CPU is solely responsible for generating start and stop conditions. A start condition is generated by the high-to-low transition on the data line (***edata***) while the clock line is high (***eclk***). A stop condition is generated by the low-to-high transition on the data line (***edata***) while the clock line is high (***eclk***).

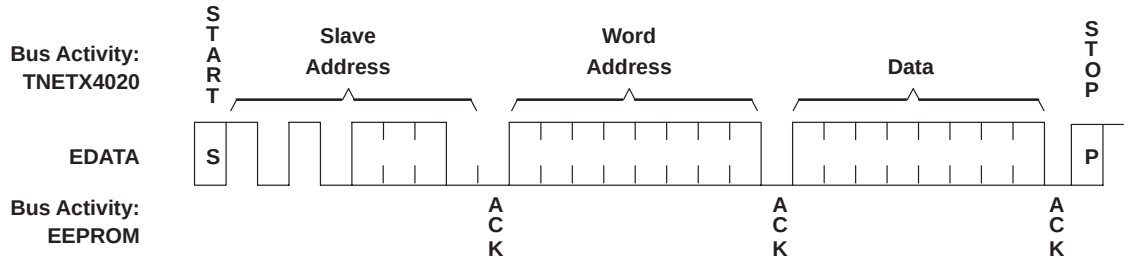
All read and write operations are performed using 9-bit block transfers where the first eight bits are data (slave address, word address, or data read/written) and the ninth bit is an acknowledge (ACK). Depending on the operation, the ACK is generated either by the external CPU or by the EEPROM. The following exception must be noted for all the read operations: the external CPU terminates the operation by generating a stop condition, omitting the ACK (see Figures 5–1 through 5–5). The ACK is generated by the external CPU/EEPROM (depending on the operation), pulling the data line (***edata***) low. Refer to the Figures 5–1 through 5–5 for the entity that generates the ACK.

The external CPU can perform or initiate the following accesses to the EEPROM:

Byte Write

One byte is written to an EEPROM address specified by the external CPU (see Figure 5–1).

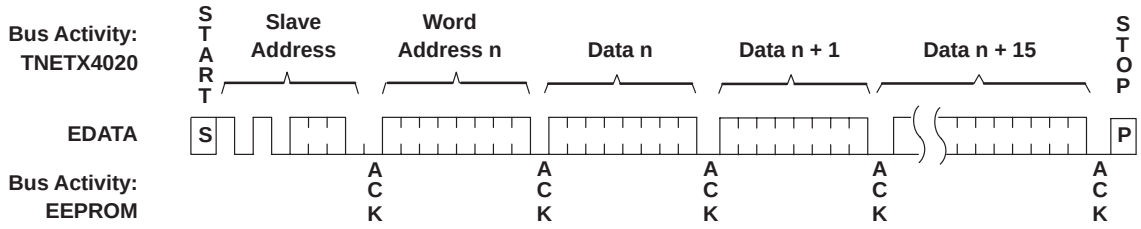
Figure 5–1. Byte Write



Page Write

A maximum of 16 bytes is written to the EEPROM. The starting address for the write is specified by the external CPU and the subsequent bytes are written to the EEPROM in incrementing order (see Figure 5–2).

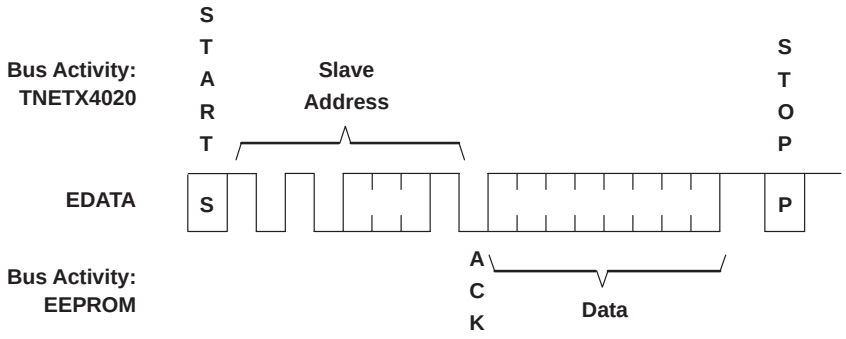
Figure 5–2. Page Write



Current Address Read

This is a read of one byte from the address location specified by the EEPROM's address counter (see Figure 5–3).

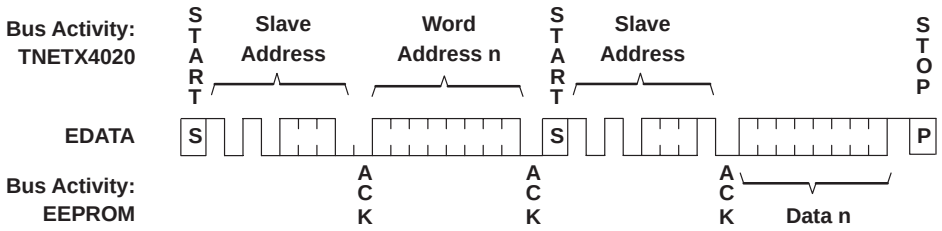
Figure 5–3. Current Address Read



Random Read

This is a read of one byte from a random address location specified by the external CPU (see Figure 5–4).

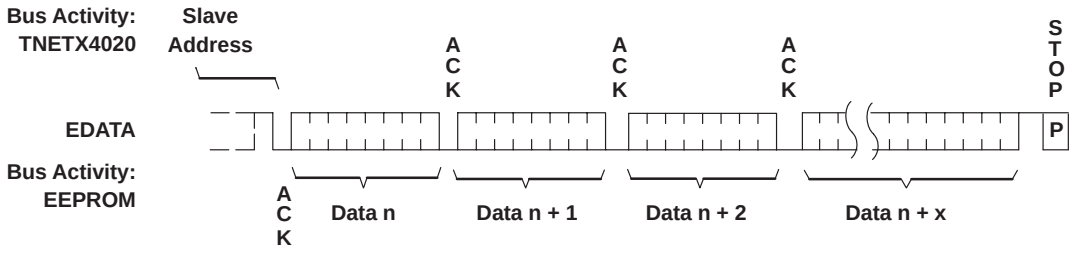
Figure 5–4. Random Read



Sequential Read

This is a read of one byte from the EEPROM, where the external CPU specifies the starting address. The bytes read out of the EEPROM are in incrementing order (see Figure 5–5).

Figure 5–5. Sequential Read



The slave address determines which device is connected to the TNETX4020, and indicates to the EEPROM whether the access is read or write.

Table 5–5. Slave Address Format (Bits 7:0)

Device Type Identifier (7:4)	Device Address	R/W
Bit(s) 7:4	The identifier for both 24C02 and 24C08 is: 0b1010	
Bit(s) 3:1	Device address of 24C02 = 0b000 Device address of 24C08 = 0b100	
Bit(s) 7:4	Read access: 1 Write access: 0	

The external CPU easily can determine which device is connected to the TNETX4020 by writing the slave address of the two devices. The EEPROM responds with an ACK to the slave address that matches its device address.

The word address comprises eight bits, which provide access to any byte within a 256-byte block. For the 24C08, the word address field is only the lower eight bits of the word address used by that device. The two least significant bits of the device address field (in the slave address) make up bits [9:8] of the word address, enabling the external CPU to access any byte in the 1024-byte EEPROM.

If a 24C02 device is used, it should be addressed at 0b000. If a 24C08 device is used, it should be addressed at 0b1xx. The TNETX4020 EEPROM load-state machine checks for an ACK at 0b000, then at 0b101. 0b101 is where the download starts with the 24C08 device because the TNETX4020 ***SysControl*** register must be the last data read out of the EEPROM in either case. Each address in an EEPROM corresponds directly to the register address that byte is applied to, subject to the read/write properties of the registers.

5.2.2 Pseudocode

The first of the following pseudocode procedures generates the start condition, writes the slave address, and verifies the EEPROM acknowledges the access. The second procedure describes the external CPU acknowledge for the previous 9-bit block, 8-bit read, and generation of the stop condition, which is the last part of any read operation.

This code is only a functional example for EEPROM accesses. Actual code should comprehend the other bits in the ***SIO*** register to avoid problems with physical-layer devices attached to the MDIO interface. Actual code also would comply with the timing required by the EEPROMs because the software has complete control of the EEPROM interface, including the timing. Even on this slow (~100 kHz) interface, care should be taken to change only one signal at a time to provide proper setup and hold times at the external pins.

```
// The external CPU generates the start condition, writes the slave address and
// checks for an acknowledge from the EEPROM. This is where the pseudo codes
// ends, but if there is no acknowledge the external CPU should put out a start
// condition and the slave address of the other EEPROM.

{
   *DIO_addr_lo = 0xA1;
   *DIO_addr_hi = 0x00;       // DIO address: SIO register

   // START CONDITON: HIGH to LOW transition on edata when eclk is HIGH

   *DIO_data = 0x07;          // eclk = 1, etxen=1, edata = 1. Creating a start
                              // condition

   *DIO_data = 0x06;          // eclk = 1, etxen=1, edata = 0. Start condition
                              // created

   // DEVICE TYPE IDENTIFIER: 1010

   *DIO_data = 0x03;          // eclk =0 , etxen=1, edata = 1. Loading a "1" for
                              // write

   *DIO_data = 0x07;          // eclk = 1, etxen=1, edata = 1. Clocking out a "1"

   *DIO_data = 0x02;          // eclk =0 , etxen=1, edata = 0. Loading a "0" for
                              // write

   *DIO_data = 0x06;          // eclk = 1, etxen=1, edata = 0. Clocking out a "0"

   *DIO_data = 0x03;          // eclk =0 , etxen=1, edata = 1. Loading a "1" for
                              // write

   *DIO_data = 0x07;          // eclk = 1, etxen=1, edata = 1. Clocking out a "1"

   *DIO_data = 0x02;          // eclk =0 , etxen=1, edata = 0. Loading a "0" for
                              // write

   *DIO_data = 0x06;          // eclk = 1, etxen=1, edata = 0. Clocking out a "0"

   // 24C02 DEVICE ADDRESS: 000

   *DIO_data = 0x02;          // eclk =0 , etxen=1, edata = 0. Loading a "0" for
                              // write
```

```
    *DIO_data = 0x06;          // eclk = 1, etxen=1, edata = 0. Clocking out a “0”

    *DIO_data = 0x02;          // eclk =0 , etxen=1, edata = 0. Loading a “0” for
                               // write

    *DIO_data = 0x06;          // eclk = 1, etxen=1, edata = 0. Clocking out a “0”

    *DIO_data = 0x02;          // eclk =0 , etxen=1, edata = 0. Loading a “0” for
                               // write

    *DIO_data = 0x06;          // eclk = 1, etxen=1, edata = 0. Clocking out a “0”

    // SPECIFY ACCESS TO BE A READ

    *DIO_data = 0x03;                // eclk =0 , etxen=1, edata = 1. Loading a “1”
                                     // for write

    *DIO_data = 0x07;                // eclk = 1, etxen=1, edata = 1. Clocking out
                                     // a “1”

    // CHECK FOR ACK: edata driven LOW by the EEPROM

    *DIO_data = 0x01;                // eclk =0 , etxen=0, edata = 1. Set up a read
                                     // to determine if the EEPROM has acknowledged
                                     // the transfer (edata set to 0)

    *DIO_data = 0x05;                // eclk =1 , etxen=0, edata = 1. Performing
                                     // the read.

    ack = *DIO_data;                 // read the SIO register to determine the ACK.
                                     // If ack equals 0x00 (this is true for this
                                     // case, but edata must be set to 0) then the
                                     // EEPROM connected is a 24C02. If no ACK was
                                     // generated,then the external CPU must repeat
                                     // the above codewith a Device Code of 0b100.
}

// The external CPU generates an ACK, reads the 8-bit block of data and then
// generates a stop condition.
// The psuedo code assumes that the DIO address has been configured to 0x00A1 (i.e.,
// the SIO register.

{
    // GENERATING AN ACK: driving edata LOW.

    *DIO_data = 0x02;                // eclk = 0, etxen=1, edata = 0. Loading a
                                     // “0” for write

    *DIO_data = 0x06;                // eclk = 1, etxen=1, edata = 0. Clocking out
                                     // a “0”

    // READ 8 BITS OF DATA
    for (i=0; i<8; i++)

    {
```

```
      *DIO_data = 0x00;             // eclk =0 , etxen=0, edata = 0. Set up a read.

      *DIO_data = 0x04;             // eclk =1 , etxen=0, edata = 0. Performing
                                    // the read.

    if (i = 0)
      data = edata;                 // first bit is stored in data
   else
      data = (data << 1) | edata;   // 8-bit block is stored in a data (of type
                                    // BYTE)
   }

   // GENERATING A STOP CONDITION: transition of edata from LOW to HIGH while eclk
   // is HIGH
   // Note that the 9th clock cycle ACK is omitted

   *DIO_data = 0x07;                // eclk = 1, etxen=1, edata = 0. Creating a stop
                                    // condition

   *DIO_data = 0x06;                // eclk = 1, etxen=1, edata = 1. Stop condition
                                    // created

}
```

5.2.3 Low-Level Support Routines From TSMAN

Low-level EEPROM read and write routines used in TSMAN, a program for setting up and observing ThunderSwitch II-based Ethernet switches, are presented in this section. A computer printer port is used to access the DIO interface of TNETX4020 devices. Schematics showing how to connect TI evaluation modules (EVMs) are available from TI. The hardware-level prototypes for sending a byte through the printer port, retrieving a byte through the printer port, and handling the control signals are in the header routines in Appendix B. Source code for TSMAN for the printer port is available from TI.

The following code is the reference to the header information.

```
//-------------------------------------------------------------------------
// Thunderswitch interface functions
//-------------------------------------------------------------------------
// File: eeprom.c
//
// This module contains defined the following functions associated with
// accessing Thunderswitch hardware:
//
// TsEeRd      – Read bytes of data from Thunderswitch's EEPROM
// TsEeWr      – Write bytes of data to Thunderswitch's EEPROM
// TsEeRdByte  – Read a byte of data from Thunderswitch's EEPROM
// TsEeWrByte  – Write a byte of data to Thunderswitch's EEPROM
//
// The following functions are general helper functions, and are
// static to this module.
//
// TS_EeAck    – check for ack from eeprom
// TS_EeAddr   – send address to access for read/write to eeprom.
// TS_EeSel    – select EEPROM device to access
// Set         – Set a bit in the SIO register
// Clr         – Clr a bit in the SIO register
// Tog         – Toggle (Set, Clr) a bit in the SIO register
//
// This module uses the following hardware specific functions.
// These are dependent on the system specific processor to TSwitch
// interface and must be ported.
//
// InByte      – Read byte from TSWITCH Host Register
// OutByte     – Write byte to TSWITCH Host Register
//
//-------------------------------------------------------------------------

#include "tsif.h"
#include "tsifhw.h"

static int     TS_EeAck( );
static void    TS_EeAddr( WORD wAddr );
static void    TS_EeSel( WORD wCmd );
static void    Set( BYTE, BYTE* );
static void    Clr( BYTE, BYTE* );
static void    Tog( BYTE, BYTE* );

// Macros
#define READ         1
#define WRITE        0
#define ACCESS_SEQ   0x50  // == 01010000
                           // 1010 = device type identifier
                           // 000  = first device on bus
                           // (MSbit is ignored in the code)
```

The following code supports single-byte and multiple-byte reads.

```
//-------------------------------------------------------------------------
//  TsEeRdByte()     Read a byte of data from Thunderswitch's EEPROM
//-------------------------------------------------------------------------
//  Parameters:
//   WORD  wAddr     Address of EEPROM registers to read
//   BYTE* pbData    Buffer to receive data
//
//  Return val:
//   nonzero if success
//-------------------------------------------------------------------------
int TsEeRdByte( WORD wAddr, BYTE* pbData )
{
   BYTE  bSIO,
         i,
         bTmp = 0x00;

   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   // send eeprom start sequence

   Set( ECLOK, &bSIO );
   Set( EDATA|ETXEN, &bSIO );
   Clr( EDATA, &bSIO );
   Clr( ECLOK, &bSIO );

   // select EEPROM device to use and set up to write address to read
   // from out to device
   TS_EeSel( WRITE );

   // if the EEPROM doesn't ACK, something's wrong
   if( !TS_EeAck() )
      return (0);

   // Send address in EEPROM to read from
   TS_EeAddr( wAddr );

   // if the EEPROM doesn't ACK, something's wrong
   if( !TS_EeAck() )
      return (0);

   // The Sel, Ack and Addr routines have changed what's in SIO_XCTRL
   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   // send start access sequence for the read
   Set( ETXEN, &bSIO );
   Set( ECLOK, &bSIO );
   Set( EDATA|ETXEN, &bSIO );
   Clr( EDATA, &bSIO );
   Clr( ECLOK, &bSIO );

   // send EEPROM device selection sequence and set up for read
   TS_EeSel( READ );

   if( !TS_EeAck() )
      return (0);

   *pbData = 0;
   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   for( i = 0x80; i; i >>= 1 )
   {
      Set( ECLOK, &bSIO );
```

```
      TsDioRd( TS_SIO_XCTRL, 1, &bTmp );
      if( bTmp & EDATA )
         *pbData |= i;

      Clr( ECLOK, &bSIO );
   }

   Tog( ECLOK, &bSIO );

   // send stop access sequence
   Clr( EDATA, &bSIO );
   Set( ECLOK, &bSIO );
   Set( EDATA, &bSIO );
   Clr( ETXEN, &bSIO );
   return (1);
}

//---------------------------------------------------------------------------
//  TsEeRd() – Read data from Thunderswitch's EEPROM
//---------------------------------------------------------------------------
//  Parameters:
//   WORD  wAddr     Address of EEPROM registers to read
//   BYTE* pbData    Buffer to receive data
//   int   nCount    number of bytes to read
//
//  Return val:
//   nonzero if success
//---------------------------------------------------------------------------
int TsEeRd( WORD wAddr, int nCount, BYTE* pbData )
{
    while (nCount--)
    {
        if (!TsEeRdByte(wAddr,pbData))
            return 0;
        wAddr += 1;
        pbData += 1;
    }
    return 1;
}
```

The following code supports single-byte and multiple-byte writes of information to DIO registers.

```
//-------------------------------------------------------------------------
//
//  TsEeWrByte()     Write a single byte to Thunderswitch's
//                   EEPROM.
//


//  Parameters:
//   WORD  wAddr      Address of EEPROM location to write
//   BYTE  bBuf       Data to write
//
//  Return val:
//   nonzero if success
//-------------------------------------------------------------------------
int TsEeWrByte( WORD wAddr, BYTE bData )
{
   BYTE  bSIO,
         i;

   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   // send eeprom start sequence

   Set( ECLOK, &bSIO );
   Set( EDATA, &bSIO );
   Set( ETXEN, &bSIO );
   Clr( EDATA, &bSIO );
   Clr( ECLOK, &bSIO );

   // select EEPROM device to use and set up to write address to read
   // from out to device
   TS_EeSel( WRITE );

   // if the EEPROM doesn't ACK, something's wrong
   if( !TS_EeAck() )
      return (0);

   // Send address in EEPROM to read from
   TS_EeAddr( wAddr );

   // if the EEPROM doesn't ACK, something's wrong
   if( !TS_EeAck() )
      return (0);

   // The Ack, Sel and Addr routines have changed SIO_XCTRL
   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );
   Set( ETXEN, &bSIO );

   for( i = 0x80; i; i >>= 1 )
   {
      if( i & bData )
         Set( EDATA, &bSIO );
      else
         Clr( EDATA, &bSIO );

      Tog( ECLOK, &bSIO );
   }
```

```
   if( !TS_EeAck() )
      return (0);

   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   // send stop access sequence
   Set( ETXEN, &bSIO );
   Clr( EDATA, &bSIO );
   Set( ECLOK, &bSIO );
   Set( EDATA, &bSIO );
   Clr( ETXEN, &bSIO );

   // wait until eeprom writes the data
   do
   {
      TsDioRd(TS_SIO_XCTRL, 1, &bSIO );
      // start sequence
      Set( ECLOK, &bSIO );
      Set( EDATA, &bSIO );
      Set( ETXEN, &bSIO );
      Clr( EDATA, &bSIO );
      Clr( ECLOK, &bSIO );
      TS_EeSel( WRITE );
   } while ( !TS_EeAck() );
   return 1;
}

//-------------------------------------------------------------------------
//  TsEeWr() – Write data to Thunderswitch's EEPROM
//-------------------------------------------------------------------------
//  Parameters:
//   WORD  wAddr     Address of EEPROM registers to read
//   BYTE* pbData    data to write
//          int          nCount              number    of    bytes    to    write

//
//  Return val:
//   nonzero if success
//-------------------------------------------------------------------------
int TsEeWr( WORD wAddr, int nCount, BYTE* pbData )
{
    while (nCount--)
    {
        if (!TsEeWrByte(wAddr,*pbData))
            return 0;
        wAddr += 1;
        pbData += 1;
    }
    return 1;
}
```

These are the helper routines mentioned in the header.

```
//---------------------------------------------------------------------------
//  TS_EeSel() –    select EEPROM device to access (see Excel XL24C02
//                  device specification)
//
//  Parameters:
//    cmd           WORD specify read or write operation
//
//  Return val:
//    void
//---------------------------------------------------------------------------
static void TS_EeSel( WORD wCmd )
{
   BYTE  bSIO,
         i;

   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   for( i = 0x40; i; i>>= 1 )
   {
      if( i & ACCESS_SEQ )
         Set( EDATA, &bSIO );
      else
         Clr( EDATA, &bSIO );
      Tog( ECLOK, &bSIO );
   }
   if (wCmd)
      Set( EDATA, &bSIO );
   else
      Clr( EDATA, &bSIO );

   Tog( ECLOK, &bSIO );
}

//---------------------------------------------------------------------------
//  TS_EeAddr() –    send address to access for read/write to eeprom.
//
//  Parameters:
//   addr           WORD address on eeprom to access
//
//  Return val:
//   void
//---------------------------------------------------------------------------
static void TS_EeAddr( WORD wAddr )
{
   BYTE  i,
         bSIO;

   TsDioRd( TS_SIO_XCTRL, 1, &bSIO );

   Set( ETXEN, &bSIO );

   for( i = 0x80; i; i >>= 1 )
   {
      if( wAddr & i )
         Set( EDATA, &bSIO );
      else
         Clr( EDATA, &bSIO );

      Tog( ECLOK, &bSIO );
   }
}
```

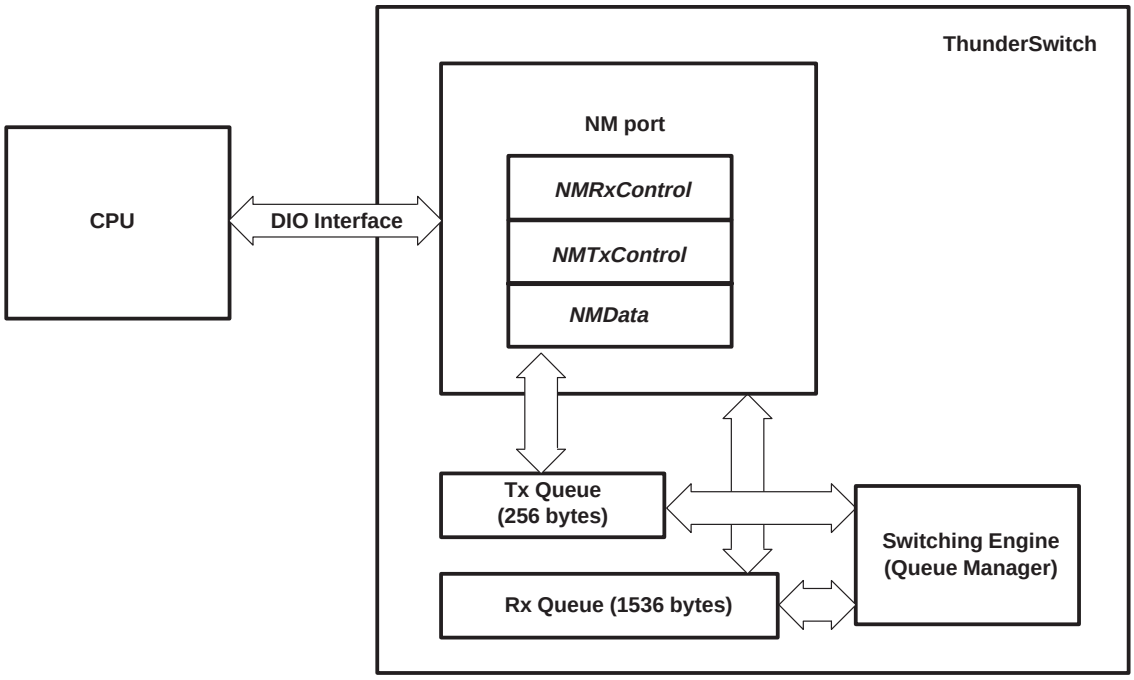
```
//---------------------------------------------------------------------
//  TS_EeAck() –     check for ack from eeprom
//
//  Parameters:
//   none
//
//  Return val:
//   int             nonzero if ack received, 0 otherwise
//---------------------------------------------------------------------
static int TS_EeAck( )
{
   BYTE bSIO,
        bTmp;

   if ( TsDioRd(TS_SIO_XCTRL, 1, &bSIO) )
      return (0);

   Clr( ETXEN, &bSIO );
   Set( ECLOK, &bSIO );

   if ( TsDioRd(TS_SIO_XCTRL, 1, &bTmp) )
      return (0);

   Clr( ECLOK, &bSIO );

   if (bTmp & EDATA)
      return(0);
   else
      return(1);
}

//---------------------------------------------------------------------
//  Set() –     Set a bit in the SIO register
//
//  Parameters:
//   BYTE  bFlags    bits to set
//   BYTE* pbSIO     current value of the SIO register
//
//  Return val:
//   void
//---------------------------------------------------------------------
static void Set( BYTE bFlags, BYTE* pbSIO )
{
   *pbSIO |= bFlags;
   TsDioWr( TS_SIO_XCTRL, 1, pbSIO );
}

//---------------------------------------------------------------------
//  Clr() –     Clr a bit in the SIO register
//
//  Parameters:
//   BYTE  bFlags    bits to set
//   BYTE* pbSIO     current value of the SIO register
//
//  Return val:
//   void
//---------------------------------------------------------------------

static void Clr( BYTE bFlags, BYTE* pbSIO )
{
   *pbSIO &= ~bFlags;
   TsDioWr( TS_SIO_XCTRL, 1, pbSIO );
}
```

```
//-------------------------------------------------------------------------
//  Tog() –     Toggle (Set, Clr) a bit in the SIO register
//
//  Parameters:
//   BYTE  bFlags    bits to toggle
//   BYTE* pbSIO     current value of the SIO register
//
//  Return val:
//   void
//-------------------------------------------------------------------------
static void Tog( BYTE bFlags, BYTE* pbSIO )
{
   Set( bFlags, pbSIO );
   Clr( bFlags, pbSIO );
}
```

5.3 Network Management Port

The TNETX4020 allows the external CPU to receive and transmit frames using the network management (NM) port. The frames can be read out of the NM port and written into the NM port using the ***NMRxControl***, ***NMTxControl***, and ***NMData*** registers. As shown in Figure 5–6, dedicated Tx and Rx queues are used to buffer the frame data passing through the NM port. See the *Using the Tswitch Network Management Port for Frame I/O BTW Tswitch Devices/CPUs* application report, literature number SPWA010, for detailed information on how to use the NM port to transmit and receive frames.

Figure 5–6. NM Port Block Diagram



The NM port can intermix reception and transmission as desired. The direction of the ***NMData*** access (read or write) determines whether a byte is removed from the transmit queue or added to the receive queue. The DIO interface itself is only half duplex because it cannot do a read and a write at the same time.

Note:

The NM port registers are described from the point of view of the TNETX4020, not the CPU. Thus, the ***NMRxControl*** register is used when the TNETX4020 is *receiving* a frame from the CPU. Similarly, when the CPU is getting a frame from TNETX4020, the ***NMTxControl*** register is used because TNETX4020 is *transmitting* the data to the CPU.

5.3.1 Frame Format on the NM Port

Frames traversing the NM port differ slightly from the standard Ethernet frame format. A frame received or transmitted on the NM port must contain an IEEE Std 802.1q-like header in the four bytes following the source address. The byte-numbering scheme used in this section numbers bytes in the order that they appear on the wire. Thus, byte 1 is first, byte 2 is next, etc.

The standard Ethernet frame format is shown in Figure 5–7.

Figure 5–7. Ethernet Frame Format

Destination Address	Source Address	Length/Type	Data	FCS (CRC-32)
6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

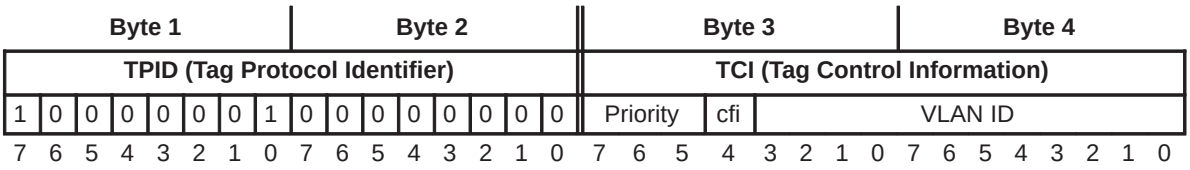
An Ethernet frame with an IEEE Std 802.1q header looks like:

Figure 5–8. Ethernet Frame With an IEEE Std 802.1q Tag Header

Destination Address	Source Address	802.1Q Header		Length/Type	Data	FCS (CRC-32)
		TPID	TCI			
6 bytes	6 bytes	4 bytes		2 bytes	46-1500 bytes	4 bytes

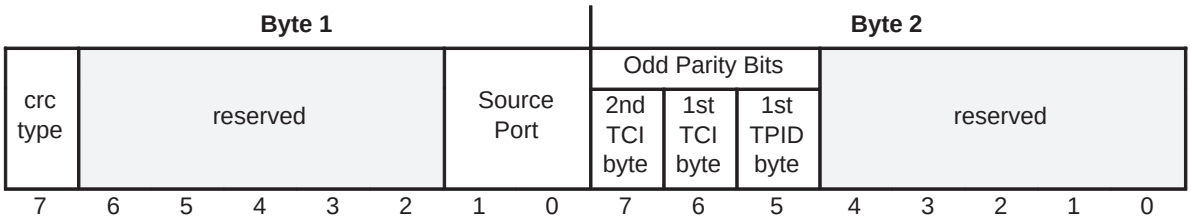
The 4-byte IEEE Std 802.1q header contains a 2-byte tag protocol identifier (TPID, whose value is 81–00), and two bytes of tag control information (TCI) as shown in Figure 5–9.

Figure 5–9. IEEE Std 802.1q Tag Header



The 2-byte TCI field of the ThunderSwitch IEEE Std 802.1q header is identical to the IEEE Std 802.1q definition. However, the TPID field of the ThunderSwitch IEEE Std 802.1q header is used within ThunderSwitch for other purposes, so a frame traversing the NM port will *not* have the IEEE Std 802.1q TPID value of 81–00 in these two bytes. Instead, these two bytes are encoded as shown in Figure 5–10.

Figure 5–10. ThunderSwitch Encoding of TPID Field



If ***crctype*** = 1, the CRC word in the frame will exclude the 4-byte TPID/TCI header, that is, the tag field can be removed without recalculating the CRC.

Frames read out of the NM port are in the format described in Table 5–10. Furthermore, frames written into the switch also must conform to this format. Refer to the *Using the Tswitch Network Management Port for Frame I/O BTW Tswitch Devices/CPUs* application report, literature number SPWA010, for a detailed description of the frame format.

5.3.2 Sending a Frame to the NM Port

To send a frame to the NM port, the external CPU must use the ***NMRxControl*** and ***NMData*** registers. The ***nmrx*** bit in the ***Int*** register or the **SRXRDY** pin can be used because it indicates when the NM port's receive buffer is empty, i.e., the TNETX4020 can receive a maximum-size frame.

NMRxControl (Network Management Receive Control Register) @ 0x0818–0x081A

The NM port is treated as Port 2 internally.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved			***freebuffs[4:0]***					***crc***	***eof***	***alen***	reserved			***portcode[1:0]***		
Reset Value	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	hs	hs	hs	hs	hs	hs	hs	hs	–	–	–	hs	hs	
Field Type	r	r	r	r	r	r	r	r	rw	rw	rw	r	r	r	rw	rw	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	hs	hs	hs	hs	hs	hs	
Field Type	r	r	rwh	rwh	rwh	rwh	rwh	rwh	

Table 5–6. NMRxControl (Network Management Receive Control Register)

Bit	Name	Description
23:22	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
21:16	***xroutecode***	If ***portcode*** = 1 (output port = 1), and ***alen*** = 0 (override internal lookup results), and ***pretag*** = 1 in ***Port1Control*** (port 1 connected to a crossbar), then ***xroutecode*** completely determines the destination ports on the crossbar, that is, the destination address is ignored. Otherwise, ***xroutecode*** is not used. The value used is that present before any data for the packet is sent to the switch. It may be changed for the next packet after data loading has started for the current packet.
15:13	reserved	Reserved. Writes to these bits have no effect. They always read as 0.

Table 5–6. NMRxControl (Network Management Receive Control Register) (Continued)

Bit	Name	Description
12:8	***freebuffs***	Free buffers available. This field indicates the number of 64-byte buffers that can be written safely to ***NMData*** without risk of SRDY being held inactive high to hold off further writes. This value is valid only after each complete burst of 64 bytes has been written to ***NMData*** or after ***NMRxControl*** has been written with ***eof*** = 1, i.e., the value is undefined if a noninteger multiple of 64 bytes has been written to ***NMData*** without having also written a 1 to ***eof***. The maximum value for this field is 24, equivalent to 1536 bytes available. The **SRXRDY** pin is high if this field is 24; otherwise, it is low. The ***nmrx*** bit in the ***Int*** register is set to 1 when this field reaches 24, i.e., when SRXRDY transitions from low to high. If the NM port or the entire switch enters a flow-controlled situation, this register appears to contain a zero to prevent the host from sending more data into the NM port. When the congestion clears, this register reflects the actual number of free buffers available.
7	***crc***	CRC generation. Determines whether the device should insert the CRC word into the received frame. Must be valid when ***eof*** is written with a 1 (ignored at all other times). ❑ ***crc*** = 1. The last four bytes of received frame data are replaced automatically with valid CRC for that frame. ❑ ***crc*** = 0. The received frame is expected to contain a valid CRC word.
6	***eof***	End of frame. This bit is set to 1 after all data for a frame have been written to ***NMData***. It clears to 0 when the frame has been received. Further writes to ***NMRxControl*** or ***NMData*** hold SRDY high until the frame has been received and this bit returns to 0. ***crc*** should be written with the desired value at the same time this bit is written with a 1.
5	***alen***	Address-lookup enable. Indicates how the frame to be written into ***NMData*** should be forwarded. Must be valid before any data for the frame is written into ***NMData***. ❑ ***alen*** = 1. The normal address lookup determines the destination of the frame. ***alen*** = 0. Portcode determines the destination for the frame.
4:2	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
1:0	***portcode***	Transmit port number. Indicates to which port the frame is to be sent if ***alen*** = 0 (ignored if ***alen*** = 1). Must be valid before any data for the frame is written into ***NMData***.

NMData (Network Management Data Register) @ 0x0820

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***nmdata[7:0]***								
Reset Value	–	–	–	–	–	–	–	–	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	

Table 5–7. NMData (Network Management Data Register)

Bit	Name	Description
7:0	***nmdata***	Network management data. An external CPU can use this address to write data for transmission or read received data on the NM port (port 2). ❑ On transmit, ***NMTxControl*** should be read before reading from each 256 buffer via ***NMData*** to determine how many bytes are in the buffer. ❑ On receive, ***NMRxControl*** should be written before and after writing an entire frame to ***NMData.*** If the transmit FIFO for the port becomes full while writing to ***NMData***, the ***$\overline{SRDY}$*** pin is held high to prevent data being lost. This should happen only in exceptional circumstances in which the RDRAM memory is full.

Note:

The **SRXRDY** signal and the ***nmrx*** interrupt are set when the receive FIFO is empty. This indicates that the NM port is ready to accept a frame of any length (up to 1535 bytes).

The NM receive interrupt is issued only if the end of a frame was loaded into the NM receive buffer to be processed. Host software responsible for sending frames on the NM port should adopt the practice of sending all of an inbound frame to the buffer for maximum efficiency (frame is not sent into the fabric until end has arrived from host), and minimize chances of stalling the interrupt service routine with part of a frame in the buffer and the ISR waiting for an all-clear signal. Oversize frames from the host to the NM port also can cause ISR stalls for the same reason.

The following general steps are used by the CPU to send a frame to the NM port:

1) Ensure that the switch is able to receive the frame. This can be detected in three ways:
   - ❑ The ***nmrx*** bit in the ***Int*** register is set to 1 when the NM receive buffers are empty. This indicates that all previous frame data received on the NM port has been transferred to external memory and the NM port is now able to receive a frame of any size up to 1535 bytes in length.
   - ❑ **SRXRDY** = 1 means that the RX FIFO has 24 free buffers (switch can receive a frame of any size up to 1535 bytes). The transition of **SRXRDY** from 0 to 1 generates the ***nmrx*** interrupt.
   - ❑ If the ***freebuffs*** field in ***NMRxControl*** is nonzero, some frame data can be received. ***freebuffs*** indicates how many blocks of 64 bytes can be written to ***NMData*** (see step 3) without risk of **$\overline{SRDY}$** being held inactive high, thus stalling the DIO interface.

2) A write is performed to ***NMRxControl*** with the following parameters:

   - ***eof*** = 0. If ***eof*** = 1, the frame would be prematurely received.
   - Set the ***alen*** bit as desired.
     - ***alen*** = 1. The IALE examines the frame and determines its routing using its normal frame-routing algorithm. The ***portcode*** field in this case is ignored.
     - ***alen*** = 0. The portcode field must specify the desired destination port number. If an invalid port number is specified, the frame is ultimately discarded and the NM port's Rx Align/Code Errors statistic is incremented.
   - The ***crc*** bit value is ignored at this time.

3) The frame data is written as a stream of bytes in native Ethernet format to the ***NMData*** register until all the frame's bytes have been written. If, during the course of this, the switch cannot continue to receive data, **$\overline{\text{SRDY}}$** is held high until it can receive data, causing the DIO interface to stall. To avoid such stalls, not more than 64 * ***freebuffs*** bytes of data should be written to ***NMData*** without rechecking the ***freebuffs*** value. The ***freebuffs*** value is valid only after integer quantities of 64 bytes have been written (or ***eof*** has been written with a 1).

4) A write to ***NMRxControl*** is performed with:

   - ***eof*** = 1. (If ***eof*** = 0, nothing happens).
   - The ***crc*** bit is set as required:
     - ***crc*** = 1. CRC is inserted by the switch. The final four bytes written to ***NMData*** are replaced by the CRC word calculated by the switch. In this case, the TPID field parity fields are ignored, and no parity checking takes place. The switch sets the parity bits and generates a valid CRC for the assumed good data supplied from the host.
     - ***crc*** = 0. The frame supplied by the CPU must already contain a valid CRC word and header parity protection. If the CRC is not valid or the parity protection indicates an error, the frame is discarded and the NM port's Rx CRC Errors statistic is incremented.
     - The ***alen*** bit and ***portcode*** field are ignored at this time.

The ***eof*** bit in ***NMRxControl*** remains at 1 until the frame has been received successfully. Any further writes to ***NMRxControl*** or ***NMData*** cause **$\overline{\text{SRDY}}$** to be held high until the ***eof*** bit is 0. Reads of byte 1 of ***NMRxControl*** (***freebuffs***) are similarly blocked while ***eof*** = 1 (until ***eof*** returns to 0) because the amount of free space is not known until the frame is received into the switch. When the frame has been received successfully, it is transmitted to the destination port(s) in the manner specified in ***NMRxControl***. The NM port then is ready to receive another frame from the CPU by repeating these steps.

5.3.2.1 Specifying How the Frame Is Routed Through the TNETX4020

The NM port provides two methods for the CPU to specify how a frame sent from the CPU to the switch is routed through and out of the switch. Refer to the flowchart (see Figure 5–11) and code listings (see section 5.3.3.5).

- The CPU can specify directly the destination port number. This bypasses the frame-routing algorithm used by the IALE. This method is useful if the CPU already knows the correct destination port for the frame being sent to the NM port.

  To specify the destination port number, the CPU sets ***alen*** = 0 and writes the desired destination port number in the ***portcode*** field of ***NMRxControl*** before starting frame-data transmission to the NM port. The port number must be in the range of 0 to the number of ports available on the TNETX4020 minus 1. If an invalid destination port is specified, the frame ultimately is discarded and the NM port's Rx Align/Code Errors statistic is incremented.

- The CPU can specify that the IALE be used to determine the frame's routing using its normal frame-routing algorithm. This is done by setting ***alen*** = 1 in the ***NMRxControl*** register before starting frame-data transmission.

5.3.3 TSWriteFrame – Flowchart and Sample Code

Once the MAC frame has been built according to the required NM port frame format rules, the frame can be written to the NM port. In this section, an example is given of a simple blocking (does not return until completed) implementation of such a function, TSWriteFrame.

The function prototype for TSWriteFrame is described in the following paragraphs.

5.3.3.1 TSWriteFrame – Write a Frame to the NM Port

```
typedef unsigned int    WORD;
typedef unsigned char   BYTE;
typedef unsigned char   BOOL;
```

Syntax

int TSWriteFrame (BYTE *FrameData, WORD FrameBytes, BYTE PortCode, BOOL NeedsCRC)

Parameters

FrameData	Pointer to start of MAC frame data
FrameBytes	Total bytes in the frame (including CRC)
PortCode	Port routing code, or 0xFF to specify IALE routing
NeedsCRC	0 if frame already includes CRC and odd-parity header-protection bits, nonzero if switch needs to add these

Return Value

0 on error; 1 if frame is received by NM port OK

Note:

The example code in this document uses two functions to access the internal registers from the management CPU. The code for these functions is not provided here, as implementations vary by platform environment.

5.3.3.2 DioRdByte – Read a Byte From an Internal Register

```
typedef unsigned int    WORD;
typedef unsigned char   BYTE;
```

Syntax

void DioRdByte(WORD DioAddress, BYTE *data)

Parameters

DioAddress	Byte address of DIO register to be read
data	Pointer to variable where value will be stored

Return Value

None

5.3.3.3 DioWrByte – Write a Byte to an Internal Register

```
typedef unsigned int    WORD;
typedef unsigned char   BYTE;
```

Syntax

void DioWrByte(WORD DioAddress, BYTE data)

Parameters

DioAddress	Byte address of DIO register (to be written to)
Data	Value to write

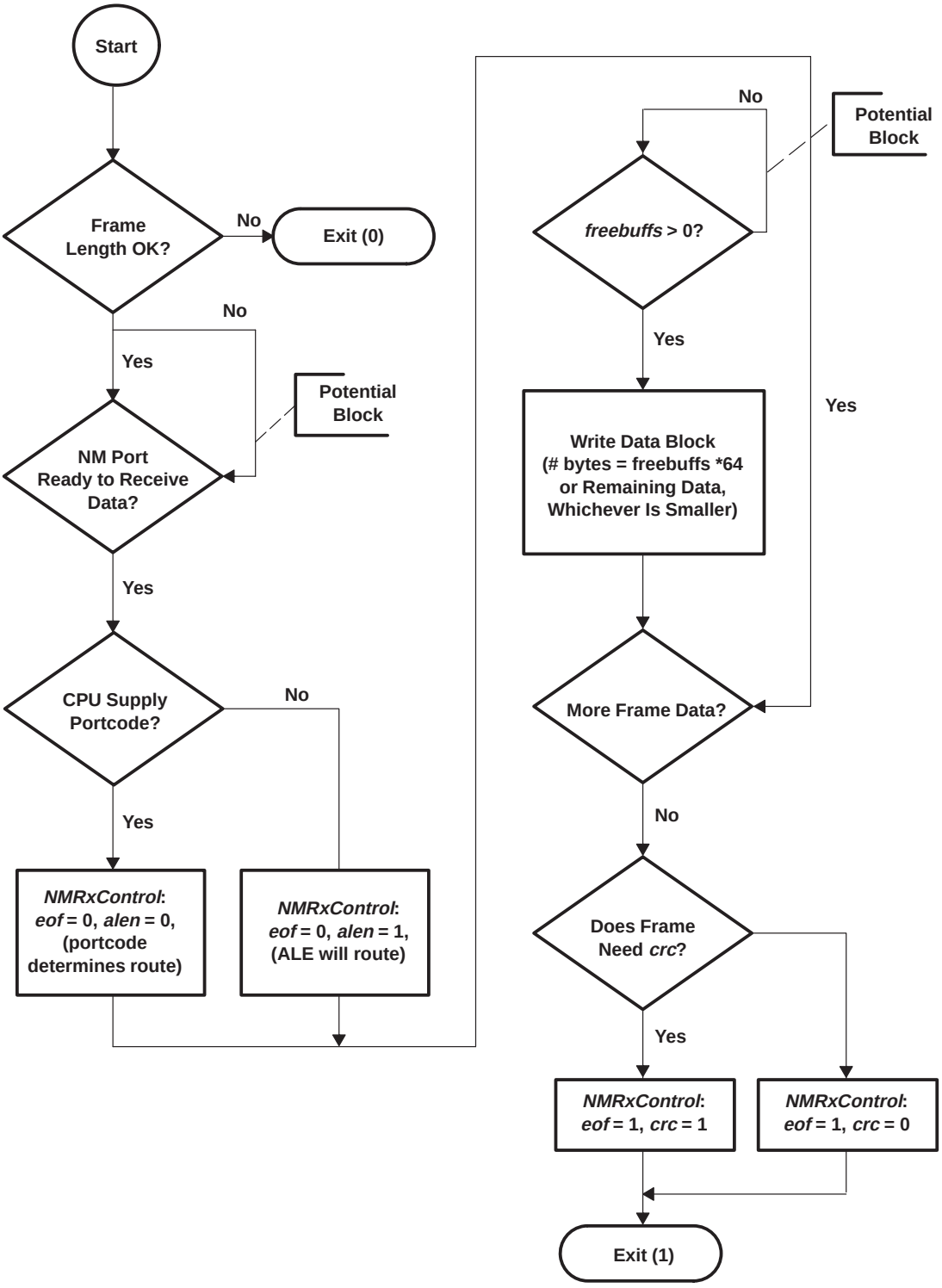
Return Value

None

5.3.3.4 TSWriteFrame – Flowchart

Figure 5–11 is the flowchart for the TSWriteFrame example function. Paragraph 5.3.3.5 contains a code listing for this function.

Figure 5–11. TSWriteFrame Flowchart



5.3.3.5 TSWriteFrame – Code Listing

```
typedef unsigned intWORD;
typedef unsigned char   BYTE;
typedef unsigned char   BOOL;

//TNETX4020 DIO register locations
#define NMRxControl    0x0818
#define NMData         0x0820

// NMRxControl register bit fields
#define PORTCODE_MASK  0x001F
#define ALEN           0x0020
#define EOF            0x0040
#define CRC            0x0080
#define FREEBUFFS_MASK 0x1F00

#define OK          1
#define ERROR       0

int TSWriteFrame( FrameData, FrameBytes, PortCode, NeedsCRC )
   BYTE   *FrameData;      // Pointer to Frame Data
   WORD   FrameBytes;      // Total # of bytes in the frame
   BYTE   PortCode;        // Port routing code, or 0xFF for IALE routing
   BOOL   NeedsCRC;        // 0 if frame already includes CRC
{
   BYTE   data;
   WORD   BytesToWrite;

   //-------------------------------------------------------------------------
   // Validate frame length. Must be >= 64 bytes and <= 1535 bytes.
   //-------------------------------------------------------------------------
   if( FrameBytes < 64 || FrameBytes > 1535 )
      return ERROR;
   //-------------------------------------------------------------------------
   // Let's insure that TNETX4020 has finished receiving any prior frames into
   // its RX FIFO and that NMData and NMRxControl are available for writing.
   //-------------------------------------------------------------------------
   do
      DioRdByte( NMRxControl, &data );
   while (data & EOF);     // EOF=1 means switch is still processing prior frame
   //-------------------------------------------------------------------------
   // Tell NM port we're ready to start sending it frame data. This is done by
   // setting up NMRxControl as follows:
   // 1.  Set EOF=0
   // 2a. If no valid PortCode is specified (PortCode=0xFF), then set ALEN=1.
   //     The IALE will examine the frame and determine its routing using its
   //     normal frame routing algorithm.
   // 2b. If a valid PortCode value is specified, then set ALEN=0 and set up
   //     portcode field in NMRxControl.
   // 3.  CRC bit is ignored.
   //-------------------------------------------------------------------------
   data = 0;               // Force ALEN=0, EOF=0
   if( PortCode == 0xFF )
      data |= ALEN;        // No PortCode specified. Use IALE.
   else
      data |= (PortCode & PORTCODE_MASK);  // PortCode supplied. ALEN=0 and
                                           // valid portcode
   DioWrByte( NMRxControl, data );
```

```
   //-------------------------------------------------------------------------
   // Transfer the frame to the switch. Since the NM port's RX FIFO may
   // contain other frames waiting to be forwarded, we must check the amount
   // of space remaining in the FIFO (via freebuffs) to insure we do not
   // exceed this amount. If the FIFO space is exceeded, we run the risk of
   // stalling the DIO i/f.
   //
   // Freebuffs indicates the number of 64 byte buffers that may safely be
   // written to NMData without risk of SRDY# being held inactive-high to
   // hold off further writes.
   //-------------------------------------------------------------------------
   do
   {
      DioRdByte( NMRxControl+1, &data );
      freebuffs = data & FREEBUFFS_MASK;
      if( freebuffs > 0 )
      {
         BytesToWrite = (FrameBytes <= (freebuffs*64)) ? FrameBytes :
                                (freebuffs*64));
         FrameBytes -= BytesToWrite;
         while( BytesToWrite-- )
            DioWrByte( NMData, *FrameData++ );
      }
   } while FrameBytes;
   //-------------------------------------------------------------------------
   // Tell TNETX4020 all frame data has been transferred. This is done by setting
   // up NMRxControl as follows:
   // 1. Set EOF=1
   // 2. Set the CRC mode. If the frame already contains a CRC and header
   //    parity protection, (NeedsCRC == 0), then set CRC=0. However, if the
   //    switch needs to insert the CRC (NeedsCRC != 0), then set CRC=1.
   // 3. ALEN and portcode are ignored.
   //-------------------------------------------------------------------------
   data = EOF;         // EOF=1, CRC=0
   if( NeedsCRC )
      data |= CRC;     // CRC needed. Set CRC=1
   DioWrByte( NMRxControl, data );
   //-------------------------------------------------------------------------
   // TNETX4020 will now perform the final steps required to complete the
   // receipt of our frame into its RX FIFO. Because this is not an
   // instantaneous operation, EOF will transition from 1 to 0 only after this
   // operation completes. In the meantime, writes to NMData and NMRxControl
   // are blocked.
   //
   // Since we put a check at the beginning of this function to insure that
   // TNETX4020 has completed any prior frame reception on its NM port, we can
   // simply exit now. If this check wasn't performed at the beginning, we'd
   // likely wait here for EOF to go to 0.
   //-------------------------------------------------------------------------
   return OK;
}
```

5.3.4 Getting a Frame From the NM Port

To get a frame from the NM port, the external CPU must use the ***NMTxControl*** and ***NMData*** registers. The ***nmtx*** bit in the ***Int*** register can be used because it indicates when frame data is available to be read out of the NM port's Tx buffer.

NMTxControl (Network Management Transmit Control Register) @ 0x081C–0x081E

The NM port is treated as Port 2 internally.

The **STXRDY** pin outputs a 1 if any of the ***eof***, ***sof***, or ***iof*** bits is set to 1; otherwise, it outputs 0.

The ***nmtx*** bit in the ***Int*** register is set to 1 when any of the ***eof***, ***sof***, ***iof*** bits become set to 1 after they were all previously 0, i.e., when **STXRDY** transitions from a 0 to a 1).

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***bytes[7:0]***								***sof***	***eof***							
Context-Sensitive Field									1	x	***pfe***	reserved			***port-code [1:0]***		Byte Address Offset 0x0
									0	1	***pfe***	reserved					
									0	0	***pfe***	***iof***	reserved				
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	23	22	21	20	19	18	17	16	
Field Name	reserved							***flush***	Byte Address Offset 0x2
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	hs	
Field Type	r	r	r	r	r	r	r	rw	

Table 5–8. NMTxControl (Network Management Transmit Control Register)

Bit	Name	Description
23:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16	***flush***	Flush frame. Setting this bit causes all bytes remaining in the frame to be discarded. This prevents the external CPU from reading frames it does not require. The discard process is not instantaneous; during the flush, this bit remains set at 1 and ***NMTxControl*** is write protected. This bit autoclears when the flush is complete. Writing 0 to this bit has no effect. This bit should not be set to 1 unless a frame has bytes unread, i.e., do not set this bit between frames or it may coincide with the arrival of the next frame and cause an undesired flush.
15:8	***bytes***	Byte count. This field indicates the number of bytes of frame data contained within the next 256 bytes read from ***NMData.*** If this data includes the end of frame, the count includes the 4-byte CRC. A value of 0 should be interpreted as 256 bytes. These bits are valid only if one or more of ***sof***, ***eof***, and ***iof*** is a 1.
7	***sof***	Start of frame. Indicates that the next 256 ***bytes*** to be read from ***NMData*** contain the start of a frame. If, having read the end-of-frame data from ***NMData***, ***sof*** then reads as 0, the management CPU can infer that no more frames are available.
6	***eof***	End of frame. If a 1, the end of frame is within the next 256 bytes. The ***bytes*** field indicates how many bytes should be read to complete the transmission of the frame.
5	***pfe***	Previous frame error. This bit is valid after the last byte of a frame has been read from ***NMData***. If the bit is a 1, the frame contained a CRC error or the TCI parity protection detected an error; if it is 0, the frame contained no errors. This bit remains valid until the first byte of data of the next frame is read from ***NMData***, a flush is initiated, or a reset occurs. After a flush is complete, it is 0.
4	***iof*** (***eof*** = 0, ***sof*** = 0)	Interior of frame. Valid only if ***eof*** = 0 and ***sof*** = 0. Indicates that the next 256 bytes to be read from ***NMData*** are in the interior of a frame, i.e., not the first or last 256 bytes, and are available to be read. If the external CPU is expecting to read either an interior-of-frame buffer or end-of-frame buffer, it can poll until either ***eof*** or ***iof*** is detected as a 1, then resume reading data from ***NMData.***
3:0	***portcode*** (***sof*** = 1)	Received port number. This field indicates the number of the port that received the frame. It should be interpreted only as ***portcode*** when ***sof*** is read as 1.

The following general steps are used by the CPU to get a frame from the NM port:

1) Determine that the switch has frame data available, which can be detected in three ways:

   - ***sof*** = 1 in ***NMTxControl***.
   - ***nmtx*** in the ***Int*** register is set to 1.
   - **STXRDY** = 1. **STXRDY** transition from 0 to 1 causes ***nmtx*** interrupt.

2) Each frame transmitted on the NM port is presented as a series of 256-byte buffers. The CPU must read the ***NMTxControl*** register and examine the ***sof***, ***eof***, and ***iof*** bits to determine when the next 256 bytes of data are available and what they contain. The first buffer of data has the ***sof*** bit set to 1 and contains at least 64 bytes. The following table provides an interpretation of the ***sof***, ***eof***, and ***iof*** bits.

sof	***eof***	***iof***	**Interpretation**
0	0	0	No data is currently available for transmission. This condition can occur between frames or between 256-byte buffers within a frame. The following are possible: ❑ Continue polling ***NMTXControl*** until at least one of these bits is a 1, or ❑ Wait for ***nxtx*** in ***Int*** to become 1 (and interrupt asserted if enabled) ❑ Only case where **STXRDY** will be 0
0	0	1	Intermediate buffer of a frame longer than 512 bytes. ❑ This buffer contains 256 bytes of data. The ***bytes*** field is 0 (0 is interpreted as 256 bytes). ❑ The other bits of ***NMTxControl*** are reserved and should not be relied upon to be 0.
0	1	N/A	Final buffer of frame longer than 256 bytes. ❑ This buffer of frame is longer than 256 bytes of data. The ***bytes*** field indicates how many bytes there are (0 is interpreted as 256 bytes). ❑ The other bits of ***NMTxControl*** are reserved and should not be relied upon to be 0.
1	0	N/A	First buffer of a frame longer than 256 bytes. ❑ This buffer contains 256 bytes of data. The ***bytes*** field is 0 (0 is interpreted as 256 bytes). ❑ The ***portcode*** filed indicates the source port.
1	1	N/A	The buffer contains a complete frame. ❑ This buffer contains between 64 and 256 bytes of data. The ***bytes*** field indicates how many bytes there are (0 is interpreted as 256 bytes). ❑ The ***portcode*** field indicates the source port.

3) The CPU then reads the bytes in the buffer by doing the appropriate number of reads from ***NMData***, one read per byte. ***NMData*** should not be read unless at least one of the ***sof***, ***eof***, or ***iof*** bits was found to be a 1; otherwise, unpredictable behavior can occur.

4) Steps 2 and 3 are repeated until a read of ***NMTxControl*** returns a value of ***eof*** = 1, indicating that this is the final buffer of the frame and the ***bytes*** field in the most significant byte indicates how many bytes remain to be read.

   At any time during steps 3 and 4, the CPU can choose to discard the remainder of the frame. This is achieved by setting the ***flush*** bit to 1 (***NMTxControl*** will then be write protected until the flush has completed). This initiates an automatic discard mechanism that signals its completion by returning ***flush*** to 0, re-enabling writes to ***NMTxControl***. The ***sof***, ***eof***, ***iof***, and ***pfe*** bits and the ***portcode*** field are undefined during a flush and become valid again only after a flush is complete. No reads of ***NMData*** should be performed after a flush has been initiated; otherwise unpredictable behavior occurs. Also, a flush should not be initiated if there is no frame data remaining to be read, because this could risk initiating a flush at the same moment that the data for a new frame becomes available.

5) A further read of the least significant byte of ***NMTxControl*** then indicates if the frame that has just been read contained an error and if there is another frame waiting for the CPU.

   - If ***pfe*** = 1, the frame that has just been read contained a CRC error or a header parity-protection error.
   - If ***sof*** = 1, another frame is queued and steps 2 through 5 are repeated until step 5 reads the ***sof*** bit as a 0, indicating that the NM port transmit queue is empty. It then returns to step 1 for the detection of future frames.

5.3.5 TSReadFrame – Flowchart and Sample Code

Once the CPU has determined that a frame is available, it can read the frame from the NM port. This section presents an example of a simple blocking (does not return until completed) implementation of such a function, TSReadFrame.

The function prototype for TSReadFrame is described in the following paragraphs.

5.3.5.1 TSReadFrame – Read a Frame From the NM Port

```
typedef unsigned int    WORD;
typedef unsigned char   BYTE;
typedef unsigned char   BOOL;
```

Syntax

int TSReadFrame (BYTE *FrameData, WORD *FrameBytes, BYTE *PortCode)

Parameters

FrameData	Pointer to buffer where MAC frame data is stored (must be at least 1536 bytes in length to ensure buffer is not overrun when reading large frames)
FrameBytes	Pointer to variable where number of bytes (including CRC) is written.
PortCode	Pointer to variable where port routing code is written.

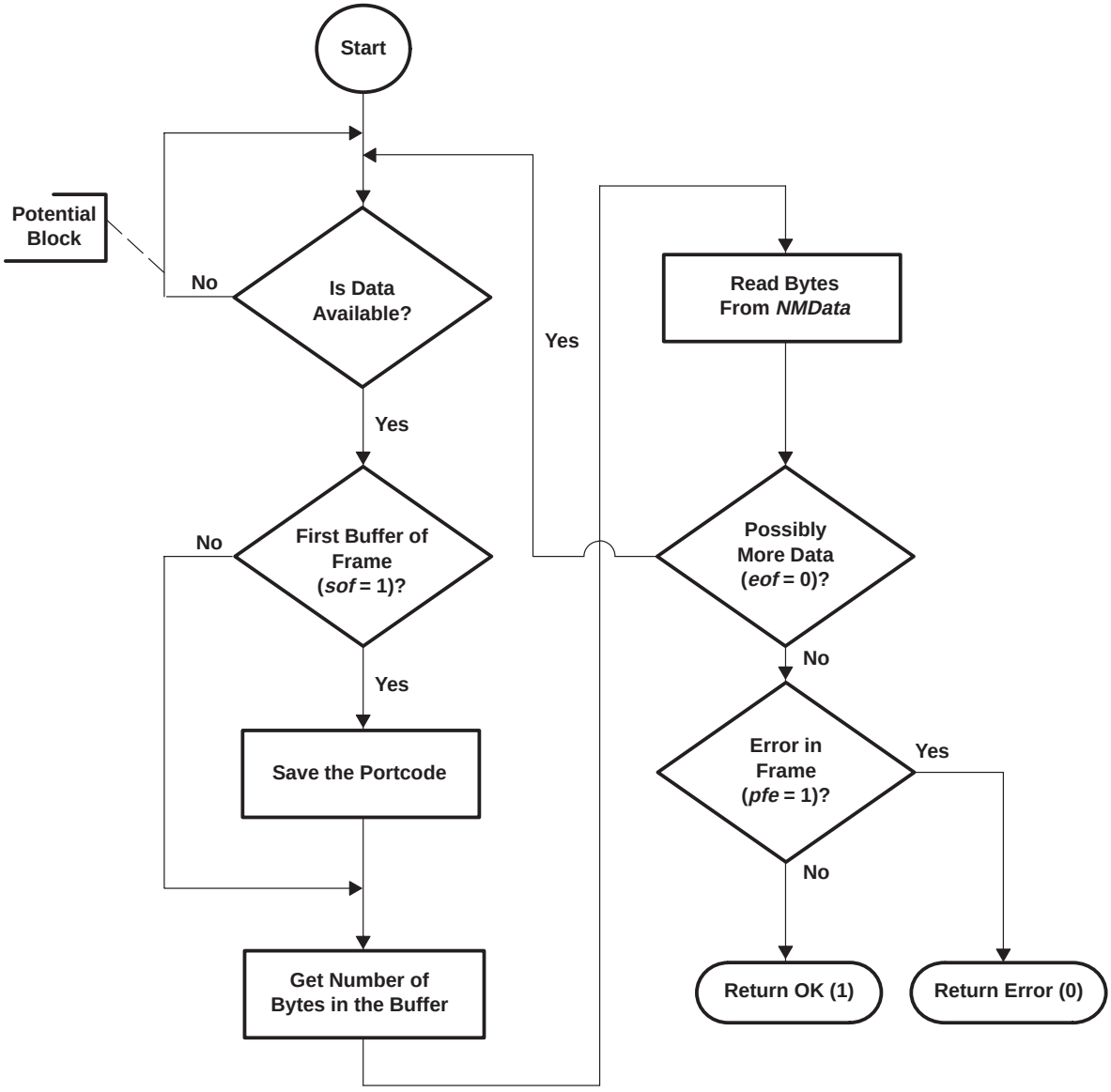
Return Value

0 on error, 1 if frame is read from NM port OK

5.3.5.2 TSReadFrame – Flowchart

The following is the flowchart for the TSReadFrame example function. The next section contains a code listing for this function.

Figure 5–12. TSReadFrame Flowchart



5.3.5.3 TSReadFrame – Code Listing

```
typedef unsigned intWORD;
typedef unsigned char   BYTE;
typedef unsigned char   BOOL;

//TNETX4020 DIO register locations
#define NMTxControl    0x081C
#define NMData         0x0820

// NMTxControl register bit fields
#define PORTCODE_MASK  0x001F
#define IOF            0x0010
#define PFE            0x0020
#define EOF            0x0040
#define SOF            0x0080

#define OK          1
#define ERROR       0

int TSReadFrame( FrameData, FrameBytes, PortCode )
   BYTE   *FrameData;      // Pointer to Frame Data
   WORD   *FrameBytes;     // Total # of bytes in the frame
   BYTE   *PortCode;   // Port routing code, or 255 for IALE routing
{
   BYTE   data, BytesInBuffer;
   WORD   BytesRead;

   *FrameBytes = 0;
   //-------------------------------------------------------------------------
   // The following do loop will process one buffer (at most 256 bytes) of
   // frame data per loop. It will be repeated until all frame data buffers
   // have been read.
   //-------------------------------------------------------------------------
   do
   {
      //----------------------------------------------------------------------
      // Wait for at least one buffer of data to be available from TNETX4020.
      // This is indicated by either SOF, IOF, and/or EOF set to 1.
      //----------------------------------------------------------------------
      do
         DioRdByte( NMTxControl, &data );
      while( (data & (SOF | IOF | EOF)) == 0 );
      //----------------------------------------------------------------------
      // Save the port code if this is the first buffer of the frame.
      //----------------------------------------------------------------------
      if( data & SOF )
         *PortCode = data & PORTCODE_MASK;
      //----------------------------------------------------------------------
      // Get the # of bytes in this buffer. Remember that a value of 0 means
      // 256 bytes.
      //----------------------------------------------------------------------
      DioRdByte( NMTxControl+1, &BytesInBuffer );
      if( BytesInBuffer == 0 ) BytesInBuffer = 256;
      //----------------------------------------------------------------------
      // Read the buffer data from the NM port.
      //----------------------------------------------------------------------
      *FrameBytes += BytesInBuffer;
      while( BytesInBuffer-- )
         DioRdByte( NMData, FrameData++ );
   } while( (data & EOF) == 0 );    // EOF==1 means no more data
```

```
    //-------------------------------------------------------------------------
    // Now that we've read the entire frame, the frame error indicator is
    // valid. Let's check if the frame just read contained a CRC error or
    // the TCI parity protection detected an error and return the result.
    //-------------------------------------------------------------------------
    DioRdByte( NMTxControl, &data );
    if( data & PFE )
        return ERROR;
    return OK;
}
```

5.4 Internal Address-Lookup Engine (IALE) Management

The external CPU has complete control over the IALE. The TNETX4020 is configured so the IALE is maintained and managed by the switch, i.e., learn from the wire and using an aging-out algorithm, or the external CPU can manage and maintain the IALE. (See Chapter 4 for detailed information on how to configure the IALE.) The external CPU can, by way of the DIO interface, perform the following operations on the IALE:

- Add/update addresses
- Delete addresses
- Perform a search of the IALE using programmable search criteria

It is expected that most systems will use the external CPU in conjunction with the on-chip features to maintain the IALE. The TNETX4020 automatically adds/updates unicast addresses to the IALE from the wire, as well as deletes unicast addresses using one of two aging algorithms. To complement these on-chip features, the external CPU can add multicast addresses and manage and maintain the IALE by using interrupt service routines.

5.4.1 Add Operation

To add a new address or to update an existing address in the IALE, the software must provide the address of the node in native Ethernet format, as well as the VLAN index and port information. All the necessary address information must be written to the ***Addnode***, ***AddVLAN***, and ***AddPort*** registers by the software before enabling the add operation.

AddNode (Address Register) @ 0x0458–0x045D

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table 5–9.AddNode (Address Register)

Bit	Name	Description
47:0	***address***	Add node address. The unicast or multicast address in this register is added to the address-lookup table when ***add*** in ***AddDelControl*** is set to 1. The node address in ***AddNode*** is kept in Ethernet's native format. When the *Add* state machine is operating, this field is locked and writes to it have no effect.

AddVLAN (Add Address VLAN Index Register) @ 0x045F

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table 5–10. AddVLAN (Add Address VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	Add VLAN index. This is used with the ***AddNode*** and **AddPort** registers to add/edit a node's VLAN assignment. When the Add state machine is operating, this field is locked and writes to it have no effect.

The bit fields in the ***AddPort*** register are coded differently, depending on whether the address stored in ***AddNode*** is unicast or multicast.

AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Unicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved										***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	***secure***	***locked***	***cuplnk***	***new***	reserved											
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	

Table 5–11. AddPort (Add Routing Code Register) – Unicast Format (Determined by Bit 40 of Address)

Bit	Name	Description
31	***nblck***	Not-blocked flag. This bit sets the ***RxUniBlockPorts*** override function. ❑ ***nblck*** = 1. ***RxUniBlockPorts*** are not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxUniBlockPorts*** bit for the source port is not set. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
30	***secure***	Secured-address flag. This bit determines the security level for the address contained in ***AddNode***. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
29	***locked***	Locked address flag. This bit determines the lock status for the address in ***AddNode***. Any frames received from a locked-source address are discarded. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
28	***cuplnk***	Copy-frames-to-uplink flag. This bit determines the Copy Uplink status for the address contained in ***AddNode***. Addresses tagged with this bit add the port specified in the ***UplinkPort*** register to the routing code. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.

Table 5–11. AddPort (Add Routing Code Register) – Unicast Format (Determined by Bit 40 of Address)(Continued)

Bit	Name	Description
27	***new***	New-address flag. This bit determines the new status for the address contained in ***AddNode***. Having set ***new*** to 1, this address easily can be found by a Find command, which has ***new*** in ***FindControl*** = 1. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
26:24	reserved	Reserved. These bits are used in multicast format.
23:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Current extended port for node. This field determines the destination port for the unicast address shown in ***AddNode***. If a nonexistent port is specified, no address record is entered when the Add is performed. (The Add will complete in the normal manner but no record will exist.) When an Add is in progress, this bit is locked against modification, and writes to it have no effect.

AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Multicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	reserved	***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	

Table 5–12. AddPort (Add Routing Code Register) – Multicast Format (Determined by Bit 40 of Address)

Bit	Name	Description
31	***nblck***	Not blocked flag. This bit sets the ***RxMultiBlockPorts*** override function. ❑ ***nblck*** = 1. ***RxMultiBlockPorts*** is not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxMultiBlockPorts*** bit for the source port is not set. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
30	reserved	Reserved. This bit is used in unicast format.
29:24	***xroute code***	Extended route code for multicast. This field determines the route code used to generate the preframe tag added to frames routed to port 1 when ***pretag*** = 1 in ***Port1Control***. This field is ignored if pretagging is disabled or both bit 0 and bit 1 of ***portvector*** are 0. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
23:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:3	reserved	Reserved. These bits are used in unicast format.
2:0	***portvector***	Port bit vector for multicast. This field determines the port bit vector for the multicast address contained in ***AddNode***. The bit values in this field correspond one to one with the port assignment. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.

When the software has supplied the required information in the previously described registers, the add/edit of a node is initiated by setting the ***add*** bit in the ***AddDelControl*** register to 1. The ***add*** bit remains set to 1 until completion of the add operation. Upon the completion of the add operation, the ***add*** bit is cleared by the switch. The external CPU may poll this bit to determine when the operation has completed because the switch does not generate an interrupt on a new address added by the external CPU. During the add operation, the ***AddNode***, ***AddVLAN***, and ***AddPort*** registers are locked, and writes to them have no effect.

AddDelControl (Add/Delete Control Register) @ 0x045E

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved				***delp***	***delv***	***add***	***del***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	hs	hs	hs	hs	
Field Type	r	r	r	r	rw	rw	rw	rw	

Table 5–13. AddDelControl (Add/Delete Control Register)

Bit	Name	Description
7:4	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
3	***delp***	Delete addresses on port. When set to 1, all unicast address records whose port assignment matches ***DelPort*** are deleted unless they are marked as secure or locked. Multicast addresses are not deleted. This bit remains a 1 until the delete process is complete, when it self-clears. During this period ***delp***, ***delv***, and ***del*** are write protected. This bit can be asserted at the same time as ***delv***, but not if ***delv*** is already set. This bit can be used with ***delv*** to delete all nonlocked, nonsecure, unicast address records on a particular port with a particular VLAN.
2	***delv***	Delete addresses on VLAN. When set to 1, all unicast address records whose VLAN assignment matches ***DelVLAN*** are deleted unless they are marked as secure or locked. Multicast addresses are not deleted. This bit remains a 1 until the delete process is complete, when it self-clears. During this period ***delp***, ***delv***, and ***del*** are write protected. This bit can be asserted at the same time as ***delp***, but not if ***delp*** is already set. This bit can be used with ***delp*** to delete all nonlocked, nonsecure, unicast address records on a particular port with a particular VLAN.
1	***add***	Add address. When set to 1, the information in ***AddPort***, ***AddNode***, and ***AddVLAN*** is used to add or edit an entry in the address records.This bit remains a 1 until the add process is complete, when it self-clears. During this period, ***add*** is write protected.

Table 5–13. AddDelControl (Add/Delete Control Register) (Continued)

Bit	Name	Description
0	***del***	Delete address. When set to 1, the information in ***DelNode*** and ***DelVLAN*** is used to delete an entry from address records. Any address records can be deleted this way, including multicast addresses and secure and/or locked unicast addresses. This bit remains a 1 until the delete process is complete, when it self-clears. During this period, ***delp***, ***delv***, and ***del*** are write protected. This bit cannot be set to 1 if ***delp*** or ***delv*** also is being set to 1. If such an attempt is made, none of these bits will be set to 1 and no delete will be performed.

5.4.1.1 Pseudocode

The following pseudocode describes an add operation of a unicast address.

```
// The external CPU is to perform an add operation with the following criteria:
// Node = 00.11.22.33.44.55 (decimal format)
// VLAN = 0x01, the default value in VLAN0QID
// Port = 0x04
//        nblk = 0
//        secure = 0
//        locked = 0
//        cuplnk = 0
//        new = 0
//

{

   *DIO_addr_lo = 0x5F;
   *DIO_addr_hi = 0x04;       // DIO address: AddVLAN register

   // ADDVLAN REGISTER : Node belongs to VLAN 0x01

   *DIO_data_inc = 0x00;      // AddVLAN = 0x01, VLAN0QID defaults to 0x1

   // DIO address points to AddPort register

   *DIO_data_inc = 0x04;      // AddPort = 0x04

   // Set DIO address to 0x0458 – start of the AddNode register

   *DIO_addr_lo = 0x58;
   *DIO_addr_hi = 0x04;       // DIO address: AddNode register

   // AddNode register = 00.11.22.33.44.55 (MAC address)

   *DIO_data_inc = 0x00;      // AddNode[47:40] = 0x00
   *DIO_data_inc = 0x11;      // AddNode[39:32] = 0x11
   *DIO_data_inc = 0x22;      // AddNode[31:24] = 0x22
   *DIO_data_inc = 0x33;      // AddNode[23:16] = 0x33
   *DIO_data_inc = 0x44;      // AddNode[15:8] = 0x44
   *DIO_data_inc = 0x55;      // AddNode[7:0] = 0x55

   // DIO address will now point to AddDelControl register (0x045E)

   *DIO_data = 0x02;          // add =1, Initiates add operation

   while (add)                // Polling AddDelControl to determine when add
   {   }                      // operation has completed.

}
```

5.4.2 Delete Operation

The TNETX4020 supports four different address-delete operations. The delete operation to be performed is specified by the ***del***, ***delp***, and ***delv*** bits in the ***AddDelControl*** register. Only one of the four delete operations can be attempted at one time. Illegal delete operations, i.e., bit combinations not supported, are ignored and the ***del***, ***delp***, and ***delv*** bits remain 0.

1) Delete any specific address indicated by ***DelNode*** and ***DelVLAN*** registers. The contents of ***DelPort*** register are ignored. This operation is initiated by setting the ***del*** bit to 1.

2) Delete all unicast† addresses on a particular port specified by ***DelPort*** register. The contents of ***DelNode*** and ***DelVLAN*** registers are ignored. This operation is initiated by setting the ***delp*** bit to 1. Multicast addresses assigned to a particular port cannot be deleted by this operation.

3) Delete all unicast† addresses on a particular VLAN specified by the ***DelVLAN*** register. The contents of ***DelNode*** and ***DelPort*** registers are ignored. This operation is initiated by setting the ***delv*** bit to 1. Multicast addresses assigned in a VLAN cannot be deleted by this operation.

4) Delete all unicast† addresses on a particular VLAN that are specified by the ***DelVLAN*** register and assigned to a particular port specified by the ***DelPort*** register. The contents of the ***DelNode*** register are ignored. This operation is initiated by setting both the ***delv*** and ***delp*** bits to 1 in a single DIO access.

† Nonlocked, nonsecure unicast addresses

AddDelControl (Add/Delete Control Register) @ 0x045E

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved				***delp***	***delv***	***add***	***del***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	hs	hs	hs	hs	
Field Type	r	r	r	r	rw	rw	rw	rw	

Table 5–14. AddDelControl (Add/Delete Control Register)

Bit	Name	Description
7:4	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
3	***delp***	Delete addresses on port. When set to 1, all unicast address records whose port assignment matches ***DelPort*** are deleted unless they are marked as secure or locked. Multicast addresses are not deleted. This bit remains a 1 until the delete process is complete, when it self-clears. During this period ***delp***, ***delv***, and ***del*** are write protected. This bit can be asserted at the same time as ***delv***, but not if ***delv*** is already set. This bit can be used with ***delv*** to delete all nonlocked, nonsecure, unicast address records on a particular port with a particular VLAN.
2	***delv***	Delete addresses on VLAN. When set to 1, all unicast address records whose VLAN assignment matches ***DelVLAN*** are deleted unless they are marked as secure or locked. Multicast addresses are not deleted. This bit remains a 1 until the delete process is complete, when it self-clears. During this period ***delp***, ***delv***, and ***del*** are write protected. This bit can be asserted at the same time as ***delp***, but not if ***delp*** is already set. This bit can be used with ***delp*** to delete all nonlocked, nonsecure, unicast address records on a particular port with a particular VLAN.
1	***add***	Add address. When set to 1, the information in ***AddPort***, ***AddNode***, and ***AddVLAN*** is used to add or edit an entry in the address records.This bit remains a 1 until the add process is complete, when it self-clears. During this period, ***add*** is write protected.
0	***del***	Delete address. When set to 1, the information in ***DelNode*** and ***DelVLAN*** is used to delete an entry from address records. Any address records can be deleted this way, including multicast addresses and secure and/or locked unicast addresses. This bit remains a 1 until the delete process is complete, when it self-clears. During this period, ***delp***, ***delv***, and ***del*** are write protected. This bit cannot be set to 1 if ***delp*** or ***delv*** also is being set to 1. If such an attempt is made, none of these bits will be set to 1 and no delete will be performed.

The external CPU must write the necessary address information to the ***DelNode***, ***DelVLAN***, and ***DelPort*** registers, as specified for the specific delete operation to be performed, before enabling the delete operation.

DelNode (Delete Address Register) @ 0x046C–0x0471

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table 5–15. DelNode (Delete Address Register)

Bit	Name	Description
47:0	***address***	Delete node address. The unicast or multicast address in this register is deleted from the address-lookup table when the ***del*** bit in ***AddDelControl*** is set to 1. The node address in ***DelNode*** is kept in Ethernet's native format. When the *Delete* state machine is operating, this field is locked, and writes to it have no effect.

DelPort (Delete Port Register) @ 0x0472

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table 5–16. DelPort (Delete Port Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Extended delete port. When ***delp*** in ***AddDelControl*** is set, all addresses associated with this port are deleted. If ***delv*** also is a 1, the address records are further qualified by the VLAN specified in ***DELVLAN***. Once set, ***delp*** is locked until the delete completes, and writes to it have no effect.

DelVLAN (Delete VLAN Index Register) @ 0x0473

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table 5–17. DelVLAN (Delete VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	Delete VLAN index. Index into ***VLANxQID*** table. ❑ If the ***del*** bit in ***AddDelControl*** is set, this field is used with the ***DelNode*** and ***DelPort*** registers to delete a node from the address-lookup table. ❑ When the ***del*** and ***delv*** bits in ***AddDelControl*** are both set, all addresses associated with this VLAN index are deleted. In this case, the ***DelNode*** and ***DelPort*** registers are don't care. When the Delete state machine is performing a Delete or Delete VLAN operation, this field is locked and writes to it have no effect.

When the software has supplied the required information, the specific delete operation is determined and initiated by configuring the ***del***, ***delp***, and ***delv*** bits in the ***AddDelControl*** register (see the following table). The ***del, delp,*** and ***delv*** bits associated with a particular delete operation remain set to 1 until completion of the operation. Upon completion of the operation, the bits are cleared by the switch. The external CPU must poll these bits to determine when the operation has completed because the switch does not generate an interrupt on an address deletion by the external CPU. During the delete operation, the ***DelNode***, ***DelVLAN***, and ***DelPort*** registers are locked, and writes to them have no effect.

The following table summarizes each of the possible delete operations. The ***AddDelControl*** field of the table indicates the value with which the register must be configured to initiate the delete operation. The Register(s) field determines which register(s) must be used in conjunction with the operation. The delete registers not listed for a specific operation are don't cares for that particular operation.

AddDelControl	Register(s)	Description
0x01	DelNode DelVLAN	Delete a specific address specified by the ***DelNode*** and ***DelVLAN*** registers.
0x04	DelVLAN	Delete all unicast† addresses associated with the VLAN specified in ***DelVLAN***.
0x08	DelPort	Delete all unicast† addresses associated with the port specified in ***DelPort***.
0x0C	DelVLAN DelPort	Delete all unicast† addresses on a particular VLAN (specified by ***DelVLAN***) assigned to a particular port (specified by ***DelPort***)

† Nonlocked, nonsecure unicast addresses

5.4.2.1 Pseudocode

The delete operations supported by the TNETX4020 are almost identical. The only difference between the delete operations is the registers that are used as criteria and the bits that are set in ***AddDelControl*** register. The following pseudocode describes the delete of a specific address specified by its MAC address and VLAN.

```
// The external CPU is to perform a delete operation with the following criteria:
// Node = 00.11.22.33.44.55 (decimal format)
// VLAN = 0x01, the default value of VLAN0QID register

//
// The switch will delete the node with the MAC address of 00.11.22.33.44.55
// belonging to VLAN 0x01. The del bit will be cleared upon the completion of
// the operation.

{

   *DIO_addr_lo = 0x6C;

   *DIO_addr_hi = 0x04;       // DIO address: DelNode register

   // DELNODE REGISTER – MAC address of node to be deleted 00.11.22.33.44.55
   *DIO_data_inc = 0x00;      // DelNode[47:40] = 0x00

   *DIO_data_inc = 0x11;      // DelNode[39:32] = 0x11
   *DIO_data_inc = 0x22;      // DelNode[31:24] = 0x22
   *DIO_data_inc = 0x33;      // DelNode[23:16] = 0x33
   *DIO_data_inc = 0x44;      // DelNode[15:8] = 0x44
   *DIO_data_inc = 0x55;      // DelNode[7:0] = 0x55

   // Set DIO address to 0x0473 – start of the DelVLAN register

   *DIO_addr_lo = 0x73;
   *DIO_addr_hi = 0x04;       // DIO address: DelVLAN register

   // DELVLAN REGISTER – VLAN to be deleted 0x01

   *DIO_data = 0x00;          // DelVLAN = 0x01, default VLAN in VLAN0QID register

   // Set DIO address to 0x045E – start of the AddDelControl register

   *DIO_addr_lo = 0x5E;
   *DIO_addr_hi = 0x04;       // DIO address: AddDelControl register

   // ADDDELCONTROL REGISTER: Initiate delete by setting del bit to 1

   *DIO_data = 0x01;          // del =1, Initiates delete operation

   while (del)                // Polling AddDelControl to determine when delete
   {   }                      // operation has completed.

}
```

5.4.3 Searching the IALE

The external CPU can search the IALE using the ***FindControl*** register in conjunction with the ***FindNode***, ***FindVLAN***, and ***FindPort*** registers. The ***FindPort*** format depends on the address type being sought: unicast or multicast. The TNETX4020 supports three different search/find operations of the IALE records: *Lookup*, *Find First*, and *Find Next* (see Table 5–18) using the ***FindControl*** register. The operation to be performed is selected by setting the ***new***, ***node***, ***port***, ***vlan***, ***first***, and ***find*** bits in the ***FindControl*** register.

FindControl (Search Control Register) @ 0x0446

Bit	7	6	5	4	3	2	1	0	
Field Name	***found***	***new***	***node***	***port***	***vlan***	***first***	reserved	***find***	Byte Address Offset
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	–	hs	
Field Type	r	rw	rw	rw	rw	rw	r	rw	

Table 5–18. FindControl (Search Control Register)

Bit	Name	Description
7	***found***	Address found. If the find operation initiated by setting ***find*** is successful, this bit is set upon completion. Any data returned in ***FindVLAN***, ***FindNode***, or ***FindPort*** is valid only when this bit is a 1.
6	***new***	New address finds. When asserted, the address-lookup table is scanned for the first address that is marked as new.
5	***node***	Find by address. When asserted, the address indicated by ***address*** in ***FindNode*** is included in the match criteria.
4	***port***	Find by port. When asserted, the port indicated by ***xportcode*** in ***FindPort*** is included in the match criteria. Searches with this bit set are restricted to unicast addresses.
3	***vlan***	Find by VLAN index. When asserted, the port indicated by ***vlanindex*** in ***FindVLAN*** is included in the match criteria.
2	***first***	Lookup first address. Determines the point in the address-lookup table from which the search begins. ❑ ***first*** = 1. The search starts from the first location in the address-lookup table. ❑ ***first*** = 0. The search starts from the location in the address-lookup table immediately following that at which the previous search completed, i.e., the address currently in ***FindNode***.
1	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
0	***find***	Address lookup. When asserted, the find operation is initiated. The type of search is determined by the value of ***node***, ***vlan***, ***port***, ***new***, and ***first***. When the find is active, this register is locked and writes to it have no effect. Upon completion of the search, this bit self-clears and the register can be written again.

FindNode (Address Register) @ 0x0440–0x0445D

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[16:23]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table 5–19. FindNode (Address Register)

Bit	Name	Description
47:0	***address***	Find node address. Used to exchange addresses between the address lookup *Find* state machine and the management CPU. The node address in ***FindNode*** is kept in Ethernet's native format. When the *Find* state machine is operating, this field is locked, and writes to it have no effect. This field can be the results of a Find operation or the starting point for the next operation, depending on the contents of the ***FindControl*** register.

FindVLAN (Find Address VLAN Index Register) @ 0x0447

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table 5–20. AddVLAN (Add Address VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved
5:0	***vlanindex***	Find VLAN index. This field is used in two ways, depending on the type of search: ❑ ***vlan*** = 0 in ***FindControl***. This field returns the VLAN index for the node matching the specified search criteria. ❑ ***vlan*** = 1 in ***FindControl***. The VLAN index to use as part of the matching criteria should be written to this field before setting ***find*** in ***FindControl***. When a find is active, this register is locked, and writes to it have no effect.

The bit fields in the ***FindPort*** register are coded differently, depending on whether the address stored in ***FindNode*** is unicast or multicast.

FindPort (Search Routing Code Register) @ 0x0448–0x044B – Unicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***nodeage[7:0]***								reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	***secure***	***locked***	***cuplnk***	***new***	reserved			***nodeage[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 5–21. FindPort (Search Routing Code Register) – Unicast Format (Determined by Bit 40 of Address)

Bit	Name	Description
31	***nblck***	Not-blocked flag. This bit sets the ***RxUniBlockPorts*** override function. ❑ ***nblck*** = 1. ***RxUniBlockPorts*** are not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxUniBlockPorts*** bit for the source port is not set. When a Find is in progress, this bit is locked against modification, and writes to it have no effect.
30	***secure***	Secured address indication. This bit shows the security level for the address in ***FindNode.*** Secure addresses are not aged out and cannot move between ports. If a secured address attempts to move from one port to another, a security violation interrupt is given to the external CPU. The address record of the node is not altered if this occurs, i.e., it is not locked and still can transmit successfully on the port in the address record.
29	***locked***	Locked address flag. This bit shows the lock status for the address in ***FindNode***. Any frames received with a locked-source address in either the source or destination field are discarded. When a Find is in progress, this bit is locked against modification, and writes to it have no effect.

Table 5–21. FindPort (Add Routing Code Register) – Unicast Format (Determined by Bit 40 of Address)(Continued)

Bit	Name	Description
28	***cuplnk***	Copy-frames-to-uplink flag. This bit shows the Copy Uplink status for the address contained in ***FindNode***. Addresses tagged with this bit add the port specified in the ***UplinkPort*** register to the routing code. When a Find is in progress, this bit is locked against modification, and writes to it have no effect.
27	***new***	New-address flag. This bit shows the new status for the address contained in ***FindNode***. Having set ***new*** to 1, this address easily can be found by a *Find* command, which has ***new*** in ***FindControl*** = 1. When a Find is in progress, this bit is locked against modification. Writes to it have no effect.
26:24	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
23:8	***nodeage***	Address age stamp. Contains the age time stamp of the address contained in ***FindNode***.
7:6	reserved	Reserved. Writes to these bits have no effect, although they can be written.
5:0	***xportcode***	Current extended port for node. This field is used in two ways, depending on the type of search: ❑ ***port*** = 0 in ***FindControl***. This field returns the port for the node matching the specified search criteria. ❑ ***port*** = 1 in ***FindControl***. The port to use as part of the matching criteria should be written to this field before setting ***find*** in ***FindControl***. During searches, this field is locked, and writes to it have no effect.

FindPort (Search Routing Code Register) @ 0x0448–0x044B – Multicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	reserved	***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table 5–22. FindPort (Add Routing Code Register) – Multicast Format (Determined by Bit 40 of Address)

Bit	Name	Description
31	***nblck***	Not blocked flag. This bit sets the ***RxMultiBlockPorts*** override function. ❑ ***nblck*** = 1. ***RxMultiBlockPorts*** is not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxMultiBlockPorts*** bit for the source port is not set. When a Find is in progress, this bit is locked against modification. Writes to it have no effect.
30	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
29:24	***xroute code***	Extended route code for multicast. This field shows the route code used to generate the preframe tag added to frames routed to port 1 when ***pretag*** = 1 in ***Port1Control***. It is ignored if ***pretag*** or bit 2 of ***portvector*** is 0. When a Find is in progress, this bit is locked against modification.Writes to it have no effect.
23:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:3	reserved	Reserved. These bits are used in unicast format.
2:0	***portvector***	Port bit vector for multicast. This field determines the port bit vector for the multicast address contained in ***FindNode***. The bit values in this field correspond one to one with the port assignment. When a Find is in progress, this bit is locked against modification, and writes to it have no effect.

Lookup

A *Lookup* operation is performed when the ***node*** and ***vlan*** bits are set to 1 (***first*** bit is irrelevant). The *Lookup* operation will find the VLAN/node combination in the address-lookup table (if it exists) specified by the ***FindNode*** and ***FindVLAN*** registers. The ***new*** and ***port*** bits can be used as search criteria for the VLAN/Node combination.

Find First

The *Find-First* operation does a systematic search through the IALE's records, starting at its first location. The operation is identified as those with the ***first*** bit set to 1. The ***node***, ***vlan***, ***port***, and ***new*** bits can be used as search criteria.

Find Next

The *Find-Next* operation searches systematically through the address-lookup table, starting where a previous *Find-First* or *Find-Next* operation successfully finds a match. The *Find-Next* operation can be used only after a previous successful *Find-First* or *Find-Next* operation, because the starting point for the search is defined. The operation is identified as those with the ***first*** bit set to zero. The ***node***, ***vlan***, ***port***, and ***new*** bits can be used as the search criteria.

The following criteria can be used for the search operations described previously:

node

Searches the address-lookup table for an entry that contains the Ethernet address currently contained in the ***FindNode*** register.

vlan

Searches the address-lookup table for an entry that contains the VLAN currently contained in the ***FindVLAN*** register.

port

Searches the address-lookup table for unicast records that contain the port currently contained in the ***FindPort*** register. This criterion cannot be used to search for multicast records.

new

Searches the address-lookup table for unicast records that have their new bit set to 1. Once a new record has been found, the new bit in the record is cleared automatically. This ensures that the record is found as new only once. The new bit in the IALE's record is set to 1 when a record (address) is added or modified from the wire, as well as when the external CPU adds a record with the new bit set in the ***AddPort*** register.

first

When this bit is set to 1, the search starts from the first record in the address-lookup table. When ***first*** is set to 0, the search starts from the previous successful *Find* operation.

All possible combinations of *Lookup*, *Find-First,* and *Find-Next* operations are listed in Table 5–23.

These combinations can be classified as three types of operation: *Lookup*, *Find First*, and *Find Next,* as indicated in the Search Type and Qualifications columns.

Table 5–23. Supported Find Operations

Search Type	Qualifications	*new*	*node*	*port*	*vlan*	*first*	*find*
Lookup	(VLAN and Node)	0	1	0	1	x	1
	(VLAN and Node) and Port	0	1	1	1	x	1
	(VLAN and Node) and New	1	1	0	1	x	1
	(VLAN and Node) and New and Port	1	1	1	1	x	1
Find First	Record (of any type)	0	0	0	0	1	1
	VLAN	0	0	0	1	1	1
	Port	0	0	1	0	1	1
	Node	0	1	0	0	1	1
	VLAN and Port	0	0	1	1	1	1
	Port and Node	0	1	1	0	1	1
	New	1	0	0	0	1	1
	New and VLAN	1	0	0	1	1	1
	New and Port	1	0	1	0	1	1
	New and Node	1	1	0	0	1	1
	New and VLAN and Port	1	0	1	1	1	1
	New and Port and Node	1	1	1	0	1	1
Find Next	Record (of any type)	0	0	0	0	0	1
	VLAN	0	0	0	1	0	1
	Port	0	0	1	0	0	1
	Node	0	1	0	0	0	1
	VLAN and Port	0	0	1	1	0	1
	Port and Node	0	1	1	0	0	1
	New	1	0	0	0	0	1
	New and VLAN	1	0	0	1	0	1
	New and Port	1	0	1	0	0	1
	New and Node	1	1	0	0	0	1
	New and VLAN and Port	1	0	1	1	0	1
	New and Port and Node	1	1	1	0	0	1

Note:

To find all the records that match a certain criterion, the external CPU must first do a *Find-First* operation. The *Find-First* operation, if successful, must be followed by *Find-Next* operations. The external CPU must keep performing *Find-Next* operations until unsuccessful, i.e., the ***found*** bit is returned as 0.

5.4.3.1 Pseudocode

Search operations supported by the TNETX4020 are almost identical. The only difference is the bits that are set in the ***FindControl*** register (a *Lookup* operation from a *Find-First* or *Find-Next* operation). Hence, the following pseudocode describes only one of the three possible operations. In this case, a *Lookup* is performed.

```
// The external CPU is to perform a lookup operation with the following criteria:
// Node = 00.11.22.33.44.55 (decimal format)
// VLAN = 0x01, the value in VLAN0QID
// Port = 0x05

//
// The switch will perform a lookup on the MAC address, VLAN index and Port given
// above. If this "node" is found in the IALE, the found bit in FindControl will
// be set to 1 after the completion of the operation.

{

   *DIO_addr_lo = 0x47;
   *DIO_addr_hi = 0x04;                // DIO address: FindVLAN register

   // FINDVLAN REGISTER : Search for VLAN: 0x01

   *DIO_data_inc = 0x00;               // FindVLAN = 0x01 = VLAN0QID

   // FINDPORT REGISTER: Search for Port: 0x01

   *DIO_data_inc = 0x01;               // FindPort = 0x01

   // Set DIO address to 0x0440 – start of the FindNode register

   *DIO_addr_lo = 0x40;
   *DIO_addr_hi = 0x04;                // DIO address: FindNode register

   // FINDNODE REGISTER: Search for MAC address 00.11.22.33.44.55

   *DIO_data_inc = 0x00;               // FindNode[47:40] = 0x00
   *DIO_data_inc = 0x11;               // FindNode[39:32] = 0x11
   *DIO_data_inc = 0x22;               // FindNode[31:24] = 0x22
   *DIO_data_inc = 0x33;               // FindNode[23:16] = 0x33
   *DIO_data_inc = 0x44;               // FindNode[15:8] = 0x44
   *DIO_data_inc = 0x55;               // FindNode[7:0] = 0x55

   // The DIO address now points to FindControl (0x0446)

   *DIO_data = 0x39;                   // find =1 , vlan=1, port = 1, node = 1.
                                       // Initiates lookup operation

   while (find & vlan & port & node)   // Polling FindControl to determine
   {   }                               // when lookup operation has completed.

   if (found)                          // found = 1 indicates a match
      successful;
   else                                // found = 0 indicates no match
      not successful;

}
```

Chapter 6

Servicing Interrupts

The TNETX4020 asserts the **SINT** pin high to indicate to the external CPU that an interrupt has occurred. An interrupt is generated when a bit is set in the ***Int*** register and the corresponding bit in the ***IntEnable*** register is set. When an interrupt occurs, the external CPU may read the ***Int*** register to determine the cause of the interrupt. When the cause of the interrupt is determined, the external CPU should process the appropriate interrupt handler routine.

The **SINT** pin remains high until the bit(s) that caused the interrupt has been cleared, i.e., the bit previously set to 1 is set to 0. A bit in the ***Int*** register is cleared when a 1 is written to that bit location. Writing a 0 to any bit location in the ***Int*** register has no effect. A bit set due to an interrupt has precedence over a bit cleared by the external CPU if both occurrences are coincidental.

Topic **Page**

6.1 Interrupts 6-5

Int (Interrupt Register) @ 0x0804–0x0806

The ***Int*** register, in conjunction with the ***IntEnable*** register, provides interrupts to the external CPU. When the **SINT** pin is asserted high, the ***INT*** register gives the reason for the interrupt. Specific interrupts can be masked by clearing the appropriate bit in ***IntEnable*** register.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***new***	***newm***	***chng***	***chngm***	***secvio***	reserved	***age***	***agem***	***int***	***fndcplt***	***unkvlan***	***unkmem***	reserved	***nmtx***	***nmrx***	***full***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	h	h	h	h	–	h	h	h	
Field Type	rwc	rwc	rwc	rwc	rwc	r	rwc	rwc	rw	rwc	rwc	rwc	r	rwc	rwc	rwc	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved							***link***	
Reset Value	0	0	0	0	0	0	0	–	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table 6–1. Int (Interrupt Register)

Bit	Name	Description
23:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16	***link***	Link-change interrupt. This read-only bit is the logical OR of all the ***linkchng*** bits of all the ***PortxStatus*** registers. If any of the ***linkchng*** bits are set, this interrupt is issued if it has been enabled in ***IntEnable.***
15	***new***	New node interrupt. When 1, this bit indicates that a new node has been learned from the wire and added to the address records. If this interrupt is enabled via ***IntEnable***, the node address, port, and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until read by the CPU. The lock is released when MSByte of ***NewVLAN*** register is read. DIO adds do not set this interrupt.
14	***newm***	Missed new node interrupt indication. When 1, this bit indicates that a new node interrupt was given, but the information was not placed in ***NewNode***, ***NewPort***, and ***NewVLAN*** because an earlier-enabled ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt locked them, and the information has not been read by the CPU. DIO adds do not set this interrupt.

Table 6–1. Int (Interrupt Register) (Continued)

Bit	Name	Description
13	***chng***	Node port change interrupt. When 1, this bit indicates that there has been a change in port assignment for a node that exists in the address records. If this interrupt is enabled via ***IntEnable***, the node address, port, and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until read by the CPU. The lock is released when MSByte of ***NewVLAN*** register is read. DIO adds do not set this interrupt.
12	***chngm***	Missed node port change interrupt indication. When 1, this bit indicates that a node port-change interrupt was given, but the information was not placed in ***NewNode***, ***NewPort***, and ***NewVLAN***, because an earlier-enabled ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt locked them and the information has not been read by the CPU. DIO adds do not set this interrupt.
11	***secvio***	Security violation interrupt. If the ***NewNode***, ***NewPort***, and ***NewVLAN*** registers are not locked from a previous interrupt, this bit is set when a security violation occurs (i.e., when a node that has been secured has attempted to move port assignments). If this interrupt was enabled previously via ***IntEnable*** at the same time, the ***secvio*** bit is set, the node's source address, port and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until read by the CPU. The lock is released when MSByte of ***NewVLAN*** register is read. (**SINT** also becomes active.) DIO adds do not set this interrupt.
10	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
9	***age***	Age-out interrupt. When 1, this bit indicates that a node has been aged-out (deleted from the address-lookup table). The node address is written into ***AgedNode***, the node's assigned port is written into ***AgedPort***, and the node's VLAN is written into ***AgedVLAN.*** These registers are then locked against further modification until read by the CPU. The lock is released when MSByte of ***AgedVLAN*** register is read.
8	***agem***	Missed age-out interrupt indication. When 1, this bit indicates that an age-out interrupt was given, but the information was not placed in ***AgedNode***, ***AgedPort***, and ***AgedVLAN*** because an earlier ***age*** interrupt locked them, and the information has not been read by the CPU. The lock is released when MSByte of ***AgedVLAN*** register is read.
7	***int***	Test interrupt request. Setting this bit to 1 gives a test interrupt to the attached CPU. This bit is a normal read-write bit, unlike all other bits in this register.
6	***fndcplt***	Find completion interrupt. When 1, this bit indicates that the find state machine has completed a *Find-First* or *Find-Next* operation. This bit is not set for the *Lookup* operation. Data in the ***FindPort***, ***FindControl***, ***FindNode***, and ***FindVLAN*** registers is now valid.
5	***unkvlan***	Unknown VLAN interrupt. If the ***NewNode***, ***NewPort***, and ***NewVLAN*** registers are not locked from a previous interrupt, this bit is set when an unknown IEEE Std 802.1q VLAN Identifier (i.e., one that matches none of the VLAN IDs registered in the ***VLANnQID*** registers) was received on a port whose corresponding ***UnkVLANIntPorts*** bit was 1. If this interrupt was enabled previously via ***IntEnable*** at the same time the ***unkvlan*** bit is set, the node's source address, port, and VLAN ID are written into ***NewNode,*** ***NewPort***, and ***NewVLAN***, respectively, and locked until read by the CPU. The lock is released when MSByte of ***NewVLAN*** register is read. (**SINT** also becomes active.)

Table 6–1. Int (Interrupt Register) (Continued)

Bit	Name	Description
4	***unkmem***	Unknown VLAN member interrupt. If the ***NewNode***, ***NewPort***, and ***NewVLAN*** registers are not locked from a previous interrupt, this bit is set when a frame is received on a port that is not a member of the VLAN associated with the frame, provided that both the ***UnkVLANIntPorts*** bit and the ***RxFilterPorts*** bit corresponding to the port is 1. If this interrupt was enabled previously via ***IntEnable*** at the same time the ***unkmem*** bit is set, the node's source address, port, and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until read by the CPU. The lock is released when MSByte of ***NewVLAN*** register is read. (**SINT** also will become active.)
3	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
2	***nmtx***	Network management transmit interrupt. When 1, this bit indicates that the switch has a buffer of frame data ready to be read out (transmitted) to the external CPU via the DIO interface, having previously had no data ready. This is when any of the ***eof***, ***sof***, or ***iof*** bits of ***NMTxControl*** become set to 1 after they were all previously 0, i.e., this interrupt is not reasserted when more than one buffer becomes ready. Reading ***NMTxControl*** reveals the buffer contents.
1	***nmrx***	Network management receive interrupt. When 1, this bit indicates that the switch has emptied its NM port receive buffer of frame data and is ready to receive a frame of any size up to 1535 bytes from the external CPU via the DIO interface.
0	***full***	Address-lookup table full interrupt. When 1, this bit indicates that an address has been added (either off the wire or via DIO) that caused the address records to become full. If the records are full and an attempt is made to add another address, this bit again is set to 1, assuming it had been cleared to 0 since the records became full.

6.1 Interrupts

The following paragraphs describe each interrupt and the suggested cause of actions when that interrupt is generated. Each of these interrupts can be marked via the interrupt enable register (***IntEnable*** @ 0x0808–0x080A).

6.1.1 Link-Change Interrupt (link)

An interrupt is generated when any of the ***linkchng*** bits in the ***PortxStatus*** registers change. That is, this interrupt is generated when the link is lost for a port, as well as when a port is linked up. This interrupt enables implementation of the IETF standard link-up/link-down trap (RFC 1543).

6.1.2 New-Node Interrupt (new)

An interrupt is generated when an address has been added or modified from the wire. This node's information is added/updated in the address-lookup table.

When a new node interrupt is generated, the external CPU can obtain the Ethernet address, VLAN ID, and port by reading the ***NewNode***, ***NewVLAN***, and ***NewPort*** registers. The information in these registers is locked until read by the CPU.

6.1.3 Missed New-Node Interrupt (newm)

This interrupt is generated to indicate that a new node interrupt was given, but the information was not updated in the ***NewNode***, ***NewVLAN***, and ***NewPort*** registers because an earlier enabled ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt has locked these registers and the information has not yet been read by the external CPU.

The ***NewNode***, ***NewVLAN***, and ***NewPort*** information about the missed new-node interrupt is lost.

6.1.4 Node Port-Change Interrupt (chng)

An interrupt is generated to indicate that a port assignment of an existing address contained in IALE has changed as a result of a wire event. The node's Ethernet address, VLAN ID, and port assignment can be obtained by reading the ***NewNode***, ***NewVLAN***, and ***NewPort*** registers. The information in these registers is locked until read by the CPU.

The ***NewPort*** register contains the new port assignment, as well as the old port assignment. The external CPU may have to update the VLAN with the new port assignment. That is, the new port that the node has been assigned may not be a member of the node's VLAN.

6.1.5 Missed-Node Port-Change Interrupt (chngm)

This interrupt is generated to indicate that a node port-change interrupt was generated, but the information was not updated in the ***NewNode***, ***NewVLAN***, and ***NewPort*** registers because an earlier enabled ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt has locked these registers, and the information has not yet been read by the external CPU.

The ***NewNode***, ***NewVLAN***, and ***NewPort*** information about the missed-node port-change interrupt is lost.

6.1.6 Security-Violation Interrupt (secvio)

This interrupt is generated to indicate that a node that had been secured attempted to change port assignment. The node address, port, and VLAN ID are updated in ***NewNode***, ***NewVLAN***, and ***NewPort*** register. These registers are locked until read by the external CPU.

The ***NewPort*** register contains the new port assignment as well as the old port assignment. The external CPU must decide whether to assign the node to the new port.

6.1.7 Age-Out Interrupt (age)

This interrupt is generated when a node has been aged out (automatically deleted without external CPU intervention) from the address-lookup table. The external CPU must read the node's Ethernet address, port assignment, and VLAN ID from the ***AgedNode***, ***AgedPort***, and ***AgedVLAN*** registers, respectively. These registers are locked until read by the external CPU.

6.1.8 Missed Age-Out Interrupt (agem)

This interrupt is generated to indicate that an age-out-change interrupt was given, but the information was not updated in the ***AgedNode***, ***AgedVLAN***, and ***AgedPort*** registers because an earlier enabled ***age*** interrupt has locked these registers, and the information has not yet been read by the external CPU.

The ***AgedNode***, ***AgedVLAN***, and ***AgedPort*** information about the missed age-out interrupt is lost.

6.1.9 Test-Interrupt Request (int)

This bit is writeable by the external CPU. The external CPU can use this bit to generate a test-interrupt request. By having the corresponding bit set in the ***IntEnable*** register when the external CPU writes a one into this bit, an interrupt is generated.

6.1.10 Find-Completion Interrupt (fndcplt)

This interrupt is generated when the *Find*-state machine has completed a search operation. If the search criteria matched an address record, the information will be contained in the ***FindPort***, ***FindNode***, and ***FindVLAN*** registers. Instead of polling the ***find*** bit in the ***FindControl*** register to determine when a *Find-First* or *Find-Next* operation has completed, this interrupt can be used for that purpose.

This interrupt is generated only on the *Find-First* and *Find-Next* operations. The find bit must be polled for the *Lookup* operation. Furthermore, the found bit in the ***FindControl*** register must be read to ensure that the information from the search contained in the ***FindPort***, ***FindNode***, and ***FindVLAN*** registers is valid.

6.1.11 Unknown VLAN Interrupt (unkvlan)

This interrupt indicates that a frame containing an unknown IEEE Std 802.1q VLAN identifier, i.e., VLAN did not match any of the VLANs configured in the VLANQID registers, was received on a port whose corresponding bit in the ***UnkVLANIntPorts*** register was set to 1. The address of the node, port, and VLAN ID information are contained in the ***NewNode, NewPort***, and ***NewVLAN*** registers. These registers are locked until read by the external CPU.

When this interrupt occurs, the external CPU must read the information in the ***NewNode, NewPort***, and ***NewVLAN*** registers.The external CPU then must use this information to configure a new VLAN using the ***VLANQID*** and ***VLANPort*** registers (see *VLAN Add* system operation). The ***VLANPort*** register must be configured with at least the port that received the frame as a member of the new VLAN.

Note:

The port that received the frame with an unknown IEEE Std 802 VLAN identifier must have its corresponding bit set in the ***UnkVLANIntPorts*** register for this interrupt to be generated. See ***UnkVLANIntPorts*** register for detail.

6.1.12 Unknown VLAN Member Interrupt (unkmem)

This interrupt indicates that a frame containing a known IEEE Std 802.1q VLAN identifier was received on a port that is not a member of the VLAN, i.e., the port's corresponding bit in the port vector is not set in the ***VLANPorts*** register, and the port's corresponding bit in the ***UnkVLANIntPorts*** register was set to 1. The address of the node, port, and VLAN ID information are contained in the ***NewNode, NewPort***, and ***NewVLAN*** registers. These registers are locked until read by the external CPU.

When this interrupt occurs, the external CPU must add the port as a member of the VLAN. First, the external CPU must read the information in the ***NewNode, NewPort***, and ***NewVLAN*** registers. The ***NewVLAN*** register contains the VLAN index, which must be used to access the appropriate ***VLANPort*** register. The VLAN index is used as an index into the ***VLANPort*** register table. To add the new port as a member of the VLAN, the external CPU must set the appropriate bit in the portvector of the ***VLANPort*** register.

Note:

The port that received the frame with a known IEEE Std 802.1q VLAN identifier, but was not a member of that VLAN, must have its corresponding bit set in the ***UnkVLANIntPorts*** register for this interrupt to be generated (see ***UnkVLANIntPorts*** register for details).

6.1.13 Network-Management-Transmit Interrupt (nmtx)

This interrupt is generated to indicate to the external CPU that the management port's transmit buffer was empty, but now contains data to be read by the external CPU. The management port's transmit buffer can contain only 256 bytes. Therefore, the external CPU must determine by reading the ***NMTxControl*** register whether the data contained in the buffer is the entire frame or a portion of the frame. Furthermore, this interrupt is generated only from the transition from ‘‘no data ready to be read out” to ‘‘data available to be read out.” This interrupt is not regenerated when more than one buffer becomes available for the external CPU to read out.

6.1.14 Network-Management-Receive Interrupt (nmrx)

This interrupt is generated to indicate to the external CPU that the TNETX4020 can receive a full-size frame (a frame of any size up to 1535 bytes in length) on the management port. “Management port can receive a frame” means the external CPU can “transmit” a frame via the DIO interface.

6.1.15 Address-Lookup-Table-Full Interrupt (full)

The address-lookup-table-full interrupt can be generated in two instances:

1) After an add of an address (via the DIO or wire), which causes the address-lookup table to become full.

2) If the interrupt bit is cleared from (1), this interrupt is generated after each subsequent attempt to add an address to the address-lookup table.

Clearing the table-full condition, independent of clearing the interrupt, depends on how the table is being managed.

If the internal state machines are managing the table (***nage***, ***nauto*** = 0), addresses are deleted as necessary to make room for the new table entry. If the external CPU is managing the table, the CPU must choose and delete an address to allow the add function to complete.

Appendix A

Internal Registers and Statistics

All internal registers, statistics, internal and external RAM, PHYs, and the EEPROM are accessed indirectly by the external CPU. The external CPU must use the DIO interface to access any of the internal or external structures.

Topic **Page**

A.1 Detailed Register Map A-2

A.2 Register Description A-11

A.3 Statistic Descriptions A-105

A.1 Detailed Register Map

Table A–1 shows how the register map is partitioned.

A.1.1 System and Control Registers

Table A–1. DIO Internal Register Address Map

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Port1Control		Port0Control		0x0000
Reserved				0x0004–0x003F
Reserved	UnkVLANPort	MirrorPort	UplinkPort	0x0040
Reserved		AgingThreshold		0x0044
Reserved				0x0048–0x004F
NLearnPorts				0x0050
TxBlockPorts				0x0054
RxUniBlockPorts				0x0058
RxMultiBlockPorts				0x005C
UnkUniPorts				0x0060
UnkMultiPorts				0x0064
UnkSrcPorts				0x0068
UnkVLANIntPorts				0x006C
RxFilterPorts				0x0070
Reserved				0x0074–0x008B
Reserved			RingPorts	0x008C
Reserved				0x0090–0x009F
DevCode	Reserved	SIO	Revision	0x00A0
DevNode[23:16]	DevNode[31:24]	DevNode[39:32]	DevNode[47:40]	0x00A4
Reserved		DevNode[7:0]	DevNode[15:8]	0x00A8
Reserved				0x00AC–0x00DB
McastLimit				0x00DC
RamStatus	RamControl	Reserved		0x00E0
Reserved				0x00E4
PauseTime100		Reserved		0x00E8
PauseTime1000		Reserved		0x00EC
Reserved	FlowThreshold			0x00F0
Reserved		LEDControl		0x00F4
SysControl		StatControl		0x00F8
Reserved (for EEPROM CRC)				0x00FC
VLAN0Ports				0x0100
VLAN1Ports				0x0104
VLAN2Ports				0x0108
VLAN3Ports				0x010C
VLAN4Ports				0x0110
VLAN5Ports				0x0114
VLAN6Ports				0x0118
VLAN7Ports				0x011C
VLAN8Ports				0x0120
VLAN9Ports				0x0124

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
VLAN10Ports				0x0128
VLAN11Ports				0x012C
VLAN12Ports				0x0130
VLAN13Ports				0x0134
VLAN14Ports				0x0138
VLAN15Ports				0x013C
VLAN16Ports				0x0140
VLAN17Ports				0x0144
VLAN18Ports				0x0148
VLAN19Ports				0x014C
VLAN20Ports				0x0150
VLAN21Ports				0x0154
VLAN22Ports				0x0158
VLAN23Ports				0x015C
VLAN24Ports				0x0160
VLAN25Ports				0x0164
VLAN26Ports				0x0168
VLAN27Ports				0x016C
VLAN28Ports				0x0170
VLAN29Ports				0x0174
VLAN30Ports				0x0178
VLAN31Ports				0x017C
VLAN32Ports				0x0180
VLAN33Ports				0x0184
VLAN34Ports				0x0188
VLAN35Ports				0x018C
VLAN36Ports				0x0190
VLAN37Ports				0x0194
VLAN38Ports				0x0198
VLAN39Ports				0x019C
VLAN40Ports				0x01A0
VLAN41Ports				0x01A4
VLAN42Ports				0x01A8
VLAN43Ports				0x01AC
VLAN44Ports				0x01B0
VLAN45Ports				0x01B4
VLAN46Ports				0x01B8
VLAN47Ports				0x01BC
VLAN48Ports				0x01C0
VLAN49Ports				0x01C4
VLAN50Ports				0x01C8
VLAN51Ports				0x01CC
VLAN52Ports				0x01D0
VLAN53Ports				0x01D4

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
VLAN54Ports				0x01D8
VLAN55Ports				0x01DC
VLAN56Ports				0x01E0
VLAN57Ports				0x01E4
VLAN58Ports				0x01E8
VLAN59Ports				0x01EC
VLAN60Ports				0x01F0
VLAN61Ports				0x01F4
VLAN62Ports				0x01F8
VLAN63Ports				0x01FC
Reserved				0x0200–0x02FF
VLAN1QID		VLAN0QID		0x0300
VLAN3QID		VLAN2QID		0x0304
VLAN5QID		VLAN4QID		0x0308
VLAN7QID		VLAN6QID		0x030C
VLAN9QID		VLAN8QID		0x0310
VLAN11QID		VLAN10QID		0x0314
VLAN13QID		VLAN12QID		0x0318
VLAN15QID		VLAN14QID		0x031C
VLAN17QID		VLAN16QID		0x0320
VLAN19QID		VLAN18QID		0x0324
VLAN21QID		VLAN20QID		0x0328
VLAN23QID		VLAN22QID		0x032C
VLAN25QID		VLAN24QID		0x0330
VLAN27QID		VLAN26QID		0x0334
VLAN29QID		VLAN28QID		0x0338
VLAN31QID		VLAN30QID		0x033C
VLAN33QID		VLAN32QID		0x0340
VLAN35QID		VLAN34QID		0x0344
VLAN37QID		VLAN36QID		0x0348
VLAN39QID		VLAN38QID		0x034C
VLAN41QID		VLAN40QID		0x0350
VLAN43QID		VLAN42QID		0x0354
VLAN45QID		VLAN44QID		0x0358
VLAN47QID		VLAN46QID		0x035C
VLAN49QID		VLAN48QID		0x0360
VLAN51QID		VLAN50QID		0x0364
VLAN53QID		VLAN52QID		0x0368
VLAN55QID		VLAN54QID		0x036C
VLAN57QID		VLAN56QID		0x0370
VLAN59QID		VLAN58QID		0x0374
VLAN61QID		VLAN60QID		0x0378
VLAN63QID		VLAN62QID		0x037C
Port1QTag		Port0QTag		0x0380

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Reserved				0x0384–0x03FF
Port1Status		Port0Status		0x0400
Reserved				0x0404–0x043F
FindNode[23:16]	FindNode[31:24]	FindNode[39:32]	FindNode[47:40]	0x0440
FindVLAN	FindControl	FindNode[7:0]	FindNode[15:8]	0x0444
FindPort				0x0448
NewNode[23:16]	NewNode[31:24]	NewNode[39:32]	NewNode[47:40]	0x044C
Reserved		NewNode[7:0]	NewNode[15:8]	0x0450
NewVLAN		NewPort		0x0454
AddNode[23:16]	AddNode[31:24]	AddNode[39:32]	AddNode[47:40]	0x0458
AddVLAN	AddDelControl	AddNode[7:0]	AddNode[15:8]	0x045C
AddPort				0x0460
AgedNode[23:16]	AgedNode[31:24]	AgedNode[39:32]	AgedNode[47:40]	0x0464
AgedVLAN	AgedPort	AgedNode[7:0]	AgedNode[15:8]	0x0468
DelNode[23:16]	DelNode[31:24]	DelNode[39:32]	DelNode[47:40]	0x046C
DelVLAN	DelPort	DelNode[7:0]	DelNode[15:8]	0x0470
AgingCounter		NumNodes		0x0474
Reserved				0x0478–0x0543
XMultiGroup17				0x0544
XMultiGroup18				0x0548
XMultiGroup19				0x054C
XMultiGroup20				0x0550
XMultiGroup21				0x0554
XMultiGroup22				0x0558
XMultiGroup23				0x055C
XMultiGroup24				0x0560
XMultiGroup25				0x0564
XMultiGroup26				0x0568
XMultiGroup27				0x056C
XMultiGroup28				0x0570
XMultiGroup29				0x0574
XMultiGroup30				0x0578
XMultiGroup31				0x057C
XMultiGroup32				0x0580
XMultiGroup33				0x0584
XMultiGroup34				0x0588
XMultiGroup35				0x058C
XMultiGroup36				0x0590
XMultiGroup37				0x0594
XMultiGroup38				0x0598
XMultiGroup39				0x059C
XMultiGroup40				0x05A0
XMultiGroup41				0x05A4
XMultiGroup42				0x05A8

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
XMultiGroup43				0x05AC
XMultiGroup44				0x05B0
XMultiGroup45				0x05B4
XMultiGroup46				0x05B8
XMultiGroup47				0x05BC
XMultiGroup48				0x05C0
XMultiGroup49				0x05C4
XMultiGroup50				0x05C8
XMultiGroup51				0x05CC
XMultiGroup52				0x05D0
XMultiGroup53				0x05D4
XMultiGroup54				0x05D8
XMultiGroup55				0x05DC
XMultiGroup56				0x05E0
XMultiGroup57				0x05E4
XMultiGroup58				0x05E8
XMultiGroup59				0x05EC
XMultiGroup60				0x05F0
XMultiGroup61				0x05F4
XMultiGroup62				0x05F8
XMultiGroup63				0x05FC
PCS0Status		PCS0Control		0x0600
Reserved				0x0604
PCS0ANLinkP		PCS0ANAdvert		0x0608
PCS0ANNxt		PCS0ANExp		0x060C
Reserved		PCS0ANLinkPNxt		0x0610
Reserved				0x0614–0x061B
PCS0ExtStatus		Reserved		0x061C
PCS1Status		PCS1Control		0x0620
Reserved				0x0624
PCS1ANLinkP		PCS1ANAdvert		0x0628
PCS1ANNxt		PCS1ANExp		0x062C
Reserved		PCS1ANLinkPNxt		0x0630
Reserved				0x0634–0x063B
PCS1ExtStatus		Reserved		0x063C
Reserved				0x0640–0x07FF
Reserved		DMAAddress		0x0800
Reserved	Int			0x0804
Reserved	IntEnable			0x0808
SysTest	FreeStackLength			0x080C
RAMAddress				0x0810
Reserved			RAMData	0x0814
Reserved	NMRxControl			0x0818

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Reserved	NMTxControl			0x081C
Reserved			NMData	0x0820
Reserved				0x0824–0x0FFF
Manufacturing Test Registers – Internal Use Only				0x1000–0x11FF
Reserved				0x1200–0x3FFF
Hardware reset				0x4000–0x5FFF
Reserved				0x6000–0x7FFF
Port0 Rx Octets				0x8000
Port0 Good Rx Frames				0x8004
Port0 Broadcast Rx Frames				0x8008
Port0 Multicast Rx Frames				0x800C
Port0 Rx CRC Errors				0x8010
Port0 Rx Align/Code Errors				0x8014
Port0 Oversized Rx Frames				0x8018
Port0 Rx Jabbers				0x801C
Port0 Undersized Rx Frames				0x8020
Port0 Rx Fragments				0x8024
Port0 64octet Frames				0x8028
Port0 65–127octet Frames				0x802C
Port0 128–255octet Frames				0x8030
Port0 256–511octet Frames				0x8034
Port0 512–1023octet Frames				0x8038
Port0 1024–1518octet Frames				0x803C
Port0 Net Octets				0x8040
Port0 SQE Test Errors				0x8044
Port0 Tx Octets				0x8048
Port0 Good Tx Frames				0x804C
Port0 Single Collision Tx Frames				0x8050
Port0 Multiple Collision Tx Frames				0x8054
Port0 Carrier Sense Errors				0x8058
Port0 Deferred Tx Frames				0x805C
Port0 Late Collisions				0x8060
Port0 Excessive Collisions				0x8064
Port0 Broadcast Tx Frames				0x8068
Port0 Multicast Tx Frames				0x806C
Port0 Filtered Rx Frames				0x8070
Port0 Tx Data Errors				0x8074
Port0 Collisions				0x8078
Port0 Rx Overruns				0x807C
Port1 Rx Octets				0x8080
Port1 Good Rx Frames				0x8084
Port1 Broadcast Rx Frames				0x8088
Port1 Multicast Rx Frames				0x808C
Port1 Rx CRC Errors				0x8090

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Port1 Rx Align/Code Errors				0x8094
Port1 Oversized Rx Frames				0x8098
Port1 Rx Jabbers				0x809C
Port1 Undersized Rx Frames				0x80A0
Port1 Rx Fragments				0x80A4
Port1 64octet Frames				0x80A8
Port1 65–127octet Frames				0x80AC
Port1 128–255octet Frames				0x80B0
Port1 256–511octet Frames				0x80B4
Port1 512–1023octet Frames				0x80B8
Port1 1024–1518octet Frames				0x80BC
Port1 Net Octets				0x80C0
Port1 SQE Test Errors				0x80C4
Port1 Tx Octets				0x80C8
Port1 Good Tx Frames				0x80CC
Port1 Single Collision Tx Frames				0x80D0
Port1 Multiple Collision Tx Frames				0x80D4
Port1 Carrier Sense Errors				0x80D8
Port1 Deferred Tx Frames				0x80DC
Port1 Late Collisions				0x80E0
Port1 Excessive Collisions				0x80E4
Port1 Broadcast Tx Frames				0x80E8
Port1 Multicast Tx Frames				0x80EC
Port1 Filtered Rx Frames				0x80F0
Port1 Tx Data Errors				0x80F4
Port1 Collisions				0x80F8
Port1 Rx Overruns				0x80FC
NM Port Rx Octets				0x8100
NM Port Good Rx Frames				0x8104
NM Port Broadcast Rx Frames				0x8108
NM Port Multicast Rx Frames				0x810C
NM Port Rx CRC Errors				0x8110
Reserved				0x8114
NM Port Oversized Rx Frames				0x8118
NM Port Rx Jabbers				0x811C
NM Port Undersized Rx Frames				0x8120
NM Port Rx Fragments				0x8124
NM Port 64octet Frames				0x8128
NM Port 65–127octet Frames				0x812C
NM Port 128–255octet Frames				0x8130
NM Port 256–511octet Frames				0x8134
NM Port 512–1023octet Frames				0x8138
NM Port 1024–1518octet Frames				0x813C
NM Port Net Octets				0x8140

Table A–1. DIO Internal Register Address Map (Continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Reserved				0x8144
NM Port Tx Octets				0x8148
NM Port Good Tx Frames				0x814C
Reserved				0x8150–0x8167
NM Port Broadcast Tx Frames				0x8168
NM Port Multicast Tx Frames				0x816C
NM Port Filtered Rx Frames				0x8170
NM Port Tx Data Errors				0x8174
Reserved				0x8178–0x8FFF
Port0 Pause Tx Frames				0x9000
Port0 Pause Rx Frames				0x9004
Port0 Security Violations				0x9008
Reserved				0x900C
Port1 Pause Tx Frames				0x9010
Port1 Pause Rx Frames				0x9014
Port1 Security Violations				0x9018
Reserved				0x901C–0x9027
NM Port Security Violations				0x9028
Reserved				0x902C–9FFF
Unknown Unicast Destination Addresses				0xA000
Unknown Multicast Destination Addresses				0xA004
Unknown Source Addresses				0xA008
Reserved				0xA00C–0xFFFF

A.1.2 Port and Network Management (NM) Port Statistics

Each port has associated statistics that record events associated with frame traffic on the port (see Table A–2). In addition, there are three statistics relating to IALE's address lookups. Most of the statistics are mapped into the DIO address space, beginning at address 0x8000. Each statistic is 32 bits wide.

When accessing the statistics values from the DIO port, it is necessary to perform four 1-byte DIO reads to obtain the full 32-bit counter. Counters should always be read in ascending byte-address order (0, 1, 2, 3). To prevent the chance of the counter being updated while reading the four bytes, the entire 32-bit counter value is transferred to a holding register when byte 0 is read.

To provide ease of use with both big- and little-endian CPUs, two alternative byte-ordering schemes are supported for DIO addresses of 0x8000 and up:

- If ***bigend*** = 0 in ***StatControl***, byte address of 0 references the numerically least significant byte of a counter, and a byte address of 3 references the numerically most significant byte of a counter.
- If ***bigend*** = 1 in ***StatControl***, byte address of 0 references the numerically most significant byte of a counter, and a byte address of 3 references the numerically least significant byte of a counter.

There are 35 statistics counters maintained on a per-port basis for both 100/1000-Mbit/s ports. Of these, the first 32 are packed contiguously in DIO address space in a range starting at 0x8000. The remaining statistics are packed contiguously in an address range starting at 0x9000.

In addition, there are three address-lookup statistics in the range 0xA000–0xA00B.

The statistics can be cleared via the ***clrp***, ***portcode***, and ***clra*** bits in ***StatControl***.

If ***stop*** is set to 1 to halt the switch after it has been running, the statistics freeze at their current values. Any partial frame counts held in hardware that have not yet been updated into the actual statistics mapped at the DIO addresses are lost (cleared).

Until ***start*** is set to 1, the statistics can be written, as well as read. The bytes are built into an 8-byte word that is written into the statistics RAM when any of the eight bytes is written. This grouping also is used to clear the statistics.

Setting ***start*** to 1 does not clear the statistics. Once ***start*** has been set, the statistics become read only. If a clear is active at the time ***stop*** is set, the clear restarts after ***start*** is set.

All statistics roll over from 0xFFFFFFFF–0x00000000.

A.2 Register Description

Registers are grouped according to whether or not they can be loaded from EEPROM. To allow software to easily distinguish between devices, the ***DevCode*** register is at the same address as other devices in TI's ThunderSWITCH family.

The following paragraphs describe the registers in ascending DIO address order. The state of each register after hard reset and stop is given. The type of each register bit or field is identified in Table A–2. Table A–3 identifies which registers can be loaded from an EEPROM.

Table A–2. Register Key

Type	Description
w	Writeable bit
r	Readable bit
wc	Write-to-clear bit. Writing a 1 to a bit causes it to be set to 0. Writing a 0 has no effect.
rc	Read-to-clear bit. Reading a bit of this type returns its current value and then clears it to 0. Writing to a bit of this type has no effect.
p	Protected bit. Written only when both ***start*** = 0 and ***initd*** = 0 in ***SysControl***.
l	Loadable bit. Written only from EEPROM.
h	Cleared by hard reset (asserting $\overline{\textbf{RESET}}$, or writing to DIO address 0x4000–0x5FFF)
s	Cleared by stop (setting ***stop*** in ***SysControl***)
–	Unaffected

Table A–3. EEPROM-Loadable Map

EEPROM Loadable	Name	DIO Address
yes	Port-control registers	0x0000–0x003F
yes	Address-lookup registers	0x0040–0x009F
yes	System registers	0x00A0–0x00FF
yes	VLAN registers (24C08 EEPROM only)	0x0100–0x03FF
no	Port-status registers	0x0400–0x043F
no	Address-lookup registers	0x0440–0x05FF
no	PCS registers	0x0600–0x07FF
no	System registers	0x0800–0x0FFF

Several field names recur in many registers, particularly field names that specify where to deliver a frame that has been received. A detailed definition of these fields is in Appendix A with the numerical order definitions of the internal registers. The destinations for packets include ports 0, 1, and the management port on this device, and remote ports connected to port 1 via a crossbar-type switching element. If a packet is forwarded to port 1, and pretagging in enabled in the ***Port1Control*** register, then additional information is required to point to the external port(s) that receive the frame. The additional information is used to build the tag prepended to the packet to steer it through the crossbar. The format of the pretag is shown in the TNETX4020 data sheet, literature number SPWS049A. A summary is given in Table A–4.

Table A–4. Recurrent Field Code Definition

Width	Name	Description
4	***portcode***	Binary encoding that selects a single port on this device: ❏ 00–10 switch ports 0–2 (network management) ❏ 11 reserved
6	***xportcode***	Binary encoding that selects a single port on this device or a crossbar-matrix connected to port 8: ❏ 000000–000010 switch ports 0–2 (network management) ❏ 000011–011111 reserved ❏ 100000–110000 crossbar matrix ports 0–16 ❏ 110001–111111 reserved
10	***portvector***	Bit vector where bit *x* corresponds to switch port *x*. Any number of ports can be selected at once.
6	***xroutecode***	Binary encoding that selects one of 64 crossbar-matrix port vectors (***xportvectors***): ❏ 000000–010000 bit vector with only bit *n* set, where *n* is the decimal value of the coding (0–16) ❏ 010001–111111 bit vector contained in ***XMultiGroupn*** register, where ***n*** is the decimal value of the coding (17–63)
17	***xportvector***	Bit vector where bit *x* corresponds to crossbar-matrix port *x*. Any number of ports can be selected at the same time.

PortxControl (Port x Control Register) @ 0x0000–0x0003

Ports 0–1 are 100-/1000-Mbit/s ports. x = 0, 1.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	reserved		*tolerant*	*maxlen*	*txacc*	*rxacc*	*pretag*	*neg*	*disable*	*reqntxp*	*lckref*	*reqnrxp*	*reqpma*	*req100*	*txpace*	*reqhd*	DIO Address 0x0000 +0x2*x
Reset Value	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–5. PortxControl (Port x Control Register)

Bit	Name	Description
15:14	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
13	***tolerant***	Frame length tolerance. Determines the minimum frame length considered as a valid frame when operating in half duplex at 1000 Mbit/s (***duplex*** = 0 and ***speed*** = 1 in ***PortxStatus***), to provide some tolerance for slightly noncompliant IEEE Std 802.3z systems: ❏ ***tolerant*** = 1. Frames less than 510 bytes in length (including carrier extension) are discarded. ❏ ***tolerant*** = 0. Frames less than 512 bytes in length (including carrier extension) are discarded. This bit has no effect when ***duplex*** = 1 or ***speed*** = 0.
12	***maxlen***	Maximum frame length. Determines the maximum frame length at which a received frame is discarded. ❏ ***maxlen*** = 1. The maximum received frame length is 1518 bytes, as specified by IEEE Std 802.3. This is the maximum length on the wire. If a VLAN header is inserted into a 1518-byte frame within the MAC, the frame is stored as a 1522-byte frame within the switch. ❏ ***maxlen*** = 0. The maximum received frame length is 1535 bytes if no VLAN header is inserted, or 1531 bytes if a VLAN header is inserted. (When stored within the switch, a frame never can be longer than 1535 bytes). This bit has no effect on the maximum frame size that can be transmitted by the port.
11	***txacc***	Transmit access. Identifies whether the port is connected to the network beyond the switching system (an access port), or to another similarly capable switch device within the system (an intraswitch port). In turn, this determines how the port handles VLANs during transmission. ❏ ***txacc*** = 1. The first IEEE Std 802.1q tag header in the frame is stripped from all frames transmitted from this port. ❏ ***txacc*** = 0. The first IEEE Std 802.1q tag header in the frame is stripped from the frame only if the VLAN ID within it is identical to ***vlanqid*** in the port's ***PortxQTag*** register.

Table A–5. PortxControl (Port x Control Register) (Continued)

Bit	Name	Description
10	***rxacc***	Receive access. Identifies whether the port is connected to the network beyond the switching system (an access port), or to another similarly capable switch device within the system (an intraswitch port). In turn, this determines how the port handles VLANs during reception. ❑ ***rxacc*** = 1. An IEEE Std 802.1q tag header equal to the port's ***PortxQTag*** register is added to all frames received on this port, even if they already include one. ❑ ***rxacc*** = 0. An IEEE Std 802.1q tag header equal to the port's ***PortxQTag*** register is added to all untagged frames received on this port. If the frame already contains an IEEE Std 802.1q tag header but its VLAN ID is 0x000, the VLAN ID in the header is replaced by ***vlanqid*** from the port's ***PortxQTag*** register.
9	***pretag***	Preframe tagging. Controls support for frame tagging on Port 1 only. This bit is reserved for Port 0. ❑ In MII and PMA mode this bit is always 0; writes have no effect. ❑ ***pretag*** = 1. The device expects frames received on this port to have a pretag, and adds a pretag to all frames transmitted from this port. ❑ ***pretag*** = 0. Frames received on this port should not have a pretag. No pretag is added to transmitted frames. See *external port awareness* for details of tag formats and interpretation.
8	***neg***	Negotiation. In PMA mode. ❑ ***neg*** = 1. The on-chip PCS autonegotiates the pause and duplex capabilities described by ***reqntxp***, ***reqhd,*** and ***reqnrxp*** bits across the PMA interface. ❑ ***neg*** = 0. Autonegotiation is disabled and the mode of operation described by ***reqntxp***, ***reqhd,*** and ***reqnrxp*** bits are adopted by the port. ❑ Writing a 0 followed by a 1 to this bit forces the on-chip PCS to restart autonegotiation, even if a link already has been established and is active. This bit also can be accessed via ***enaneg*** in ***PCSxControl***. Refer to the ***PCSxControl*** description. When not in PMA mode, this bit has no effect. ❑ Writing a 1 to this bit when the previous value was 0 causes **$\overline{\text{Mxx\_LREF}}$** to be asserted active low if **Mxx_LINK** is high. It remains low until **Mxx_LINK** is deactivated.
7	***disable***	Port disable. Writing a 1 to this bit disables the port. Frames are not forwarded from or to a disabled port. Any previously queued frames are taken off the transmit queue as if they were being transmitted, but they are not sent to the PHY, i.e., they are discarded. **Mxx_TXEN** is not asserted. The state of the MAC-PHY interface signals does not impede this queue-clearing process.

Table A–5. PortxControl (Port x Control Register) (Continued)

Bit	Name	Description
6	***reqntxp***	Request no transmit pause. The effect of this bit depends on the mode of operation for which the port is configured, as determined by ***speed*** and ***pma*** in ***PortxStatus***. When configured for PMA mode (***pma*** = 1): ❑ ***reqntxp*** = 1. The PCS layer advertises an inability to transmit IEEE Std 802.3x pause frames during autonegotiation. ❑ ***reqntxp*** = 0. The PCS layer advertises an ability to transmit IEEE Std 802.3x pause frames during autonegotiation. When configured for GMII mode (***speed*** = 1, ***pma*** = 0): ❑ ***reqntxp*** = 1. Transmission of IEEE Std 802.3x pause frames is disabled for this port. ❑ ***reqntxp*** = 0. Transmission of IEEE Std 802.3x pause frames is supported for this port. When configured for MII mode (***speed*** =0): ❑ ***reqntxp*** = 1. Transmission of IEEE Std 802.3x pause frames is disabled for this port, and pause frames received on this port are treated like any other multicast frame. ❑ ***reqntxp*** = 0. IEEE Std 802.3x pause frames are supported for this port, unless Mxx_RXD5 is pulled low by the attached PHY or PMI device. Alternatively, ***reqntxp*** can be read or written via ***pause*** and ***asym*** in ***PCSxANAdvert***, but the encoding is different. See ***PCSxANAdvert*** definition for details.
5	***lckref***	Lock to reference. In PMA mode (***pma*** = 1 in ***PortxStatus***), the inverse value of this bit directly controls the **Mxx_LREF** pin when hardware reset is not being applied. **Mxx_LREF** is active during hardware reset. The pin is attached to an external PMA sublayer device, and causes it to lock to its reference clock. When not in PMA mode, this bit has no effect.
4	***reqnrxp***	Request no receive pause. The effect of this bit depends on the mode of operation for which the port is configured, as determined by ***speed*** and ***pma*** in ***PortxStatus***. When configured for PMA mode (***pma*** = 1): ❑ ***reqnrxp*** = 1. The PCS layer advertises an inability to respond to IEEE Std 802.3x pause frames during autonegotiation. ❑ ***reqnrxp*** = 0. The PCS layer advertises an ability to respond to IEEE Std 802.3x pause frames during autonegotiation. When configured for GMII mode (***speed =*** 1, ***pma*** = 0): ❑ ***reqnrxp*** = 1. IEEE Std 802.3x pause frames received on this port are ignored, but pause frames are absorbed by the MAC. ❑ ***reqnrxp*** = 0. Reception of IEEE Std 802.3x pause frames is supported for this port. When configured for 100 Mbit/s (***speed =*** 0): ❑ ***reqnrxp*** = 1. Transmission of IEEE Std 802.3x pause frames is disabled for this port, and pause frames received on this port are ignored and absorbed by the MAC. The open-drain **Mxx_RXD5** pin is asserted low. ❑ ***reqnrxp*** = 0. IEEE Std 802.3x pause frames are supported for this port, unless **Mxx_RXD5** is pulled low by the attached PHY or PMI device. Alternatively, ***reqnrxp*** can be read or written via ***pause*** and ***asym*** in ***PCSxANAdvert***, though the encoding is different. See ***PCSxANAdvert*** definition for details.

Table A–5. PortxControl (Port x Control Register) (Continued)

Bit	Name	Description
3	***reqpma***	Request PMA mode. When high, the port is configured to interface to an external PMA device, and the open-drain **Mxx_PMA** pin is pulled low. When low, the port is configured to interface only to an external PMA device if the **Mxx_PMA** pin is pulled low by external hardware.
2	***req100***	Request 100 Mbit/s. Used to control the speed of the port when not in PMA mode. ❑ ***req100*** = 1. The port is configured for 100-Mbit/s operation, and **Mxx_MII** is pulled low. ❑ ***req100*** = 0. The port is configured for 1000-Mbit/s operation unless **Mxx_MII** is pulled low externally. In PMA mode, if this bit is set, **Mxx_MII** is pulled low.
1	***txpace***	Transmit pacing. When high, the port uses transmission pacing to enhance performance.
0	***reqhd***	Request half duplex. When high, this bit puts the port in half-duplex mode. In MII mode, **Mxx_RXD4** also is asserted low. The effect of this bit when low depends on the mode of operation for which the port is configured: ❑ When in GMII mode and ***reqhd*** = 0, the port is configured for full-duplex mode. ❑ When in MII mode and ***reqhd*** = 0, the port is configured for full-duplex mode, *unless* **Mxx_RXD4** is pulled low externally. ❑ When in PMA mode and ***reqhd*** = 1, the PCS layer attempts to negotiate half-duplex operation with the attached PMD device. The result of this negotiation is determined by reading the ***duplex*** bit in the ***PortxStatus*** register. Alternatively, and depending on the value of ***neg***, this bit can be read or written via either ***pcsdp*** in ***PCSxControl*** (***neg*** = 0) or ***canfdp*** in ***PCSxANAdvert*** (***neg*** = 1). See ***PCSxControl*** and ***PCSxANAdvert*** descriptions.

UplinkPort (Uplink Routing Register) @ 0x0040

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–6. UplinkPort (Uplink Routing Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Uplink extended port code. Identifies the port to which all mirrored frames are destined. Two types of mirroring are possible: ❑ Destination-address mirroring. Individual unicast addresses can be mirrored if the ***cuplnk*** bit for that address is set. In this case, all frames destined to this address are forwarded to the port specified in this field. ❑ Port mirroring. A port selected through ***MirrorPor****t* can be selected for port mirroring. In this case, all frames received by or destined for this port are copied to the port specified in this field. This includes frames that normally would not be routed through the device, such as when both destination and source are assigned to the same port. Port mirroring is possible only if ***mirr*** in ***SysControl*** is set. If this field specifies an unimplemented port number, that port is not included in the port routing code. Selection of a reserved crossbar-switch port is interpreted as an ***xroutecode***, and the pretag is derived by indexing into the ***XMultiGroup*** registers.

MirrorPort (Mirrored Port Select Register) @ 0x0041

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved						***portcode[1:0]***		
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	h	h	
Field Type	r	r	r	r	r	r	rwl	rwl	

Table A–7. MirrorPort (Mirrored Port Select Register)

Bit	Name	Description
7:2	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
1:0	***portcode***	Mirror extended port code. Identifies the port that is to be monitored when ***mirr*** in ***SysControl*** is set. All frames received by or destined for this port are copied to the port identified by the ***xportcode*** field of ***UplinkPort*****,** unless ***UplinkPort*** contains the same port number as the source port. If the port code names an unimplemented port, this option has no effect on the port routing code.

UnkVLANPort (Unknown VLAN Routing Register) @ 0x0042

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–8. UnkVLANPort (Unknown VLAN Routing Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Unknown VLAN extended port code. Identifies the port to which frames with multicast destination addresses and an unknown IEEE Std 802.1q VLAN Identifier (i.e., one that matches none of the VLAN IDs registered in ***VLAN0QID***–***VLAN63QID***) are forwarded, provided ***unkvlan*** in ***SysControl*** is set. Frames that have a unicast destination address and an unknown VLAN always are discarded, although they can cause an ***unkvlan*** interrupt if it is unmasked. If this field specifies an unimplemented port number, this port is not included in the port-routing code. Selection of a reserved crossbar-switch port is interpreted as an ***xroutecode***, and the pretag is derived by indexing into the ***XMultiGroup*** registers.

AgingThreshold (Age Out Time Select Register) @ 0x0044–0x0045

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address Offset 0x0
Field Name	***threshold[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–9. AgingThreshold (Age Out Time Select Register)

Bit	Name	Description
15:0	***threshold***	Aging threshold. There are two aging modes that are selected by the value of this register. ❑ When ***threshold*** is 0x0000 or 0xFFFF, table-full aging is performed. The oldest address is aged out only when the address-lookup table becomes full. The ***AgingCounter*** increments by one only after a new address is added to the address records. ❑ When ***threshold*** is not 0x0000 or 0xFFFF, threshold aging is performed. The value in this field is the time threshold in multiples of 8 seconds. (i.e., 0x0001 = 8 seconds). All address records that are older than this time are aged out. The ***AgingCounter*** increments by one every 8 seconds. Aging does not delete unicast addresses that have been locked or secured, or multicast addresses. ❑ Aging is disabled when ***nauto*** bit and/or the ***nage*** bit is set to 1 in ***SysControl***. It then becomes the external CPU's responsibility to delete unwanted address records. The ***AgingCounter*** continues to increment in the manner selected by this ***threshold*** value, i.e., only after address adds, or once every 8 seconds, allowing the address-deletion criteria to be determined by the user.

NLearnPorts (Address Learning Disable Register) @ 0x0050–0x0053

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–10. NLearnPorts (Address Learning Disable Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Disable learning port vector. Disables learning for the ports whose corresponding bit is set. Frames received on a port where learning is disabled are forwarded, but are not added to the address-lookup table.

TxBlockPorts (Block Transmit Register) @ 0x0054–0x0057

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–11. TxBlockPorts (Block Transmit Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Block transmit port vector. Blocks forwarding of frames, depending on the port to which the frame is destined. Frames destined to ports whose corresponding bit is set are discarded.

RxUniBlockPorts (Block Unicast Receive Register) @ 0x0058–0x005B

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–12. RxUniBlockPorts (Block Unicast Receive Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Block unicast receive port vector. Blocks forwarding of unicast frames, depending on the port from which the frame originated. Unicast frames received on ports whose corresponding bit is set are discarded. ***RxUniBlockPorts*** can be overridden if the destination address in the address-lookup table has the ***nblck*** bit set.

RxMultiBlockPorts (Block Multicast Receive Register) @ 0x005C–0x005F

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved													***portvector [2:0]***			0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset
Field Name	reserved																0x2
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–13. RxMultiBlockPorts (Block Multicast Receive Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Block multicast receive port vector. Blocks forwarding of multicast frames, depending on the port from which the frame originated. Multicast frames received on ports whose corresponding bit is set are discarded. ***RxMultiBlockPorts*** can be overridden if the destination address in the address-lookup table has the ***nblck*** bit set.

UnkUniPorts (Unknown Unicast Address Routing Register) @ 0x0060–0x0063

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table A–14. UnkUniPorts (Unknown Unicast Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown unicast destination address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown unicast destination port vector. When a frame has a unicast destination address that is not contained within the address-lookup table, this field determines to which ports the frame should be forwarded.

UnkMultiPorts (Unknown Multicast Address Routing Register) @ 0x0064–0x0067

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table A–15. UnkMultiPorts (Unknown Multicast Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown multicast destination address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown multicast destination port vector. When a frame has a multicast destination address that is not contained within the address-lookup table, this field determines to which ports the frame should be forwarded.

UnkSrcPorts (Unknown Source Address Routing Register) @ 0x0068–0x006B

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved		***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rwl	rwl	rwl	rwl	rwl	rwl	r	r	r	r	r	r	r	r	

Table A–16. UnkSrcPorts (Unknown Source Address Routing Register)

Bit	Name	Description
31:30	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
29:24	***xroutecode***	Extended route code. This field indicates the route code used to generate the preframe tag added to frames with an unknown source address routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Unknown source port vector. When a frame is received from an address that is not contained within the address-lookup table, this field determines to which ports the frame should be forwarded.

UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register) @ 0x006C–0x006F

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–17. UnKVLANIntPorts (Unknown VLAN Interrupt Enable Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Enable unknown VLAN interrupts port vector. Enables generation of unknown VLAN and unknown VLAN member interrupts for the ports whose corresponding bit is set, provided the appropriate bits in ***IntEnable*** are set also. This vector has no effect on how frames are forwarded or filtered.

RxFilterPorts (Receive Filtering Enable Register) @ 0x0070–0x0073

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–18. RxFilterPorts (Receive Filtering Enable Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	Receive (ingress) filtering port vector. Enables receive filtering as defined in IEEE Std 802.1q section 8.4.5. Frames received that are associated with a VLAN of which the receiving port is not a member are discarded for those ports whose corresponding bit from this field is set.

RingPorts (Ring Topology Enable Register) @ 0x008C

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***ringid[3:0]***				***holbrm***	reserved	***portvector[1:0]***		
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	r	rwl	rwl	

Table A–19. RingPorts (Ring Topology Enable Register)

Bit	Name	Description
7:4	***ringid***	Ring ID. Specifies the ID of this device in ring topology mode. Each device on the ring should be configured with a different nonzero value of this field. Frames received on a nonring port and then transmitted from a ring port use this field to form the out-of-band pretag transmitted with the frame. In this way, frames are tagged with the ID of the device that introduced them to the ring. Frames received from a ring port that have a pretag that matches this field are filtered. This ensures that frames circumnavigate the ring only once.
3	***holbrm***	HOLB ring mode. The ***holbrm*** bit works only in conjunction with the ***flow*** and ***holb*** bits. ❑ When ***holbrm***, ***flow***, and ***holb*** bits are set, ports enabled for ring mode assert the **Mxx_FLOW** pin when the half FlowThreshold value is exceeded. Setting ***holbrm*** has no effect on the ***FLOW*** led threshold. ❑ When ***holbrm*** is reset, and ***flow*** and ***holb*** bits are set, ports enabled for ring mode assert the **Mxx_FLOW** pin when the HOLB threshold or half FlowThreshold value is exceeded, whichever occurs first. ❑ If either ***flow*** or ***holb*** is reset, setting ***holbrm*** has no effect.
2	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
1:0	***portvector***	Ring topology port vector. Determines which 100-/1000-Mbit/s ports are configured for operation in a ring topology. When set, the following capability is enabled: ❑ Frames received on the port can be transmitted back out the same port. When both ports are configured for ring mode, where the switch is acting as a bridge between two rings, frames have the ***pretag*** stripped on reception and replaced with ***ringid*** on transmission. Both ports use the same ***ringid***, which needs to be unique on both rings. If any ports are configured for ring topology, the ***pretag*** bit in ***Port1Control*** must be clear.

Revision (Revision Register) @ 0x00A0

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***majorrev[3:0]***				***minorrev[3:0]***				
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table A–20. Revision (Revision Register)

Bit	Name	Description
7:4	***majorrev***	Major hardware revision code. Indicates the major device revision number.
3:0	***minorrev***	Minor hardware revision code. Indicates the minor device revision number.

SIO (Serial Interface I/O Register) @ 0x00A1

Note:

This register is not loaded during an EEPROM load.

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***nmrst***	***mclk***	***mtxen***	***mdata***	***mdioen***	***eclk***	***etxen***	***edata***	
Reset Value	1	0	0	**MDIO**	0	0	0	**EDIO**	
Reset Type	h	h	h	–	h	h	h	–	
Field Type	rw	rw	rw	rw†	rw	rw	rw	rw	

† ***mdata*** is read only when ***mtxen*** = 0.

Table A–21. SIO (Serial Interface I/O Register)

Bit	Name	Description
7	***nmrst***	MII NOT reset. The state of this bit directly controls the state of the **MRESET** pin (MII Reset). ❑ ***nmrst*** = 1. **MRESET** is asserted high. ❑ ***nmrst*** = 0. **MRESET** is asserted low. This bit is not self-clearing and must be deasserted manually. It can be set low, then immediately set high. Because every PHY attached to the MII may not have a reset pin, you need to do ***nmrst*** and also individually reset each PHY. The ***mdioen*** bit must be set to drive **MRESET**.
6	***mclk***	MII SIO clock. This bit controls the state of the **MDCLK** pin. ❑ ***mclk*** = 1. **MDCLK** is asserted high. ❑ ***mclk*** = 0. **MDCLK** is asserted low. The ***mdioen*** bit must be set to drive **MDCLK**.
5	***mtxen***	MII SIO transmit enable. This bit is used with the ***mdata*** bit to read/write information from/to the **MDIO** pin. ❑ ***mtxen*** = 1. **MDIO** is driven with the value in the ***mdata*** bit. ❑ ***mtxen*** = 0. ***mdata*** is loaded with the value on the **MDIO** pin. The ***mdioen*** bit must be set to drive **MDIO**.
4	***mdata***	MII SIO data. This bit is used with ***mtxen*** to read/write information from/to the **MDIO** pin. ❑ ***mtxen*** = 1. **MDIO** is driven with the value in this bit. ❑ ***mtxen*** = 0. This bit is loaded with the value on the **MDIO** pin. The ***mdioen*** bit must be set to drive **MDIO**.
3	***mdioen***	MII SIO data pin enable. This bit enables the high-impedance control of the **MDIO**, **MDCLK**, and **MRESET** pins. ❑ ***mdioen*** = 1. The MII SIO interface is enabled. ❑ ***mdioen*** = 0. The MII SIO interface is in a high-impedance state.

Table A–21. SIO (Serial Interface I/O Register) (Continued)

Bit	Name	Description
2	***eclk***	EEPROM SIO clock. This bit controls the state of the **ECLK** pin. ❑ ***eclk*** = 1. **ECLK** is asserted high. ❑ ***eclk*** = 0. **ECLK** is asserted low. Also, this bit is used to indicate if no EEPROM is found. If no EEPROM is detected when an EEPROM load is attempted, this bit is held at 0 and cannot be set to a 1. When this condition occurs, setting ***stop*** in ***SysControl*** unlocks this bit, allowing software to interrogate the EEPROM. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **ECLK** pin.
1	***etxen***	EEPROM SIO transmit enable. This bit controls the direction of the **EDIO** pin. ❑ ***etxen*** = 1. **EDIO** is driven with the value in the ***edata*** bit. ❑ ***etxen*** = 0. ***edata*** is loaded with the value on **EDIO**. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **EDIO** pin.
0	***edata***	EEPROM SIO data. This bit is used to read or write the state of the **EDIO** pin. ❑ ***etxen*** = 1. **EDIO** is driven with the value in this bit. ❑ ***etxen*** = 0. This bit is loaded with the value on **EDIO**. When the ***load*** bit in ***SysControl*** is set to 1, the EEPROM load-state machine controls the **EDIO** pin.

DevCode (Device Code Register) @ 0x00A3

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***devcode[7:0]***								
Reset Value	0	0	0	0	0	1	0	0	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table A–22. DevCode (Device Code Register)

Bit	Name	Description
7:0	***devcode***	Device code. Set to 0x04 for all revisions of this device.

DevNode (Device Node Register) @ 0x00A4–0x00A9

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	–	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	r	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[24:31]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	–	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–23. DevNode (Device Node Register)

Bit	Name	Description
47:0	***address***	Device node address. This register is used to specify the 48-bit address that is used by the MAC Control sublayer as: ❑ the source address in transmitted pause frames, and ❑ one of the two valid destination addresses for pause frames (the other address being the reserved multicast address 01.80.C2.00.00.01). This register is not used for any other purpose. The address is stored in native Ethernet format. Bit 40 of this address, bit 0 of this register, is hardwired to 0 to prevent a multicast address from being entered. If the value contained in this register is desired to be used as the destination address for data frames, for example to cause frames to be routed to the NM port, this value must be added explicitly to the IALE's address records via a DIO add, or by learning from a frame received by the NM port. Until this is done, IALE treats this address as an unknown unicast address, i.e., the contents of this register are not implicitly known by the IALE.

McastLimit (Transmit Queue Multicast Limit Register) @ 0x00DC–0x00DF (DIO)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***enable[2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***threshold [2:0]***			reserved													
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	rwl	rwl	rwl	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–24. McastLimit (Transmit Queue Mcast Limit Register)

Bit	Name	Description
31:29	***threshold***	Multicast limit threshold. This value is used to calculate the number of queued multicast frames any port can have queued for transmission. The limit is calculated as $2^{(n+1)}$ multicast frames (where n = ***threshold***). ***threshold [2:0]*** — **Multicast Frame Limit** 000 — 2 001 — 4 010 — 8 011 — 16 100 — 32 101 — 64 110 — 128 111 — 256 Each port maintains a count of the number of outstanding multicast frames queued. When this count exceeds the limit set, and the multicast-limit function is enabled, subsequent multicast frames are not queued to that port. BPDU multicast frames recognized by the IALE are not unaffected.
28:10	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
9:3	reserved	Reserved. These bits may be read and written. These bits have no effect.
2:0	***enable***	Multicast limit enable portvector. The ***enable*** portvector determines the transmit ports that will take part in transmit-queue multicast limiting.

RAMControl (RAM Control Register) @ 0x00E2

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***bsel[3:0]***				***rsel[3:0]***				
Reset Value	0	0	0	1	0	1	0	0	
Reset Type	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–25. RAMControl (RAM Control Register)

Bit	Name	Description
7:4	***bsel***	RDRAM bus select. Determines the sampling time for RDRAM RSL outputs. It should not be modified unless explicitly requested by TI.
3:0	***rsel***	RDRAM read select. Determines the valid data window timing of RDRAM RSL inputs. It should not be modified unless explicitly requested by TI.

RAMStatus (RAM Status Register) @ 0x00E3

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved						***parity***	***rdinit***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	h	h	
Field Type	r	r	r	r	r	r	r	r	

Table A–26. RAMStatus (RAM Status Register)

Bit	Name	Description
7:2	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
1	***parity***	RDRAM parity enabled. This bit indicates whether the RDRAM supports data protection via setting and checking parity: ❑ ***parity*** = 1. The RDRAMs in the system have 9-bit bytes, and data is provided by the setting and checking of parity. ❑ ***parity*** = 0. The RDRAMs in the system have 8-bit bytes; therefore, no data protection is provided. This bit is not valid unless ***rdinit*** is set.
0	***rdinit***	RDRAM initialized. This bit becomes set when the RDRAM initialization sequence (which occurs after a hard reset) completes successfully.

PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s)) @ 0x00EA–0x00EB

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***pausetime[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–27. PauseTime100 (IEEE Std 802.3x Pause Interval (100-Mbit/s))

Bit	Name	Description
15:0	***pausetime***	IEEE Std 802.3x pause_time. This value defines the 16-bit unsigned integer containing the length of time for which the receiving station is requested to inhibit data-frame transmission. The pause time is measured in units of pause quanta, equal to 512-bit times of the particular implementation. The range of possible pause time is 0 to 65535 pause quanta. If this value is set to 0, pause frames never are issued, even if flow control is enabled and the switch is congested. Writing a value between 0x1 and 0x3F causes nondeterministic flow-control operation. The default value of 0x1 is for device testing; it needs to be initialized by the host CPU or by EEPROM load to 0x0 on greater than 0x3F.

PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s)) @ 0x00EE–0x00EF

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***pausetime[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–28. PauseTime1000 (IEEE Std 802.3x Pause Interval (1000-Mbit/s))

Bit	Name	Description
15:0	***pausetime***	IEEE Std 802.3x pause_time. This value defines the 16-bit unsigned integer containing the length of time for which the receiving station is requested to inhibit data-frame transmission. The pause time is measured in units of pause quanta, equal to 512-bit times of the particular implementation. The range of possible pause time is 0 to 65535 pause quanta. If this value is set to 0, pause frames never are issued, even if flow control is enabled and the switch is congested. Writing a value between 0x1 and 0x3F causes nondeterministic flow-control operation. The default value of 0x1 is for device testing; it needs to be initialized by the host CPU or by EEPROM load to 0x0 on greater than 0x3F.

FlowThreshold (Flow Control Threshold Register) @ 0x00F0–0x00F2

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***flowthreshold[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved								
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table A–29. FlowThreshold (Flow Control Threshold Register)

Bit	Name	Description
23:16	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
15:0	***flowthreshold***	Flow-control threshold value. The ***flow*** bit must be set to 1 in ***SysControl*** to enable this feature. ❑ When the number of free buffers in external memory that are available for frame reception drops below this value, flow control will be initiated in the ports (both Ethernet and NMPORT). ❑ When the number of free buffers in external memory that are available for frame reception drops below half of this value, both **Mxx_FLOW** pins are asserted. The **Mxx_FLOW** pins cease to be asserted when the free-buffer count equals or exceeds one-half the flowthreshold value, the MAC/NMPORT flow control ceases when the free-buffer count equals or exceeds the flowthreshold value. Depending on the amount of memory in the system, it may be possible to program this value to be greater than the total number of buffers in the system, thereby permanently inducing flow control.

LEDControl (LED Control Register) @ 0x00F4–0x00F5

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved	***txais***	***fltled***	***slast***	***swled[11:0]***												
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–30. LEDControl (LED Control Register)

Bit	Name	Description
15	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
14	***txais***	Transmit activity indication select. Allows the transmit content of the port-activity LEDs to be controlled: ❑ ***txais*** = 1. The port-activity indication includes transmit multicast traffic. ❑ ***txais*** = 0. The port-activity indication does not include transmit multicast traffic. Receive traffic indication is not dependent on this bit value or the type of frame being received.
13	***fltled***	Fault LED. Controls the fault LED as follows: ❑ ***fltled*** = 1. The Fault LED is turned on. ❑ ***fltled*** = 0. The Fault LED is on only if: ■ An invalid CRC was detected during the EEPROM load. This condition is cleared by hardware reset or by setting the ***load*** bit to 1 in ***SysControl***, or ■ A parity error has occurred in the external memory system. This condition is cleared by hardware reset or by setting the ***start*** bit to 1 in ***SysControl***.
12	***slast***	Software LEDs last. Determines the position of the software-controlled LEDs (as controlled by ***swled*** and ***pmled***) in the shift-out sequence. ❑ ***slast*** = 1. The software LEDs are shifted out after the port LEDs. ❑ ***slast*** = 0. The software LEDs are shifted out before the port LEDs.
11:0	***swled***	Software controlled LEDs. These 12 LEDs are available for general software control. ❑ ***swled*** = 1. The corresponding LED is turned on. ❑ ***swled*** = 0. The corresponding LED is turned off. There is no additional hardware control of these LEDs. The writing of these bits is not synchronized to the display of the LEDs. It is possible for the LED interface to read these 12 bits between a write to the least significant byte and a write to the most significant byte of this register (or vice versa). If a quantity being displayed on the LEDs straddles the byte boundary, they may appear on the LEDs 1/16 of a second apart.

StatControl (Statistics Control Register) @ 0x00F8–0x00F9

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved					***stmap***	***long***	***bigend***	***clra***	***clrp***	reserved				***portcode [1:0]***		
Reset Value	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	
Reset Type	–	–	–	–	–	h	h	h	h	h	–	–	–	–	h	h	
Field Type	r	r	r	r	r	rwl	rwl	rwl	rwl	rwl	r	r	r	r	rwl	rwl	

Table A–31. StatControl (Statistics Control Register)

Bit	Name	Description
15:11	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
10	***stmap***	Statistic mapping. Selects which statistic is recorded in multiple-function statistic counters (currently only Portx Tx Data Errors). ❑ ***stmap*** = 1. The number of frames that have been sent to the transmit queue are recorded. ❑ ***stmap*** = 0. The number of data errors at transmit are recorded.
9	***long***	Long frame handling. Determines how statistics are kept on frames greater than 1518 bytes in length. If ***long*** = 0 or ***maxlen*** = 1 in the port's ***PortxControl*** register, the following statistics do not record frames if they are over 1518 bytes long: ❑ Portx Good Rx frames ❑ Portx Broadcast Rx frames ❑ Portx Multicast Rx frames ❑ Portx Pause Rx frames ❑ Portx Rx CRC errors ❑ Portx Broadcast Rx frames ❑ Portx 1024–1518 octet frames (Rx frames only) ❑ Portx Rx octets The following statistics record frames that are more than 1518 bytes long: ❑ Portx Rx jabbers ❑ Portx Oversized Rx frames If ***long*** = 1 and ***maxlen*** = 0 in the port's ***PortxControl*** register, the boundary at which the above statistics stop or start to record frames moves from 1518 bytes to 1531 bytes. The ***long*** bit has no effect on any Tx statistics. This bit has no effect on the longest frame size that will be forwarded. The ***long*** bit solely affects the statistics.

Table A–31. StatControl (Statistics Control Register) (Continued)

Bit	Name	Description
8	***bigend***	Big endian. Determines the order in which bytes within each 32-bit counter are read: ❑ ***bigend*** = 1. A byte address of 0 references the numerically most significant byte of a counter, and a byte address of 3 references the numerically least significant byte of a counter. ❑ ***bigend*** = 0. A byte address of 0 references the numerically least significant byte of a counter, and a byte address of 3 references the numerically most significant byte of a counter. This bit affects accesses to DIO addresses only in the range 0x8000 and above.
7	***clra***	Clear address lookup statistics. Writing a 1 to this bit causes all address-lookup statistics (DIO addresses in the range 0xA000–0xA00B) to be cleared. (Writing a 0 has no effect.) This bit then becomes write protected and autoclears when the statistics clear is complete. As the statistics are maintained in RAM, and updated in a cyclic manner, clearing is not instantaneous. After a hardware reset, this bit is set to 1 so that all address-lookup statistics are cleared upon power up. This does not occur following a stop.
6	***clrp***	Clear port statistics. Writing a 1 to this bit causes all statistics counters associated with the port indicated by ***portcode*** to be cleared. (Writing a 0 has no effect.) This bit then becomes write protected and autoclears when the statistics clear is complete. As the statistics are maintained in RAM and updated in a cyclic manner, clearing is not instantaneous. After a hardware reset, this bit is set to 1 (and portcode will be all 1s) so that all port statistics are cleared upon power up. This does not occur following a stop.
5:2	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
1:0	***portcode***	Clear port code. When ***clrp*** is set to 1, this field indicates which port's statistics counters are to be cleared. If ***portcode*** is all 1s, all ports are cleared. If an unimplemented port is specified in this field and ***clrp*** is set to 1, ***clrp*** eventually clears without clearing any port statistics.

SysControl (System Control Register) @ 0x00FA–0x00FB

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***stop***	***load***	***start***	***initd***	***reboot***	***holb***	reserved	***cnotify***	***nage***	***dmainc***	reserved	***unkvlan***	***mirr***	***nauto***	***ncrc***	***flow***	Byte Address Offset 0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	hs	h	h	–	h	h	h	–	h	h	h	h	h	
Field Type	rw	rw	rwl	r	rwl	rwl	r	rwl	rwl	rwl	r	rwl	rwl	rwl	rwl	rwl	

Table A–32. SysControl (System Control Register)

Bit	Name	Description
15	***stop***	Stop system. When ***stop*** = 1, device operation is halted, and remains so until this bit is cleared to 0. All internal state machines, FIFOs, and MACs are reset. Register bits are cleared as per the register definition tables in this section. The host registers are not affected. Any data in the device is lost. ❏ This bit is set to 1 if an EEPROM load fails because of a bad CRC word. ❏ This bit is set to 1 if a parity error is detected on the external memory interface.
14	***load***	Load system. Writing a 1 to this bit causes the DIO registers to be autoloaded from an external EEPROM (if present). Writing a 0 to this bit has no effect. Writing a 1 to this bit also clears the CRC-error indication from the fault LED. All DIO operations are inhibited (**SRDY** is held inactive high) while ***load*** is a 1 once the EEPROM state machine has finished checking that an EEPROM is present and begins to download data. ***load*** clears to 0 once the download is completed. When this bit is written with a 1, the ***stop*** bit simultaneously should be written with a 0.
13	***start***	Start system. Writing a 1 to this bit causes the device to begin operation, following a reset or stop. This bit is read as a 1 until buffer and address-lookup memory initialization is complete. This causes any previous frames or address records to be erased. While memory is being initialized, all ports are disabled. All DIO writes are inhibited (**SRDY** inactive high) while ***start*** is a 1. Writing a 1 to this bit also clears the parity-error indication from the fault LED. Writing a 0 to this bit has no effect, as does writing a 1 when ***initd*** = 1. When this bit is written with a 1, the ***stop*** bit simultaneously should be written with a 0.
12	***initd***	Initialization done. This bit becomes a 1 after ***start*** has been set, and buffer and address-lookup memory initialization is complete (i.e., at the same time ***start*** is cleared). Once set, it remains so until a hard reset (pin or DIO), or until ***stop*** is set to 1. Until this bit is a 1, the ports ignore any frames they receive.
11	***reboot***	Reboot. When set to 1, this bit enables unmanaged systems to recover from fatal errors without manual intervention. ❏ ***reboot*** = 0. A fault causes ***stop*** to be set. ❏ ***reboot*** = 1. A fault causes a hard reset, thereby allowing the device configuration to be automatically reloaded from EEPROM, and the device to be restarted. A fault occurs if a RDRAM parity error is detected or any of a variety of internal state consistency checks fails.

Table A–32. SysControl (System Control Register) (Continued)

Bit	Name	Description
10	***holb***	Head of line blocking. When set, ***holb*** is enabled. This function prevents switch congestion on a channel from consuming all the memory resources, causing other channels to be affected. ❑ If ***flow*** is disabled with ***holb*** enabled, the switch discards frames after the ***holb*** threshold has been exceeded (with the exception of the NM port). ❑ If ***flow*** is enabled with ***holb***, the switch attempts to preserve frames already in the fabric and prevent further submission on RX ports, contributing to the congestion. The ***FLOW*** LED illuminates to indicate that a flow request was made. Where it is not possible to use flow control, frames are discarded from the Rx FIFO and counted as an overrun. When the NM port enters ***holb***, the ***NumFreeBufs*** field in ***NmRxControl***, is held at zero after the next ***eof*** bit write, until the condition clears, preventing further frame submission from the host computer. No frames are discarded that are received from the NM port. Whenever the ***holb*** threshold is exceeded and the ***flow*** bit is set, the appropriate ***Mxx_FLOW*** pin on the port which entered ***holb*** is asserted.
9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***cnotify***	Change notification. Determines what action is taken when a known source address is received on a port other than the one currently associated with the address: ❑ ***cnotify*** = 1. Change notification is enabled. The frame additionally is forwarded to the ports specified by ***UnkSrcPorts*** ANDed with the appropriate ***VLANnPorts*** register, provided that address learning is enabled for the port to which the address has moved (appropriate bit = 0 in ***NLearnPorts***). This allows the address-lookup tables of other switches to be updated in accordance with the change. ❑ ***cnotify*** = 0. Change notification is disabled.
7	***nage***	No aging. Allows automatic address-record aging to be disabled independent of learning. (***nauto*** = 1disables both learning and aging). ❑ ***nage*** = 1. Aging is disabled even if ***nauto*** = 0. ❑ ***nage*** = 0. Aging is enabled unless ***nauto*** = 1. If ***nauto*** = 1, the value of ***nage*** is irrelevant – aging is disabled. This bit has no effect on address learning or time-stamp updating.
6	***dmainc***	DMA address autoincrement. When accessing the DIO interface with a DMA controller (**SDMA** pin low), this bit determines whether the address held in ***dmaaddress*** should be incremented between successive accesses. ❑ ***dmainc*** = 1. The address increments between accesses. ❑ ***dmainc*** = 0. The address does not increment between accesses.
5	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
4	***unkvlan***	Unknown VLAN forwarding. This bit enables forwarding of multicast frames that have an unknown IEEE Std 802.1q VLAN Identifier (i.e., one that matches none of the VLAN IDs registered in ***VLAN0QID***–***VLAN63QID***). ❑ ***unkvlan*** = 1. All multicast frames belonging to unknown VLANs are forwarded to the port indicated in ***UnkVLANPort***. ❑ ***unkvlan*** = 0. All multicast frames belonging to unknown VLANs are discarded.

Table A–32. SysControl (System Control Register) (Continued)

Bit	Name	Description
3	***mirr***	Port mirroring. This bit enables port mirroring. ❑ ***mirr*** = 1. All frames received on or sent to the port specified in ***MirrorPort*** are copied to the port specified in ***UplinkPort.*** ❑ ***mirr*** = 0. No mirroring is performed.
2	***nauto***	Not automatically add address mode. This bit selects the manner in which addresses are added to the address records. ❑ ***nauto*** = 1. Addresses can be added only to records using DIO adds. The aging-state machine will be disabled; it is the external CPU's responsibility to manage the address records. ❑ ***nauto*** = 0. New addresses are learned from the wire and automatically added to the address tables. (They also can be added using DIO adds.) However, if ***nage*** = 1 and the address tables become full, further addresses are not added unless space is created first using DIO deletes.
1	***ncrc***	No CRC check. This bit determines whether addresses are learned if the frame containing them contains an invalid CRC. ❑ ***ncrc*** = 1. Unknown addresses are added to the address-lookup table without regard to the validity of the CRC. ❑ ***ncrc*** = 0. Unknown addresses are added to the address-lookup table only if the frame containing the address has a valid CRC.
0	***flow***	Flow control enable. This bit enables flow control. ❑ ***flow*** = 1. The device implements flow control. Ports in full duplex that are configured for IEEE Std 802.3x use pause frames. Those in full duplex not configured for pause frames will discard packets. Ports in half duplex use collisions. ❑ ***flow*** = 0. No flow control is implemented.

VLAN0Ports–VLAN63Ports (VLAN Routing Register) @ 0x0100–0x01FF

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0100 + 0x4*n
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	rwl	rwl	rwl	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	DIO Address 0x0102 + 0x4*n
Field Name	reserved																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–33. VLAN0Ports–VLAN63Ports (VLAN Routing Register)

Bit	Name	Description
31:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2:0	***portvector***	VLAN port vector. Frames with multicast destination addresses that are associated with VLAN index n are forwarded to the ports determined by logically ANDing the port vector associated with the address in the address-lookup table with this field in ***VLANnPorts***. If the address is unknown, ***VLANnPorts*** are ANDed with ***UnkUniPorts*** or ***UnkMultiPorts***, depending on whether the address is unicast or multicast, to determine the ports to which the frame is forwarded. Destination addresses are associated with a particular VLAN index in one of several ways: ❑ If the frame contains an IEEE Std 802.1q VLAN ID that matches the entry in ***VLANnQID***, the frame is associated with VLAN index *n.* ❑ If the frame contains no IEEE Std 802.1q VLAN ID or contains a VLAN ID of 0x000, the frame is associated with the VLAN in the source port's ***PortxQTag*** register.

VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register) @ 0x0300–0x037F

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0300 + 0x2*n
Field Name	reserved				***vlanqid[11:0]***												
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Value†	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	–	–	–	–	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

† ***VLAN0QID*** is initialized to 000000000001; All other ***VLANxQID*** registers are initialized to 000000000000.

Table A–34. VLAN0QID–VLAN63QID (IEEE Std 802.1q VLAN Identifier Register)

Bit	Name	Description
15:12	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
11:0	***vlanqid***	IEEE Std 802.1q VLAN identifier. Specifies the IEEE Std 802.1q VLAN identifier associated with each of the device's VLANs. The IEEE Std 802.1q ID associated with a received frame is converted to an internal VLAN index using these registers. Frames are associated with an IEEE Std 802.1q VLAN ID in one of two ways, depending on the value of the ***rxacc*** bit at the ingress ***PortxControl*** register: ❑ If the frame contains an IEEE Std 802.1q VLAN ID, that ID is used. ❑ If the frame contains no IEEE Std 802.1q VLAN ID or has an ID of 0x000, the default source-port VLAN contained in ***PortxQTag*** is used. Unicast frames that contain an IEEE Std 802.1q VLAN ID that is not recognized are discarded. Multicast frames that contain an IEEE Std 802.1q ID that is not recognized are forwarded according to ***UnkVLANPort*** if ***unkvlan*** in ***SysControl*** = 1; otherwise, the frames are discarded. The least significant byte of a ***VLANxQID*** register must be written first because it goes into a holding latch. Only when the most significant byte is written is the whole 12-bit ***vlanqid*** entered into the ***VLANxQID*** register.

Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register) @ 0x0380–0x0383

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0380 + 0x2*x
Field Name	reserved				***vlanqid[11:0]***												
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	–	–	–	–	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	rwl	

Table A–35. Port0QTag–Port1QTag (Default IEEE Std 802.1q VLAN Tag Register)

Bit	Name	Description
15:12	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
11:0	***vlanqid***	IEEE Std 802.1q VLAN identifier. Identifies the default IEEE Std 802.1q VLAN ID associated with each port. How it is used depends on the value of ***rxacc*** in ***PortxControl***: ❑ ***rxacc*** = 1. An IEEE Std 802.1q tag header always is added to the frame (even if the frame already is tagged). The default source port VLAN ID contained in this field is used. ❑ ***rxacc*** = 0. If a received frame contains no IEEE Std 802.1q tag header, one is added using the default source port VLAN ID contained in this field. If the frame has a tag header with an ID of 0x000, the VLAN tag is replaced by this field.

PortxStatus (Port x Status Register) @ 0x0400–0x0403

Ports 0–1 are 100-/1000-Mbit/s ports. x = 0, 1.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	reserved							***pma***	***txpactv***	***nlink***	***pausetx***	***speed***	***duplex***	***pauserx***	***txqueue***	***linkchng***	DIO Address 0x0400 + 0x2*x
Reset Value	0	0	0	0	0	0	0	0	0	–	–	–	–	–	–	0	
Reset Type	–	–	–	–	–	–	–	–	hs	–	–	–	–	–	–	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rc	

Table A–36. PortxStatus (Port x Status Register)

Bit	Name	Description
15:9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***pma***	PMA mode. This bit indicates whether the on-chip PCS layer is enabled. ❑ ***pma*** = 1. It indicates enabled. ❑ ***pma*** = 0. It indicates disabled. The value of this bit reflects the inverse of the **$\overline{\text{Mxx\_PMA}}$** pin.
7	***txpactv***	Transmit pause active. When this bit is 1, the port has received a pause frame and its pause-time period is being observed. The port does not start transmitting a new data frame while this bit is a 1. Any frame in midtransmission when this bit becomes a 1 is transmitted fully. The transmission of pause frames is unaffected by this bit's value.
6	***nlink***	No link. This bit indicates the link state of the network port. ❑ ***nlink*** = 1. The link is inactive. ❑ ***nlink*** = 0. The link is active. The value of this bit is determined in one of two ways: ❑ ***pma*** = 1. This bit reports the inverse of ***lkstatus*** in ***PCSxStatus***. ❑ ***pma*** = 0. This bit reports the inverse of the state of the port's **Mxx_LINK** pin.

Table A–36. PortxStatus (Port x Status Register) (Continued)

Bit	Name	Description
5	***pausetx***	Transmit pause frame support. This bit indicates whether the port can generate IEEE Std 802.3x pause frames. ❑ ***pausetx*** = 1. IEEE Std 802.3x pause frames can be generated, provided ***flow*** = 1 in ***SysControl***. ❑ ***pausetx*** = 0. IEEE Std 802.3x pause frames cannot be generated. The value of this bit is determined in one of several ways. ❑ ***speed*** = 1 and ***pma*** = 0. This bit reflects the inverse of ***reqntxp*** in ***PortxControl***. ❑ ***speed*** = 0. This bit reflects the continuously sampled value of the **Mxx_RXD5** pin. ❑ ***pma*** = 1. This bit and ***pauserx*** are set according to either the capabilities advertised (***neg*** = 0 in ***PortxControl***) or negotiated (***neg*** = 1) by the PCS. ■ When ***neg*** = 1, this is determined by ANDing ***pause*** and ***asym*** in ***PCSxANAdvert*** with ***lppause*** and ***lpasym***, respectively, in ***PCSxANLinkP***. The mapping is as follows: ***pause*** AND ***lppause*** / ***asym*** AND ***lpasym*** → ***pausetx*** / ***pauserx*** 0 0 → 0 0 0 1 → 1 0 1 0 → 1 1 1 1 → 0 1 ■ When ***neg*** = 0, this is determined by ***pause*** and ***asym*** only: ***pause*** / ***asym*** → ***pausetx*** / ***pauserx*** 0 0 → 0 0 0 1 → 1 0 1 0 → 1 1 1 1 → 0 1
4	***speed***	Network speed. This bit indicates the speed of the network port: ❑ ***speed*** = 1. Indicates 1000 Mbit/s ❑ ***speed*** = 0. Indicates 100 Mbit/s The value of this bit is determined in one of two ways. ❑ ***pma*** = 1. This bit is 1. ❑ ***pma*** = 0. This bit reflects the value of the **$\overline{\text{Mxx\_MII}}$** pin.
3	***duplex***	Full duplex network. This bit indicates the duplex state of the network port. ❑ ***duplex*** = 1. Indicates full duplex ❑ ***duplex*** = 0. Indicates half duplex The value of this bit is determined in one of several ways: ❑ ***speed*** = 1 and ***pma*** = 0. This bit reflects the inverse of ***reqhd*** in ***PortxControl***. ❑ ***speed*** = 0. This bit reflects the continuously sampled value of the **Mxx_RXD4** pin. ❑ ***pma*** = 1. This bit reflects the value of ***fulldpx*** in ***PCSxExtStatus***.

Table A–36. PortxStatus (Port x Status Register) (Continued)

Bit	Name	Description
2	***pauserx***	Pause-frame Rx support. This bit indicates whether the port can respond to IEEE Std 802.3x pause frames. In 100-Mbit/s mode (***speed*** = 0), this bit is unused and always reads 0. In 1000-Mbit/s mode (***speed*** = 1), this bit indicates whether the port will respond to IEEE Std 802.3x pause frames. ❑ ***pauserx*** = 1. IEEE Std 802.3x pause frames are responded to. ❑ ***pauserx*** = 0. IEEE Std 802.3x pause frames are ignored and absorbed by the MAC. The value of this bit is determined in one of several ways: ❑ ***speed*** = 1 and ***pma*** = 0. This bit reflects the inverse of ***reqnrxp*** in ***PortxControl***. ❑ ***speed*** = 0. This bit reflects the continuously sampled value of the **Mxx_RXD5** pin. ❑ ***pma*** = 1. This bit and ***pausetx*** are set according to the capabilities negotiated by the PCS layer. Refer to the ***pausetx*** description above.
1	***txqueue***	Transmit data queued. This bit indicates whether any frame data remains to be transmitted on this port. ❑ ***txqueue*** = 1. The port has untransmitted frame data queued. ❑ ***txqueue*** = 0. The transmit queue is empty.
0	***linkchng***	Link change. This bit is set to a 1 if the port's ***nlink*** bit changes from a 1 to a 0 or from a 0 to a 1, indicating that the link has gone down or up, respectively. This bit is cleared to 0 when this register is read. If an ***nlink*** change attempts to set the bit at the same time as a read tries to clear it, the bit is set. This bit is logically ORed with the ***linkchng*** bits of all the ***PortxStatus*** registers to produce the ***link*** interrupt bit in the ***Int*** register.

FindNode (Search Node Address Register) @ 0x0440–0x0445

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***address[39:32]***								***address[47:40]***								0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset
Field Name	***address[23:16]***								***address[31:24]***								0x2
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset
Field Name	***address[7:0]***								***address[15:8]***								0x4
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table A–37. FindNode (Search Node Address Register)

Bit	Name	Description
47:0	***address***	Find node address. Used to exchange addresses between the address-lookup *Find* state machine and the external CPU. The node address in ***FindNode*** is kept in Ethernet's native format. The function of ***FindNode*** depends on the bits set in ***FindControl***. When the *Find* state machine is operating, this register is locked, and writes to it have no effect.

FindControl (Search Control Register) @ 0x0446

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***found***	***new***	***node***	***port***	***vlan***	***first***	reserved	***find***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	–	hs	
Field Type	r	rw	rw	rw	rw	rw	r	rw	

Table A–38. FindControl (Search Control Register)

Bit	Name	Description
7	***found***	Address found. If the find operation initiated by setting ***find*** is successful, this bit is set upon completion. Any data returned in ***FindVLAN***, ***FindNode***, or ***FindPort*** is valid only when this bit is a 1.
6	***new***	New address finds. When asserted, the address-lookup table is scanned for the first address that is marked as new.
5	***node***	Find by address. When asserted, the address indicated by ***address*** in ***FindNode*** is included in the match criteria.
4	***port***	Find by port. When asserted, the port indicated by ***xportcode*** in ***FindPort*** is included in the match criteria. Searches with this bit set are restricted to unicast addresses.
3	***vlan***	Find by VLAN index. When asserted, the port indicated by ***vlanindex*** in ***FindVLAN*** is included in the match criteria.
2	***first***	Lookup first address. Determines the point in the address-lookup table from which the search begins. ❑ ***first*** = 1. The search starts from the first location in the address-lookup table. ❑ ***first*** = 0. The search starts from the location in the address-lookup table immediately following that at which the previous search completed.
1	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
0	***find***	Address lookup. When asserted, the find operation is initiated. The type of search is determined by the value of ***node***, ***vlan***, ***port***, ***new,*** and ***first***. When the find is active, this register is locked, and writes to it have no effect. Upon completion of the search, this bit self-clears, and the register can be written again.

FindVLAN (Search VLAN Index Register) @ 0x0447

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table A–39. FindVLAN (Search VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	Find VLAN index. This value points to an actual VLAN tag via the ***VLAN Routing*** Registers. This field is used in two ways, depending on the type of search. ❑ ***vlan*** = 1 in ***FindControl***. The VLAN index to use as part of the matching criteria should be written to this field before setting ***find*** in ***FindControl***. ❑ ***vlan*** = 0 in ***FindControl***. This field returns the VLAN index for the node matching the specified search criteria. When a find is active, this register is locked, and writes to it have no effect.

FindPort (Search Routing Code Register) @ 0x0448–0x044B – Unicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***nodeage[7:0]***								reserved		***xportcode[5:0]***						Byte Address Offset 0x0
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Field Name	***nblck***	***secure***	***locked***	***cuplnk***	***new***	reserved			***nodeage[15:8]***								Byte Address Offset 0x2
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–40. FindPort (Search Routing Code Register) – Unicast Format

Bit	Name	Description
31	***nblck***	Not blocked indication. This bit shows whether the ***RxUniBlockPorts*** register is used with this address. ❑ ***nblck*** = 1. ***RxUniBlockPorts*** is not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxUniBlockPorts*** bit for the source port also is not set.
30	***secure***	Secured address indication. This bit shows the security level for the address in ***FindNode.*** Secure addresses are not aged out and cannot move between ports. If a secured address attempts to move from one port to another, a security violation interrupt is given to the external CPU. The address record of the node is not altered if this occurs, i.e., it is not locked and still can transmit successfully on the port in the address record.
29	***locked***	Locked address indication. This bit shows the lock status for the address in ***FindNode***. Any frames received that have a locked address in either the source or destination field are discarded. No security violation is issued. A host CPU must set/clear this bit via an add (edit) operation.
28	***cuplnk***	Copy frames to uplink indication. This bit shows the Copy Uplink status for the address in ***FindNode***. Addresses tagged with this bit add the port specified in the ***UplinkPort*** register to the routing code.

Table A–40. FindPort (Search Routing Code Register) – Unicast Format (Continued)

Bit	Name	Description
27	***new***	New address indication. This bit shows that the node contained in ***FindNode*** has been added or had its port changed since it was last reported by a DIO find. All DIO finds clear the new bit in the address record after an address is found.
26:24	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
23:8	***nodeage***	Address age stamp. Contains the age time stamp of the address in ***FindNode***.
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Current extended port for node. This field is used in two ways, depending on the type of search. ❏ ***port*** = 1 in ***FindControl***. The port to use as part of the matching criteria should be written to this field before setting ***find*** in ***FindControl***. ❏ ***port*** = 0 in ***FindControl***. This field returns the port for the node matching the specified search criteria. During searches, this field is locked, and writes to it have no effect.

FindPort (Search Routing Code Register) @ 0x0448–0x044B – Multicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	reserved	***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–41. FindPort (Search Routing Code Register) – Multicast Format

Bit	Name	Description
31	***nblck***	Not blocked indication. This bit shows whether the ***RxMultiBlockPorts*** register is used with this address. ❏ ***nblck*** = 1. ***RxMultiBlockPorts*** is not used to filter frames. ❏ ***nblck*** = 0. Frames are forwarded only if appropriate ***RxMultiBlockPorts*** bit for the source port also is not set.
30	reserved	Reserved. This bit is used in unicast format.
29:24	***xroutecode***	Extended route code multicast. This field shows the route code used to generate the preframe tag added to frames routed to port 1 when ***pretag*** = 1 in ***Port1Control***.
23:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:3	reserved	Reserved for multicast. These bits are defined in unicast format.
2:0	***portvector***	Port bit vector for multicast. This field shows the port bit vector for the multicast address in ***FindNode***. The bit values in this field correspond one to one with the port assignments.

NewNode (New Address Register) @ 0x044C–0x0451

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–42. NewNode (New Address Register)

Bit	Name	Description
47:0	***address***	New node address. Used with ***Int*** and ***IntEnable*** to pass information to the external CPU. This field is valid on a ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt. The node address in ***NewNode*** is kept in Ethernet's native format. To avoid the data in this field being changed if another interrupt occurs while reading the current data, the field is locked until the most significant byte of ***NewVLAN*** is read (DIO address 0x0457).

NewPort (New Address Port Register) @ 0x0454–0x0455

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved		***newxportcode[5:0]***						reserved		***oldxportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–43. NewPort (New Address Port Register)

Bit	Name	Description
15:14	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
13:8	***newxportcode***	Extended port for address in ***NewNode***. This field shows the assigned port number for the address in ***NewNode***.
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***oldxportcode***	Old extended port for address in ***NewNode***. ❑ When a node port-change interrupt is asserted (***chng*** = 1 in ***Int***), this field contains the old port location for the address. If ***nauto*** = 1, this is still true, but the address record is not changed. ❑ When a security violation interrupt is asserted (***secvio*** = 1 in ***Int***), this field shows the port to which the node attempted to move. Any other events that write to ***NewPort*** cause this field to be set to 0.

NewVLAN Interrupt (New Address VLAN Register) @ 0x0456–0x0457, unkvlan Int Definition

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved				***vlanqid[11:0]***												
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–44. NewVLAN Interrupt (New Address VLAN Register)

Bit	Name	Description
15:12	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
11:0	***vlanqid***	IEEE Std 802.1q VLAN identifier for address in ***NewNode***. Upon a ***vlan*** interrupt, this field shows the IEEE Std 802.1q VLAN identifier for the address in ***NewNode***.

NewVLAN Other Interrupts (New Address VLAN Register) @ 0x0456–0x0457

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved										***vlanqid[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–45. NewVLAN Other Interrupts (New Address VLAN Register)

Bit	Name	Description
15:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	VLAN index for address in ***NewNode***. This field shows the VLAN index for the address in ***NewNode***.

AddNode (Address Register) @ 0x0458–0x045D

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table A–46. AddNode (Address Register)

Bit	Name	Description
47:0	***address***	Add node address. The unicast or multicast address in this register is added to the address-lookup table when ***add*** in ***AddDelControl*** is set to 1. The node address in ***AddNode*** is kept in Ethernet's native format. When the *Add* state machine is operating, this field is locked, and writes to it have no effect.

AddDelControl (Add/Delete Control Register) @ 0x045E

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved				***delp***	***delv***	***add***	***del***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	hs	hs	hs	hs	
Field Type	r	r	r	r	rw	rw	rw	rw	

Table A–47. AddDelControl (Add/Delete Control Register)

Bit	Name	Description
7:4	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
3	***delp***	Delete addresses on port. When set to 1, all unicast address records whose port assignment matches ***DelPort*** are deleted unless they are marked as secure or locked. Multicast addresses are not deleted. This bit remains a 1 until the delete process is complete, when it self-clears. During this period ***delp***, ***delv***, and ***del*** are write protected. This bit can be asserted at the same time as ***delv***, but not if ***delv*** is already set. This bit can be used with ***delv*** to delete all nonlocked, nonsecure, unicast address records on a particular port with a particular VLAN.
2	***delv***	Delete addresses on VLAN. When set to 1, all unicast address records whose VLAN assignment matches ***DelVLAN*** are deleted unless they are marked as secure or locked. Multicast addresses are not deleted. This bit remains a 1 until the delete process is complete, when it self-clears. During this period ***delp***, ***delv***, and ***del*** are write protected. This bit can be asserted at the same time as ***delp***, but not if ***delp*** is already set. This bit can be used with ***delp*** to delete all nonlocked, nonsecure, unicast address records on a particular port with a particular VLAN.
1	***add***	Add address. When set to 1, the information in ***AddPort***, ***AddNode***, and **AddVLAN** is used to add or edit an entry in the address records.This bit remains a 1 until the add process is complete, when it self-clears. During this period, **add** is write protected.
0	***del***	Delete address. When set to 1, the information in ***DelNode*** and ***DelVLAN*** is used to delete an entry from address records. Any address records can be deleted this way, including multicast addresses and secure and/or locked unicast addresses. This bit remains a 1 until the delete process is complete, when it self-clears. During this period, ***delp***, ***delv***, and ***del*** are write protected. This bit cannot be set to 1 if ***delp*** or ***delv*** also is being set to 1. If such an attempt is made, none of these bits is set to 1 and no delete is performed.

AddVLAN (Add Address VLAN Index Register) @ 0x045F

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table A–48. AddVLAN (Add Address VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	Add VLAN index. This is used with the ***AddNode*** and ***AddPort*** registers to add/edit a node's VLAN assignment. When the Add state machine is operating, this field is locked and writes to it have no effect.

AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Unicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved										***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	***secure***	***locked***	***cuplnk***	***new***	reserved											
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	

Table A–49. AddPort (Add Routing Code Register) – Unicast Format

Bit	Name	Description
31	***nblck***	Not-blocked flag. This bit sets the ***RxUniBlockPorts*** override function. ❑ ***nblck*** = 1. ***RxUniBlockPorts*** are not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxUniBlockPorts*** bit for the source port is not set. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
30	***secure***	Secured-address flag. This bit determines the security level for the address contained in ***AddNode***. When an Add is in progress, this bit is locked against modification and writes to it have no effect.
29	***locked***	Locked-address flag. This bit determines the lock status for the address in ***AddNode***. Any frames received from a locked-source address are discarded. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
28	***cuplnk***	Copy-frames-to-uplink flag. This bit determines the Copy Uplink status for the address contained in ***AddNode***. Addresses tagged with this bit add the port specified in the ***UplinkPort*** register to the routing code. When an Add is in progress, this bit is locked against modification and writes to it have no effect.
27	***new***	New-address flag. This bit determines the new status for the address contained in ***AddNode***. Having set ***new*** to 1, this address easily is found by a *Find* command, which has ***new*** in ***FindControl*** = 1. When an Add is in progress, this bit is locked against modification and writes to it have no effect.

Table A–49. AddPort (Add Routing Code Register) – Unicast Format (Continued)

Bit	Name	Description
26:24	reserved	Reserved. These bits are used in the multicast format.
23:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Current extended port for node. This field determines the destination port for the unicast address shown in ***AddNode***. If a nonexistent port is specified, no address record is entered when the Add is performed. (The Add will complete in the normal manner but no record will exist.) When an Add is in progress, this bit is locked against modification, and writes to it have no effect.

AddPort (Add Routing Code Register) @ 0x0460–0x0463 – Multicast Format

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved													***portvector [2:0]***			
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***nblck***	reserved	***xroutecode[5:0]***						reserved								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	–	–	–	–	–	–	–	–	
Field Type	r	r	rw	rw	rw	rw	rw	rw	r	r	r	r	r	r	r	r	

Table A–50. AddPort (Add Routing Code Register) – Multicast Format

Bit	Name	Description
31	***nblck***	Not blocked flag. This bit sets the ***RxMultiBlockPorts*** override function. ❑ ***nblck*** = 1. ***RxMultiBlockPorts*** is not used to filter frames. ❑ ***nblck*** = 0. Frames are forwarded only if the appropriate ***RxMultiBlockPorts*** bit for the source port is not set. When an Add is in progress, this bit is locked against modification, and writes to it have no effect.
30	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
29:24	***xroutecode***	Extended route code for multicast. This field determines the route code used to generate the preframe tag added to frames routed to port 1 when pretagging is enabled. This field is ignored if pretagging is disabled or both bit 0 and bit 1 of ***portvector*** are 0. Locked against modification while an Add is in progress.
23:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:3	reserved	Reserved. These bits are used in the unicast format.
2:0	***portvector***	Port bit vector for multicast. This field determines the port bit vector for the multicast address contained in ***AddNode***. The bit values in this field correspond one-to-one with the port assignment. When an Add is in progress, this bit is locked against modification.

AgedNode (Aged Out Address Register) @ 0x0464–0x0469

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–51. AgedNode (Aged Out Address Register)

Bit	Name	Description
47:0	***address***	Aged node address. This register is used with the ***Int*** and ***IntEnable*** registers to pass information to the external CPU about addresses that have been deleted from the address-lookup table due to the aging process. This field is valid on an ***age*** interrupt. The node address in ***AgedNode*** is kept in Ethernet's native format. To avoid the data in this field being changed if another interrupt occurs while reading the current data, the field is locked until ***AgedVLAN*** is read.

AgedPort (Aged Out Address Port Assignment Register) @ 0x046A

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	

Table A–52. AgedPort (Aged Out Address Port Assignment Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Extended port for aged node. This register displays the assigned port for the deleted address contained in ***AgedNode.*** This byte remains locked against further modification by address aging until the ***AgedVLAN*** register has been read.

AgedVLAN (Aged Out Address VLAN Index Register) @ 0x046B

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	h	h	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	

Table A–53. AgedVLAN (Aged Out Address VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	VLAN index for aged node. This register displays the assigned VLAN index for the deleted address contained in ***AgedNode.*** When this byte is read, all the Age registers, i.e., ***AgedNode***, ***AgedPort***, and ***AgedVLAN***, are unlocked and free to record the next address record to be aged.

DelNode (Delete Address Register) @ 0x046C–0x0471

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***address[39:32]***								***address[47:40]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***address[23:16]***								***address[31:24]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	Byte Address Offset 0x4
Field Name	***address[7:0]***								***address[15:8]***								
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table A–54. DelNode (Delete Address Register)

Bit	Name	Description
47:0	***address***	Delete node address. The unicast or multicast address in this register is deleted from the address-lookup table when the ***del*** bit in ***AddDelControl*** is set to 1. The node address in ***DelNode*** is kept in Ethernet's native format. When the *Delete* state machine is operating, this field is locked, and writes to it have no effect.

DelPort (Delete Port Register) @ 0x0472

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***xportcode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table A–55. DelPort (Delete Port Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***xportcode***	Extended delete port. When ***delp*** in ***AddDelControl*** is set, all addresses associated with this port are deleted. If ***delv*** also is a 1, the address records are further qualified by the VLAN specified in ***DelVLAN***. Once set, ***delp*** is locked until the delete completes, and writes to it have no effect.

DelVLAN (Delete VLAN Index Register) @ 0x0473

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	reserved		***vlanindex[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	h	h	h	h	h	h	
Field Type	r	r	rw	rw	rw	rw	rw	rw	

Table A–56. DelVLAN (Delete VLAN Index Register)

Bit	Name	Description
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5:0	***vlanindex***	Delete VLAN index. ❑ If the ***del*** bit in ***AddDelControl*** is set, this field is used with the ***DelNode*** and ***DelPort*** registers to delete a node from the address-lookup table. ❑ When the ***del*** and ***delv*** bits in ***AddDelControl*** are both set, all addresses associated with this VLAN index are deleted. In this case, the ***DelNode*** and ***DelPort*** registers are don't care. When the Delete state machine is performing a Delete or Delete VLAN operation, this field is locked, and writes to it have no effect.

NumNodes (Node Count Register) @ 0x0474–0x0475

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***numnodes[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–57. NumNodes (Node Count Register)

Bit	Name	Description
15:0	***numnodes***	Number of nodes. Contains the number of addresses currently in the address-lookup table. This register always should be read least significant byte first; reading this location causes the register contents to be transferred to a holding latch to prevent the value changing while it is being read.

AgingCounter (Age Out Time Count Register) @ 0x0476–0x0477

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***count[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–58. AgingCounter (Age Out Time Count Register)

Bit	Name	Description
15:0	***count***	Aging counter. This register allows the current value of the AgingCounter to be observed. This can be used with the ***nodeage*** field returned in ***FindPort*** when the aging process is not being managed automatically, i.e., if ***nauto*** and/or ***nage*** = 1 in ***SysControl***. This register always should be read least significant byte first; reading this location causes the register contents to be transferred to a holding latch to prevent the value changing while it is being read. This register is cleared to 0 by hardware reset. It is frozen at its current value when the ***stop*** bit in ***SysControl*** is set to 1. Setting the ***start*** bit to 1 in ***SysControl*** clears this counter to 0 and it is held at 0 until IALE's initialization has completed, when it increments to 1. After this time, the counter increments according to the selected aging algorithm. See the ***AgingThreshold*** register description for details of how the incrementing mode of this register is selected.

XMultigroup17–XMultigroup63 (Extended Multicast Group Register) @ 0x0544–0x05FF

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address Offset 0x0540 + 4*n
Field Name	***xportvector[15:0]***																
Reset Value	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	DIO Address Offset 0x0542 + 4*n
Field Name	reserved															***xportvector***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rw	

Table A–59. XMultigroup17–XMultigroup63 (Extended Multicast Group Register)

Bit	Name	Description
31:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16:0	***xportvector***	Extended multicast group port vector. This field determines the extended port vector to be inserted into the preframe tag added to frames routed to port 1 when ***pretag*** = 1 in ***Port1Control***. ***XMultiGroupn*** is associated with ***xroutecode*** = *n*, where *n* is in the range 17–63. If ***xroutecode*** is less than 17, it is interpreted as a unicast extended port vector, and only bit *n* of the vector is set.

PCSxControl (Port x PCS Control Register) @ 0x0600–0x0601, 0x0620–0x0621

Ports 0–1 are 100-/1000-Mbit/s ports. x = 0, 1.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0600 +0x20*x
Field Name	***pcsreset***	***loopback***	***speed1***	***enaneg***	reserved		***newaneg***	***pcsdp***	***control test***	***speed0***	reserved						
Reset Value	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	
Reset Type	hs	h	–	h	–	–	h	h	h	–	–	–	–	–	–	–	
Field Type	rw	rw	r	rw	r	r	rw	rw	rw	r	r	r	r	r	r	r	

Table A–60. PCSxControl (Port x PCS Control Register)

Bit	Name	Description
15	***pcsreset***	PCS software reset. When a 1 is written to ***pcsreset***, a single cycle pulse is generated, which causes the PCS to enter a software reset state. All state machines and registers internal to the PCS will be reset as per a hard reset (as indicated in the register definition tables) with the exception of the most significant byte of ***PCSxControl***, which takes on the value written at that time. Writes to PCS registers when ***pcsreset*** = 1 will still take effect. This bit reads as 1 when the PCS is disabled (***pma*** = 0 in ***PortxStatus***). However, zero-to-one transitions on ***pcsreset*** not caused by a DIO write (for example by clearing ***pma*** or setting ***stop***) do *not* cause the PCS registers to assume their reset state. When PCS is enabled, this bit automatically returns to 0 when reset is complete. Writing 0 has no effect.
14	***loopback***	Loopback. When high, this port is forced into full-duplex mode, so it can receive frames it transmits. The **Mxx_EWRAP** pin reflects the value of this bit, thereby enabling external wrap testing at the PMA device. In this mode, signal detect (**Mxx_LINK**) from the PMA device is ignored. Link will be asserted after the PCS has negotiated with itself.
13	***speed1***	Speed selection MSB. Combined with ***speed0***, this bit indicates the speed of operation of the PCS. The on-chip PCS is intended to run at only 1000 Mbit/s; consequently, this bit is hardwired as 0.
12	***enaneg***	Enable autonegotiation. This bit is an alternative location at which to read or write ***neg*** in ***PortxControl***. Please refer to the ***PortxControl*** description for details.
11:10	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
9	***newaneg***	Restart autonegotiation. When set high, the on-chip PCS restarts autonegotiation of the pause and duplex capabilities described by ***reqntxp***, ***reqhd***, and ***reqnrxp*** bits in ***PortxControl*** across the PMA interface. This bit is self-clearing and returns to 0 when autonegotiation is completed. Writing a 0 to this bit has no affect. This bit reads high when autonegotiation has been restarted by writing a 0, followed by a 1 to ***enaneg*** (***neg*** in ***PortxControl***). This always reads 0 if ***neg*** = 0 in ***PortxControl***.

Table A–60. PCSxControl (Port x PCS Control Register) (Continued)

Bit	Name	Description
8	***pcsdp***	Duplex mode. If ***neg*** = 0 in ***PortxControl*** (***enaneg*** in ***PCSxControl***), this bit determines whether the PCS should operate in half-duplex (***pcsdp*** = 0) or full-duplex (***pcsdp*** = 1) mode. This bit behaves as an alternative location at which to read or write ***reqhd*** in ***PortxControl*** and ***canfdp*** in ***PCSxANAdvert***. See the respective register descriptions for details. (The sense of ***reqhd*** is inverted with respect to ***pcsdp*** and ***canfdp***.) If ***neg*** = 1, the value of this bit is ignored by the PCS layer.
7	***coltest***	Collision test. This bit is used to enable a collision test mode within the PCS. ❏ ***coltest*** = 1. The PCS generates test collisions on transmit. ❏ ***coltest*** = 0. The PCS does not generate test collisions.
6	***speed0***	Speed selection LSB. Combined with ***speed1***, this bit indicates the speed of operation of the PCS. The on-chip PCS is intended to run at only 1000 Mbit/s; consequently, this bit is hardwired to 1.
5:0	reserved	Reserved. Writes to these bits have no effect. They always read as 0.

PCSxStatus (Port x PCS Status Register) @ 0x0602–0x0603, 0x0622–0x0623

x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***100bt4***	***fdx100***	***hdx1000***	***fd10***	***hd10***	***fdt2100***	***hdt2100***	***extstat***	reserved		***andone***	***rfault***	***aneg***	***lkstatus***	reserved	***extcap***	DIO Address 0x0602 +0x20*x
Reset Value	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	h	h	h	h	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–61. PCSxStatus (Port x PCS Status Register)

Bit	Name	Description
15	***100bt4***	100 BASE-T4. This port does not support 100BASE-T4; consequently, this bit always reads as 0.
14	***fdx100***	100 BASE-X full duplex. This port does not support 100BASE-X at the PMA layer; consequently, this bit always reads as 0.
13	***hdx100***	100 BASE-X full duplex. This port does not support 100BASE-X at the PMA layer; consequently, this bit always reads as 0.
12	***fd10***	10Mbit/s full duplex. This port does not support 10-Mbit/s operation; consequently, this bit always reads as 0.
11	***hd10***	10Mbit/s full duplex. This port does not support 10-Mbit/s operation; consequently, this bit always reads as 0.
10	***fdt2100***	100 Base-T2 full duplex. This port does not support 100BASE-T2; consequently, this bit always reads as 0.
9	***hdt2100***	100 Base-T2 half duplex. This port does not support 100BASE-T2; consequently, this bit always reads as 0.
8	***extstat***	Extended status information. Indicates extended status information resides in ***PCSxExtStatus***. This bit is hardwired to 1.
7:6	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
5	***andone***	Autonegotiation complete ❑ ***andone*** = 1. Autonegotiation process completed, the contents of registers ***PCSxANAdvert, PCSxANLinkP,*** and ***PCSxANExp*** are valid. ❑ ***andone*** = 0. Autonegotiation process not completed, the contents of registers ***PCSxANAdvert, PCSxANLinkP,*** and ***PCSxANExp*** are meaningless. This bit always reads as 0 if autonegotiation is disabled (***enaneg*** = 0 in ***PCSxControl*** register).

Table A–61. PCSxStatus (Port x PCS Status Register) (Continued)

Bit	Name	Description
4	***rfault***	Remote fault. ***rfault*** is set to 1 during autonegotiation if an error in the protocol is detected and negotiation is restarted. At all other times it reads as 0.
3	***aneg***	Autonegotiation ability. This bit indicates whether the PCS is capable of autonegotiation. ❏ ***aneg*** = 1. The PCS has the ability to autonegotiate. ❏ ***aneg*** = 0. The PCS lacks the ability to autonegotiate. This bit reflects the value of ***enaneg*** in ***PCSxControl***.
2	***lkstatus***	Link status. This bit indicates the status of the connection to the link partner. When set to 1, the PCS is reporting a good connection. ❏ ***lkstatus*** = 1. Link is up. ❏ ***lkstatus*** = 0. Link is down. If the PCS detects a loss of link, this bit is set to 0 and autonegotiation restarts if enabled (***aneg*** = 1). This bit remains 0 until a good link is re-established. If autonegotiation across the PMA interface is disabled, this bit always reads 1. If PCS is disabled, this bit reads as 0.
1	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
0	***extcap***	Extended register capabilities. This bit always reads as 0. The PCS does not support an extended register set.

PCSxANAdvert (Port x PCS Autonegotiation Advertisement Register) @ 0x0608–0x0609, 0x0628–0x0629

x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x0608 +0x20*x
Field Name	***nxtpg***	reserved	***rf1***	***rf2***	reserved			***asym***	***pause***	***canhdp***	***canfdp***	reserved					
Reset Value	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	
Reset Type	h	–	h	h	–	–	–	h	h	h	h	–	–	–	–	–	
Field Type	rw	r	rw	rw	r	r	r	rw	rw	rw	rw	r	r	r	r	r	

Table A–62. PCSxANAdvert (Port x PCS Autonegotiation Advertisement Register)

Bit	Name	Description
15	***nxtpg***	Next page. This bit is used during autonegotiation to indicate to the link partner that the PCS wishes to exchange next pages.
14	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
13	***rf1***	Remote fault MSB. Combined with ***rf2***, this bit is used to indicate and classify a remote fault condition to its link partner. Classification is per IEEE Std 802.3z clause 37.2.1.4.
12	***rf2***	Remote fault LSB. Combined with ***rf1***, this bit is used to indicate and classify a remote fault condition to its link partner. Classification is per IEEE Std 802.3z clause 37.2.1.4.
11:9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***asym***	Asymmetric pause. Combined with ***pause***, this bit is used to advertise the device's IEEE Std 802.3x pause-frame capability to its link partner. The interpretation of ***pause*** and ***asym*** is per IEEE Std 802.3z clause 37.2.1.3. ***asym*** and ***pause*** are alternative locations at which to read or write ***reqntxp*** and ***reqnrxp*** in ***PortxControl***, but they are encoded differently. Changes to ***asym*** and ***pause*** are reflected in ***reqntxp*** and ***reqnrxp***, and vice versa. The mapping between them is shown with the ***pause*** definition.
7	***pause***	Pause. Combined with ***asym***, this bit is used to advertise the device's IEEE Std 802.3x pause-frame capability to its link partner. The interpretation of ***pause*** and ***asym*** is per IEEE Std 802.3z clause 37.2.1.3. ***asym*** and ***pause*** are alternative locations at which to read or write ***reqntxp*** and ***reqnrxp*** in ***PortxControl***, but they are encoded differently. Changes to ***asym*** and ***pause*** are reflected in ***reqntxp*** and ***reqnrxp***, and vice versa. The mapping between them is: ***pause*** ***asym*** ↔ ***reqntxp*** ***reqnrxp*** 0 0 ↔ 1 1 0 1 ↔ 0 1 1 0 ↔ 0 0 1 1 ↔ 1 0

Table A–62. PCSxANAdvert (Port x PCS Autonegotiation Advertisement Register) (Continued)

Bit	Name	Description
6	***canhdp***	Half duplex. This bit is used to advertise to the link partner whether the PMA layer is capable of supporting half-duplex operation. ❑ ***canhdp*** = 1. The PCS can operate in half-duplex mode. ❑ ***canhdp*** = 0. The PCS cannot operate in half-duplex mode.
5	***canfdp***	Full duplex. This bit is used to advertise to the link partner whether the PMA layer is capable of supporting full-duplex operation. ❑ ***canfdp*** = 1. The PCS can operate in full-duplex mode. ❑ ***canfdp*** = 0. The PCS cannot operate in full-duplex mode. This bit behaves as an alternative location at which to read or write ***reqhd*** in ***PortxControl*** and ***pcsdp*** in ***PCSxControl***. See the respective register descriptions for details. (The sense of ***reqhd*** is inverted with respect to ***pcsdp*** and ***canfpd***.)
4:0	reserved	Reserved. Writes to these bits have no effect. They always read as 0.

PCSxANLinkP (Port x PCS Autonegotiation Link Partner Ability Register) @ 0x060A–0x060B, 0x062A–0x062B

Ports 0–1 are 100-/1000-Mbit/s ports. x = 0, 1.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***lpnxp***	***ack***	***lprf1***	***lprf2***	reserved			***lpasym***	***lppause***	***lphdp***	***lpfdp***	reserved					DIO Address 0x060A +0x20*
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	–	–	–	h	h	h	h	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–63. PCSxANLinkP (Port x PCS Autonegotiation Link Partner Ability Register)

Bit	Name	Description
15	***lpnxp***	Link partner next page. This bit indicates the link partner's intention to transmit a next page. ❑ ***lpnxp*** = 1. The link partner intends to send a next page. ❑ ***lpnxp*** = 0. The link partner does not intend to transmit subsequent pages.
14	***ack***	Acknowledge. This bit indicates that the link partner has successfully received this device's base page. ❑ ***ack*** = 1. The link partner has successfully received the base page. ❑ ***ack*** = 0. The link partner has failed to receive the base page.
13	***lprf1***	Link partner remote-fault MSB. Combined with ***lprf2***, this bit indicates and classifies a remote-fault condition detected by the link partner. Classification is per IEEE Std 802.3z clause 37.2.1.4.
12	***lprf2***	Link partner remote fault LSB. Combined with ***lprf1***, this bit indicates and classifies a remote-fault condition detected by the link partner. Classification is per IEEE Std 802.3z clause 37.2.1.4.
11:9	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
8	***lpasym***	Link partner asymmetric pause. Combined with ***lppause***, this bit indicates the link partner's IEEE Std 802.3x pause-frame capability and determines a suitable mode of operation using autonegotiation. The interpretation of ***lppause*** and ***lpasym*** is per IEEE Std 802.3z clause 37.2.1.3.
7	***lppause***	Link partner pause. Combined with ***lpasym***, this bit indicates the link partner's IEEE Std 802.3x pause-frame capability and determines a suitable mode of operation using autonegotiation. The interpretation of ***lppause*** and ***lpasym*** is per IEEE Std 802.3z clause 37.2.1.3.
6	***lphdp***	Link partner half duplex. This bit reflects the link partner's ability to support half-duplex operation. ❑ ***lphdp*** = 1. The link partner can operate in half-duplex mode. ❑ ***lphdp*** = 0. The link partner cannot operate in half-duplex mode.

Table A–63. PCSxANLinkP (Port x PCS Autonegotiation Link Partner Ability Register) (Continued)

Bit	Name	Description
5	***lpfdp***	Link partner full duplex. This bit reflects the link partner's ability to support full-duplex operation. ❑ ***lpfdp*** = 1. The PCS can operate in full-duplex mode. ❑ ***lpfdp*** = 0. The PCS cannot operate in full-duplex mode.
4:0	reserved	Reserved. Writes to these bits have no effect. They always read 0.

PCSxANExp (Port x PCS Autonegotiation Expansion Register) @ 0x060C–0x060D, 0x062C–0x062D

x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DIO Address 0x060C +0x20*x
Field Name	reserved													***cannxtpg***	***pagercd***	reserved	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	h	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	rc	r	

Table A–64. PCSxANExp (Port x PCS Autonegotiation Expansion Register)

Bit	Name	Description
15:3	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
2	***cannxtpg***	Next-page able. This bit is hardwired to 1, indicating that the device supports next pages.
1	***pagercd***	Page received. This bit is set to 1 when a new page has been received from the link partner and stored in the ***PCSxANLinkP*** or ***PCSxANLinkPNxt*** register. The value of ***pagercd*** is held until the register is read, then it is cleared to 0.
0	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.

PCSxANNxt (Port x PCS Autonegotiation Next Page Transmit Register) @ 0x060E–0x060F, 0x062E–0x062F

x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***nxtpgtx***	reserved	***msgpage***	***ack2***	***toggle***	***message[10:0]***											DIO Address 0x060E +0x20*x
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	–	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table A–65. PCSxANNxt (Port x PCS Autonegotiation Next Page Transmit Register)

Bit	Name	Description
15	***nxtpgtx***	Next page to transmit. This bit indicates whether or not this is the last page to be transmitted. ❏ ***nxtpgtx*** = 1. This is not the last page; another page will follow. ❏ ***nxtpgtx*** = 0. This is the last page.
14	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
13	***msgpage***	Message page indicator. This bit indicates the context of the ***message*** field. ❏ ***msgpage*** = 1. ***message*** is a message code field. ❏ ***msgpage*** = 0. ***message*** is an unformatted code field.
12	***ack2***	Acknowledge2. This field is used by the next-page function to indicate this device has the ability to comply with the last received message. ❏ ***ack2*** = 1. This device can comply. ❏ ***ack2*** = 0. This device cannot comply.
11	***toggle***	Toggle. This bit ensures synchronization with the link partner. This bit should be toggled with each next page transmitted. See IEEE Std 802.3u clause 28.2.3.4.6 for further details of this protocol.
10:0	***message***	Message. This field is context sensitive. ❏ ***msgpage*** = 1. This field is a message-code field. ❏ ***msgpage*** = 0. This field is an unformatted-code field.

PCSxANLinkPNxt
(Port x PCS Autonegotiation Link Partner Next Page Received Register)
@ 0x0610–0x0611, 0x0630–0x0631

x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***nxtpgrx***	***ack***	***msgpage***	***ack2***	***toggle***	***message[10:0]***											DIO Address 0x0610 +0x20*x
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	–	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–66. PCSxANLinkPNxt
(Port x PCS Autonegotiation Link Partner Next Page Received Register)

Bit	Name	Description
15	***nxtpgrx***	Next page to receive. This bit indicates whether the link partner intends to transmit another page. ❑ ***nxtpgrx*** = 1. This is not the last page; another page will follow. ❑ ***nxtpgrx*** = 0. This is the last page.
14	***ack***	Acknowledge. This field is used by the next-page function to indicate the link partner has successfully received the last message transmitted. ❑ ***ack*** = 1. The link partner received the last page transmitted. ❑ ***ack*** = 0. The link partner has not received the last page transmitted.
13	***msgpage***	Message page indicator. This bit indicates the context of the ***message*** field. ❑ ***msgpage*** = 1. ***message*** is a message-code field. ❑ ***msgpage*** = 0. ***message*** is an unformatted-code field.
12	***ack2***	Acknowledge2. This field is used to indicate the link partner has the ability to comply with the last received message. ❑ ***ack2*** = 1. The link partner can comply. ❑ ***ack2*** = 0. The link partner cannot comply.
11	***toggle***	Toggle. This bit ensures synchronization with the link partner. This bit should be toggled with each next page transmitted. See IEEE Std 802.3u clause 28.2.3.4.6 for further details of this protocol.
10:0	***message***	Message. This field is context sensitive. ❑ ***msgpage*** = 1. This field is a message-code field. ❑ ***msgpage*** = 0. This field is an unformatted-code field.

PCSxExtStatus (Port x PCS Extended Status Register) @ 0x061E–0x061F, 0x063E–0x063F

x = 0, 1

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Field Name	***fulldpx***	***halfdpx***	***fulldpt***	***halfdpt***	reserved												DIO Address 0x061E +0x20*x
Reset Value	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Table A–67. PCSxExtStatus (Port x PCS Extended Status Register)

Bit	Name	Description
15	***fulldpx***	Full duplex 1000BASE-X. Hardwired to 1, indicating that the PCS can perform full-duplex operation for 1000BASE-X.
14	***halfdpx***	Half duplex 1000BASE-X. Hardwired to 1, indicating that the PCS can perform half-duplex operation for 1000BASE-X.
13	***fulldpt***	Full duplex 1000BASE-T. This device does not support 1000BASE-T; consequently, this bit always reads 0, writes have no effect.
12	***halfdpt***	Half duplex 1000BASE-T. This device does not support 1000BASE-T; consequently, this bit always reads 0, writes have no effect.
11:0	reserved	Reserved. Writes to these bits have no effect. They always read as 0.

DMAAddress (DMA Address Register) @ 0x0800–0x0801

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***dioaddr[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Table A–68. DMAAddress (DMA Address Register)

Bit	Name	Description
15:0	***dioaddr***	DIO address. When accessing the DIO interface with a DMA Controller (**$\overline{\text{SDMA}}$** pin low), this field determines the DIO address that is accessed (rather than the ***DIOAddrLo*** and ***DIOAddrHi*** DIO registers). If ***dmainc*** = 1 in ***SysControl***, this field will autoincrement after each access.

Int (Interrupt Register) @ 0x0804–0x0806

The ***Int*** register, in conjunction with the ***IntEnable*** register, provides interrupts to the attached CPU. When the **SINT** pin is asserted high, this register gives the reason for the interrupt. Specific interrupts can be masked by clearing the appropriate bit in ***IntEnable***. Bit setting by an interrupt source dominates over bit clearing by software if both are coincidental.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***new***	***newm***	***chng***	***chngm***	***secvio***	reserved	***age***	***agem***	***int***	***fndcplt***	***unkvlan***	***unkmem***	reserved	***nmtx***	***nmrx***	***full***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	h	h	h	h	–	h	h	h	
Field Type	rwc	rwc	rwc	rwc	rwc	r	rwc	rwc	rw	rwc	rwc	rwc	r	rwc	rwc	rwc	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved							***link***	
Reset Value	0	0	0	0	0	0	0	–	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table A–69. Int (Interrupt Register)

Bit	Name	Description
23:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16	***link***	Link-change interrupt. This read-only bit is the logical OR of all the ***linkchng*** bits of all the ***PortxStatus*** registers. If any of the ***linkchng*** bits are set, this interrupt will be issued if it has been enabled in ***IntEnable.***
15	***new***	New node interrupt. When 1, this bit indicates that a new node has been learned from the wire and added to the address records. If this interrupt is enabled via ***IntEnable***, the node address, port, and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until MSByte of the ***NewVLAN*** register is read by the CPU. DIO adds do not set this interrupt.
14	***newm***	Missed new node interrupt indication. When 1, this bit indicates that a new node interrupt was given, but the information was not placed in ***NewNode***, ***NewPort***, and ***NewVLAN*** because an earlier-enabled ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt locked them, and the information has not been read by the CPU. DIO adds do not set this interrupt.
13	***chng***	Node port change interrupt. When 1, this bit indicates that there has been a change in port assignment for a node that exists in the address records. If this interrupt is enabled via ***IntEnable***, the node address, port, and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until MSByte of ***NewVLAN*** register is read by the CPU. DIO adds do not set this interrupt.

Table A–69. Int (Interrupt Register) (Continued)

Bit	Name	Description
12	***chngm***	Missed node port change interrupt indication. When 1, this bit indicates that a node port-change interrupt was given, but the information was not placed in ***NewNode***, ***NewPort*,** and ***NewVLAN***, because an earlier-enabled ***new***, ***chng***, ***secvio***, ***unkvlan***, or ***unkmem*** interrupt locked them and the information has not been read by the CPU. DIO adds do not set this interrupt.
11	***secvio***	Security violation interrupt. If the ***NewNode***, ***NewPort***, and ***NewVLAN*** registers are not locked from a previous interrupt, this bit is set when a security violation occurs (i.e., when a node that has been secured has attempted to move port assignments). If this interrupt was enabled previously via ***IntEnable*** at the same time the ***secvio*** bit is set, the node's source address, port and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until MSByte of ***NewVLAN*** register is read by the CPU. (**SINT** also becomes active.) DIO adds do not set this interrupt.
10	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
9	***age***	Age-out interrupt. When 1, this bit indicates that a node has been aged-out (deleted from) the address-lookup table. The node address is written into ***AgedNode***, the node's assigned port is written into ***AgedPort***, and the node's VLAN is written into ***AgedVLAN.*** These registers are then locked against further modification until ***AgedVLAN*** is read by the CPU.
8	***agem***	Missed age-out interrupt indication. When 1, this bit indicates that an age-out interrupt was given, but the information was not placed in ***AgedNode***, ***AgedPort***, and ***AgedVLAN*** because an earlier ***age*** interrupt locked them, and the information has not been read by the CPU.
7	***int***	Test interrupt request. Setting this bit to 1 gives a test interrupt to the attached CPU. This bit is a normal read-write bit, unlike all other bits in this register.
6	***fndcplt***	Find completion interrupt. When 1, this bit indicates that the find state machine has completed a *Find First* or *Find Next* operation. This bit is not set for the Lookup operation. Data in the ***FindPort***, ***FindControl***, ***FindNode***, ***FindVLAN***, and ***FindNodeAge*** registers is now valid.
5	***unkvlan***	Unknown VLAN interrupt. If the ***NewNode***, ***NewPort***, and ***NewVLAN*** registers are not locked from a previous interrupt, this bit is set when an unknown IEEE Std 802.1q VLAN Identifier (i.e., one that matches none of the VLAN IDs registered in the ***VLANnQID*** registers) was received on a port whose corresponding ***UnkVLANIntPorts*** bit was 1. If this interrupt was enabled previously via ***IntEnable*** at the same time the ***unkvlan*** bit is set, the node's source address, port, and VLAN ID are written into ***NewNode*,** ***NewPort***, and ***NewVLAN***, respectively, and locked until MSByte of ***NewVLAN*** register is read by the CPU. (**SINT** also becomes active.)
4	***unkmem***	Unknown VLAN member interrupt. If the ***NewNode***, ***NewPort***, and ***NewVLAN*** registers are not locked from a previous interrupt, this bit is set when a frame is received on a port that is not a member of the VLAN associated with the frame, provided that the ***UnkVLANIntPorts*** and ***RxFilterPorts*** bits corresponding to the port is 1. If this interrupt was enabled previously via ***IntEnable*** at the same time the ***unkmem*** bit is set, the node's source address, port, and VLAN index are written into ***NewNode***, ***NewPort***, and ***NewVLAN***, respectively, and locked until MSByte of ***NewVLAN*** register is read by the CPU. (**SINT** also becomes active.)

Table A–69. Int (Interrupt Register) (Continued)

Bit	Name	Description
3	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
2	***nmtx***	Network management transmit interrupt. When 1, this bit indicates that the switch has a buffer of frame data ready to be read out (transmitted) to the external CPU via the DIO interface, having previously had no data ready. This is, when any of the ***eof***, ***sof***, or ***iof*** bits of ***NMTxControl*** become set to 1 after they were all previously 0, i.e., this interrupt is not reasserted when more than one buffer becomes ready. Reading ***NMTxControl*** reveals the buffer contents.
1	***nmrx***	Network management receive interrupt. When 1, this bit indicates that the switch has emptied its NM port receive buffer of frame data and is ready to receive a frame of any size up to 1535 bytes from the external CPU via the DIO interface.
0	***full***	Address-lookup table full interrupt. When 1, this bit indicates that an address has been added (either off-the-wire or via DIO) that caused the address records to become full. If the records are full and an attempt is made to add another address, this bit again is set to 1, assuming it had been cleared to 0 since the records became full.

The ***IntEnable*** register, with the ***Int*** register, selects the type of interrupts that should be given to the attached CPU. Bit definitions in ***IntEnable*** agree one to one with bit definitions in the ***Int*** register. Only those ***Int*** bits with the corresponding ***IntEnable*** bit set generate an interrupt to the CPU.

IntEnable (Interrupt Enable Register) @ 0x0808–0x080A

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***new***	***newm***	***chng***	***chngm***	***secvio***	reserved	***age***	***agem***	***int***	***fndcplt***	***unkvlan***	***unkmem***	reserved	***nmtx***	***nmrx***	***full***	
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	h	h	h	h	h	h	h	h	
Field Type	rw	rw	rw	rw	rw	r	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved							***link***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	h	
Field Type	r	r	r	r	r	r	r	rw	

Table A–70. IntEnable (Interrupt Enable Register)

Bit	Name	Description
23:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16	***link***	Link-change interrupt enable. When this bit is set to 1, a link-change interrupt is issued if ***link*** = 1 in ***Int***, i.e., if any of the ***linkchng*** bits in any ***PortxStatus*** register is set to 1.
15	***new***	New node enable. When this bit is 1, a new node interrupt is posted if ***new*** = 1 in ***Int***. If this bit is 0, ***NewNode***, ***NewPort***, and ***NewVLAN*** are not written or locked if ***new*** = 1 in ***Int***.
14	***newm***	Missed new node interrupt enable. When this bit is 1, a missed new node interrupt is posted if ***newm*** = 1 in ***Int***.
13	***chng***	Node port change interrupt enable. When this bit is 1, a node port change interrupt is posted if ***chng*** = 1 in ***Int***. If this bit is 0, then ***NewNode***, ***NewPort***, and ***NewVLAN*** are not written or locked if ***chng*** = 1 in ***Int***.
12	***chngm***	Missed node port change interrupt enable. When this bit is 1, a missed node port interrupt is posted if ***chngm*** = 1 in ***Int***.
11	***secvio***	Security violation interrupt enable. When this bit is 1, a security violation interrupt is posted if ***secvio*** = 1 in ***Int***. If this bit is 0, ***NewNode***, ***NewPort***, and ***NewVLAN*** are not written or locked if the ***secvio*** = 1 in ***Int***.

Table A–70. IntEnable (Interrupt Enable Register) (Continued)

Bit	Name	Description
10	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
9	***age***	Age-out interrupt enable. When this bit is 1, an age-out interrupt is posted if ***age*** = 1 in ***Int***.
8	***agem***	Missed age-out interrupt enable. When this bit is 1, a missed age-out interrupt is posted if ***agem*** = 1 in ***Int***.
7	***int***	Test interrupt enable. When this bit is 1, a test interrupt is posted if ***int*** = 1 in ***Int***.
6	***fndcplt***	Find completion enable. When this bit is 1, a find completion interrupt is posted if ***fndcplt*** = 1 in ***Int***.
5	***unkvlan***	Unknown VLAN enable. When this bit is 1, a new VLAN interrupt is posted if ***unkvlan*** = 1 in ***Int***. If this bit is 0, ***NewNode***, ***NewPort***, and ***NewVLAN*** are not written or locked if ***unkvlan*** = 1 in ***Int***.
4	***unkmem***	Unknown VLAN member enable. When this bit is 1, a new VLAN member interrupt is posted if ***unkmem*** = 1 in ***Int***. If this bit is 0, ***NewNode***, ***NewPort***, and ***NewVLAN*** are not written or locked if ***unkmem*** bit of ***Int*** becomes set to 1.
3	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
2	***nmtx***	Network management transmit interrupt enable. When this bit is set to 1, a management frame transmit interrupt is posted if ***nmtx*** = 1 in ***Int***.
1	***nmrx***	Network management receive interrupt enable. When this bit is set to 1, a management frame receive interrupt is posted if ***nmrx*** = 1 in ***Int***.
0	***full***	Address-lookup table full enable. When this bit is 1, an address-lookup table full interrupt is posted if ***full*** = 1 in ***Int***.

FreeStackLength (Free Stack Length Register) @ 0x080C–0x080E

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***freestacklength[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved								
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	r	r	r	r	r	r	r	r	

Table A–71. FreeStackLength (Free Stack Length Register)

Bit	Name	Description
23:16	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
15:0	***freestacklength***	Free stack length. This register contains the number of 1K-byte buffers currently on the free stack, i.e., the number of buffers not currently used for storing backlogged frames. The free stack itself consumes 1K bytes per 512K bytes of memory. Also, channel pointers for maintaining buffer strings are allocated from this space. It always should be read least significant byte first; reading this location causes the register contents to be transferred to a holding latch to prevent the value changing while it is being read.

SysTest (System Test Register) @ 0x080F

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***passed***	***bist[1:0]***		***rdwrite***	***rdram***	***dpwrap***	***intwrap[1:0]***		
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	
Field Type	r	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Table A–72. SysTest (System Test Register)

Bit	Name	Description
7	***passed***	BIST passed. A 1 in this bit indicates that the previously initiated internal RAM or CAM BIST passed. It is cleared automatically when 01 or 10 is written to ***bist***.
6:5	***bist***	Initiate BIST. Initiates the internal RAM BIST processes. Upon completion, this field self-clears, and the result of the test is recorded in ***passed***. The coding is: 00 No BIST 01 Enable BIST of all internal RAMs 10 Enable BIST of CAM 11 Reserved
4	***rdwrite***	RDRAM write. This bit determines the transfer direction when a 1 is written to ***rdram***: ❑ ***rdwrite*** = 1. The RDRAM transfer is a write. ❑ ***rdwrite*** = 0. The RDRAM transfer is a read.
3	***rdram***	RDRAM test access. Initiates a 128-byte transfer between an internal buffer and the external RDRAM. The transfer direction is specified by ***rdwrite***. The RDRAM page address is specified by ***ramaddress***. When the 128-byte transfer is complete, this bit is cleared by hardware, allowing completion to be determined by polling.
2	***dpwrap***	Duplex wrap test mode. When high, all ports are forced into full-duplex mode and active link is asserted, so all ports can receive frames they transmit, thus enabling external wrap testing at the PHY. If port 8 is in PMA mode (***pma*** = 1 in ***PortxControl***), this capability is controlled via loopback in ***PCSxControl***.
1:0	***intwrap***	Internal wrap test mode. Ports 00–01 internally wrap back with full duplex and link according to the two-bit coding: 00 No internal wrapping 01 All ports internally wrapped 10 All ports internally wrapped, except port 0 11 All ports internally wrapped, except port 1 The NM port never can wrap. ***intwrap*** and ***dpwrap*** should not both be set to one, otherwise unpredictable behavior can occur.

RAMAddress (RAM Address Register) @ 0x0810–0x0813

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***ramaddress[15:0]***																
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	h	
Field Type	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	***inc***	***ramsel[3:0]***				reserved	***ramaddress[25:16]***										
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	h	h	h	h	h	–	h	h	h	h	h	h	h	h	h	h	
Field Type	rwp	rwp	rwp	rwp	rwp	r	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Table A–73. RAMAddress (RAM Address Register)

Bit	Name	Description
31	***inc***	RAM address autoincrement. When set to 1, ***ramaddress*** autoincrements after each access to/from ***RAMData***. If ***ramaddress*** is pointing at a reserved address, the increment will not occur.
30:27	***ramsel***	RAM select. This field selects the RDRAM to be accessed. 0000 External RDRAM 0001–1111 Reserved
26	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
25:0	***ramaddress***	RAM address. Indicates the byte within external RDRAM to be accessed. ❑ ***RAMData***, in conjunction with ***rdram*** and ***rdwrite*** in ***SysTest***, initiates reads or writes to the RDRAM.

RAMData (RAM Data Register) @ 0x00814

Bit	7	6	5	4	3	2	1	0	DIO Address Offset
Field Name	***ramdata[7:0]***								
Reset Value	–	–	–	–	–	–	–	–	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	rwp	rwp	rwp	rwp	rwp	rwp	rwp	rwp	

Table A–74. RAMData (RAM Data Register)

Bit	Name	Description
7:0	***ramdata***	RAM test data. In conjunction with ***RAMAddress*** and ***SysTest***, performs test accesses to the external RDRAM memory. ❑ Before initiating an RDRAM write, the 128 bytes to be transferred to RDRAM should be written to this address. ❑ After initiating an RDRAM read and waiting for it to complete, the 128 bytes to be transferred from RDRAM should be read from this address. During reading or writing the 128-byte transfer area, only the lower seven bits of ***RAMAddress*** are used. RDRAM accesses can be performed only when ***start*** = 0 and ***initd*** = 0 in ***SysControl***.

NMRxControl (Network Management Receive Control Register) @ 0x0818–0x081A

The NM port is treated as Port 2 internally.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	reserved			***freebuffs[4:0]***					***crc***	***eof***	***alen***	reserved			***portcode[1:0]***		
Reset Value	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	hs	hs	hs	hs	hs	hs	hs	hs	–	–	–	hs	hs	
Field Type	r	r	r	r	r	r	r	r	rw	rw	rw	r	r	r	rw	rw	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x0
Field Name	reserved		***xroutecode[5:0]***						
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	hs	hs	hs	hs	hs	hs	
Field Type	r	r	rwh	rwh	rwh	rwh	rwh	rwh	

Table A–75. NMRxControl (Network Management Receive Control Register)

Bit	Name	Description
23:22	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
21:16	***xroutecode***	If ***portcode*** = 1 (output port = 1), and ***alen*** = 0 (override internal lookup results), and ***pretag*** = 1 in ***Port1Control*** (port 1 connected to a crossbar), then ***xroutecode*** completely determines the destination ports on the crossbar. Otherwise, ***xroutecode*** is not used.
15:13	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
12:8	***freebuffs***	Free buffers available. This field indicates the number of 64-byte buffers that can be written safely to ***NMData*** without risk of SRDY being held inactive high to hold off further writes. This value is valid only after each complete burst of 64 bytes has been written to ***NMData*** or after ***NMRxControl*** has been written with ***eof*** = 1, i.e., the value is undefined if a noninteger quantity of 64 bytes has been written to ***NMData*** without having also written a 1 to ***eof***. The maximum value for this field is 24, equivalent to 1536 bytes available. The **SRXRDY** pin is high if this field is 24; otherwise, it is low. The ***nmrx*** bit in the ***Int*** register is set to 1 when this field reaches 24, i.e., when SRXRDY transitions from low to high. If the NM port or the entire switch enters a flow-controlled situation, this register will appear to contain a zero to prevent the host from sending more data into the NM port. When the congestion clears, this register will reflect the actual number of free buffers available.

Table A–75. NMRxControl (Network Management Receive Control Register) (Continued)

Bit	Name	Description
7	***crc***	CRC generation. Determines whether the device should insert the CRC word into the received frame. Must be valid when ***eof*** is written with a 1 (ignored at all other times). ❑ ***crc*** = 1. The last four bytes of received frame data are replaced automatically with valid CRC for that frame. ❑ ***crc*** = 0. The received frame is expected to contain a valid CRC word.
6	***eof***	End of frame. This bit is set to 1 after all data for a frame have been written to ***NMData***. It clears to 0 when the frame has been received. Further writes to ***NMRxControl*** or ***NMData*** hold $\overline{\text{SRDY}}$ high until the frame has been received and this bit returns to 0. ***crc*** should be written with the desired value at the same time this bit is written with a 1.
5	***alen***	Address-lookup enable. Indicates how the frame to be written into ***NMData*** should be forwarded. Must be valid before any data for the frame is written into ***NMData***. ❑ ***alen*** = 1. The normal address lookup determines the destination of the frame. ❑ ***alen*** = 0. Portcode determines the destination for the frame.
4:2	reserved	Reserved. Writes to this bit have no effect. It always reads as 0.
1:0	***portcode***	Transmit port number. Indicates to which port the frame is to be sent if ***alen*** = 0 (ignored if ***alen*** = 1). Must be valid before any data for the frame is written into ***NMData***.

NMTxControl (Network Management Transmit Control Register) @ 0x081C–0x081D

The NM port is treated as Port 2 internally.

The **STXRDY** pin outputs a 1 if any of the ***eof***, ***sof***, or ***iof*** bits is set to 1; otherwise, it outputs 0.

The ***nmtx*** bit in the ***Int*** register is set to 1 when any of the ***eof***, ***sof***, ***iof*** bits become set to 1 after they were all previously 0, i.e., when **STXRDY** transitions from a 0 to a 1.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Byte Address Offset 0x0
Field Name	***bytes[7:0]***								***sof***	***eof***							
Context-Sensitive Field									1	x	***pfe***	reserved			***port-code [1:0]***		
									0	1	***pfe***	reserved					
									0	0	***pfe***	***iof***	reserved				
Reset Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset Type	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	hs	
Field Type	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	

Bit	23	22	21	20	19	18	17	16	Byte Address Offset 0x2
Field Name	reserved							***flush***	
Reset Value	0	0	0	0	0	0	0	0	
Reset Type	–	–	–	–	–	–	–	hs	
Field Type	r	r	r	r	r	r	r	rw	

Table A–76. NMTxControl (Network Management Transmit Control Register)

Bit	Name	Description
23:17	reserved	Reserved. Writes to these bits have no effect. They always read as 0.
16	***flush***	Flush frame. Setting this bit causes all bytes remaining in the frame to be discarded. This prevents the external CPU from reading frames it does not require. The discard process is not instantaneous; during the flush, this bit remains set at 1, and ***NMTxControl*** is write protected. This bit autoclears when the flush is complete. Writing 0 to this bit has no effect. This bit should not be set to 1 unless a frame has bytes unread, i.e., do not set this bit between frames or it may coincide with the arrival of the next frame and cause an undesired flush.
15:8	***bytes***	Byte count. This field indicates the number of bytes of frame data contained within the next 256 bytes read from ***NMData.*** If this data includes the end of frame, the count includes the 4-byte CRC. A value of 0 should be interpreted as 256 bytes. These bits are valid only if one or more of ***sof***, ***eof***, and ***iof*** is a 1.

Table A–76. NMTxControl (Network Management Transmit Control Register) (Continued)

Bit	Name	Description
7	***sof***	Start of frame. Indicates that the next 256 ***bytes*** to be read from ***NMData*** contain the start of a frame. If, having read the end-of-frame data from ***NMData***, ***sof*** then reads as 0, the management CPU can infer that no more frames are available.
6	***eof***	End of frame. If a 1, the end of frame is within the next 256 bytes. The ***bytes*** field indicates how many bytes should be read to complete the transmission of the frame.
5	***pfe***	Previous frame error. This bit is valid after the last byte of a frame has been read from ***NMData***. If the bit is a 1, the frame contained a CRC error or the TCI parity protection detected an error; if it is 0, the frame contained no errors. This bit remains valid until the first byte of data of the next frame is read from ***NMData***, a flush is initiated, or a reset occurs. After a flush is complete, it is 0.
4	***iof*** (***eof*** = 0, ***sof*** = 0)	Interior of frame. Valid only if ***eof*** = 0 and ***sof*** = 0. Indicates that the next 256 bytes to be read from ***NMData*** are in the interior of a frame, i.e., not the first or last 256 bytes, and are available to be read. If the external CPU is expecting to read either an interior-of-frame buffer or end-of-frame buffer, it can poll until either ***eof*** or ***iof*** is detected as a 1, then resume reading data from ***NMData.***
1:0	***portcode*** (***sof*** = 1)	Received port number. This field indicates the number of the port that received the frame. It should be interpreted only as ***portcode*** when ***sof*** is read as 1.
Various	reserved	Reserved. Writes to these bits have no effect. They always read as 0.

NMData (Network Management Data Register) @ 0x0820

Bit	7	6	5	4	3	2	1	0	Byte Address Offset
Field Name	***nmdata[7:0]***								
Reset Value	–	–	–	–	–	–	–	–	
Reset Type	–	–	–	–	–	–	–	–	
Field Type	rw	rw	rw	rw	rw	rw	rw	rw	

Table A–77. NMData (Network Management Data Register)

Bit	Name	Description
7:0	***nmdata***	Network management data. An external CPU can use this address to write data for transmission or read received data on the NM port (port 2). ❑ On transmit, ***NMTxControl*** should be read before reading from each 256 buffer via ***NMData*** to determine how many bytes are in the buffer. ❑ On receive, ***NMRxControl*** should be written before and after writing an entire frame to ***NMData.*** If the transmit FIFO for port 2 becomes full while writing to ***NMData***, the ***SRDY*** pin is held high to prevent data being lost. This should happen only in exceptional circumstances in which the RDRAM memory is full.

This register can be read by setting up the DIO interface to 0x0820 and repeatedly reading the ***DIOData*** register. The ***NMData*** register can also be read via DMA by leaving the ***DMAAddress*** register, without increment, pointed to the same address.

A.3 Statistic Descriptions

Note:

MAC frames are handled entirely in the port logic. No MAC frames are included in any statistic.

Good Rx Frames

The total number of good frames received on the port. A good frame is defined as:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was of length 64 to 1518† bytes inclusive, and
- had no CRC error, alignment error, or code error

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

Rx Octets

The total number of bytes in all good frames received on the port. A good frame is defined as:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was of length 64 to 1518† bytes inclusive, and
- had no CRC error, alignment error, or code error

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See IEEE Std 802.3 1996; section 5.2.2.1.14.

Multicast Rx Frames

The total number of good multicast frames received on the port. A good multicast frame is defined as:

- any data or MAC control frame destined for any multicast address other than 0xFFFFFFFFFFFF, and
- was of length 64 to 1518† bytes inclusive, and
- had no CRC error, alignment error, or code error

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See RFC1757 Ref. 1.4 etherStatsMulticastPkts.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Broadcast Rx Frames

The total number of good broadcast frames received on the port. A good broadcast frame is defined as:

- any data or MAC control frame destined for address 0xFFFFFFFFFFFF only, and
- was of length 64 to 1518† bytes inclusive, and
- had no CRC error, alignment error, or code error

See the *Rx Align/Code Errors and Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See RFC1757 Ref. 1.3 etherStatsBroadcastPkts.

Rx Align/Code Errors

The total number of frames received on the port that experienced an alignment error or code error. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was of length 64 to 1518† bytes inclusive, and
- had either an alignment error or a code error

Overruns have no effect upon this statistic.

An alignment error is defined to be:

- a frame containing an odd number of nibbles, and
- also fails the Frame Check Sequence test if the final nibble is ignored.

A code error is defined to be:

- a frame that has been discarded because the port's **Mxx_RXER** pin driven with a 1 for at least one bit-time's duration at any point during the frame's reception.

In the case of the NM port, this statistic records each received frame that had ***alen*** = 0 and was attempted to be sent to an unimplemented port by the ***portcode*** field.

See RFC1623 Ref. 2.1 dot3StatsAlignmentErrors.

RFC 1757 etherStatsCRCAlignErrors Ref. 1.5 can be calculated by summing *Rx Align/Code Errors* and *Rx CRC Errors*, which is on the following page.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Rx CRC Errors

The total number of frames received on the port that experienced a CRC error. Such a frame:

- ❏ was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❏ was of length 64 to 1518† bytes inclusive, and
- ❏ had a CRC error

Overruns have no effect upon this statistic.

A CRC error is defined to be:

- ❏ a frame containing an even number of nibbles, and
- ❏ fails the Frame Check Sequence test.

See RFC1623 Ref. 2.2 dot3StatsFCSErrors.

Rx Jabbers

The total number of jabber frames received on the port. A jabber frame is:

- ❏ any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❏ was greater than 1518† bytes long, and
- ❏ had a CRC error, an alignment error, or a code error

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See RFC1757 Ref. 1.9 etherStatsJabbers.

Rx Fragments

The total number of frame fragments received on the port. A frame fragment is defined to be:

- ❏ any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❏ was less than 64 bytes long, and
- ❏ had a CRC error, an alignment error, or a code error

See the *Rx AlignICode Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See RFC1757 Ref. 1.8 etherStatsFragments.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Oversize Rx Frames

The total number of oversized frames received on the port. An oversized frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was greater than 1518† bytes long, and
- had no CRC error, alignment error, or code error

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See RFC1757 Ref. 1.7 etherStatsOversizePkts.

Undersize Rx Frames

The total number of undersized frames received on the port. An undersized frame is defined as:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was less than 64 bytes long, and
- had no CRC error, alignment error, or code error

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See RFC1757 Ref. 1.6 etherStatsUndersizePkts.

Rx + Tx Frames 64

The total number of 64-byte frames received and transmitted on the port. Such a frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- did not experience late or excessive collisions, and
- was exactly 64 bytes long. (If the frame was being transmitted and experienced an underrun or carrier loss that resulted in a frame of this size being transmitted, the frame is recorded in this statistic.)

Error conditions, such as alignment errors, CRC errors, code errors, and overruns, do not affect the recording of frames in this statistic.

For Rx reference only, see RFC1757 Ref. 1.11 etherStatsPkts64Octets.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Rx + Tx Frames 65–127

The total number of frames of size 65 to 127 bytes received and transmitted on the port. Such a frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- did not experience late or excessive collisions, and
- was 65 to 127 bytes long. (If the frame was being transmitted and experienced an underrun or carrier loss that resulted in a frame of this size being transmitted, the frame is recorded in this statistic.)

Error conditions, such as alignment errors, CRC errors, code errors, and overruns, do not affect the recording of frames in this statistic.

For Rx reference only, see RFC1757 Ref. 1.12 etherStatsPkts65to127Octets.

Rx + Tx Frames 128–255

The total number of frames of size 128 to 255 bytes received and transmitted on the port. Such a frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- did not experience late or excessive collisions, and
- was 128 to 255 bytes long. (If the frame was being transmitted and experienced an underrun or carrier loss that resulted in a frame of this size being transmitted, the frame is recorded in this statistic.)

Error conditions, such as alignment errors, CRC errors, code errors, and overruns, do not affect the recording of frames in this statistic.

For Rx reference only, see RFC1757 Ref. 1.13 etherStatsPkts128to255Octets.

Rx + Tx Frames 256–511

The total number of frames of size 256 to 511 bytes received and transmitted on the port. Such a frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- did not experience late or excessive collisions, and
- was 256 to 511 bytes long. (If the frame was being transmitted and experienced an underrun or carrier loss that resulted in a frame of this size being transmitted, the frame is recorded in this statistic.)

Error conditions, such as alignment errors, CRC errors, code errors, and overruns, do not affect the recording of frames in this statistic.

For Rx reference only, see RFC1757 Ref. 1.14 etherStatsPkts256to511Octets.

Rx + Tx Frames 512–1023

The total number of frames of size 512 to 1023 bytes received and transmitted on the port. Such a frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- did not experience late or excessive collisions, and
- was 512 to 1023 bytes long. (If the frame was being transmitted and experienced an underrun or carrier loss that resulted in a frame of this size being transmitted, the frame is recorded in this statistic.)

Error conditions, such as alignment errors, CRC errors, code errors, and overruns, do not affect the recording of frames in this statistic.

For Rx reference only, see RFC1757 Ref. 1.15 etherStatsPkts512to1023Octets.

Rx + Tx Frames 1024–1518†

The total number of frames of size 1024 to 1518† bytes received and transmitted on the port. Such a frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- did not experience late or excessive collisions, and
- was 1024 to 1518† bytes long. (If the frame was being transmitted and experienced an underrun or carrier loss that resulted in a frame of this size being transmitted, the frame is recorded in this statistic.)

Error conditions, such as alignment errors, CRC errors, code errors, and overruns, do not affect the recording of frames in this statistic.

For Rx reference only, see RFC1757 Ref. 1.16 etherStatsPkts1024to1518Octets.

SQE Test Errors

A count of the number of times that the SQE TEST ERROR message is generated by the PLS sublayer for a particular port. The SQE TEST ERROR message is defined in section 7.2.2.2.4 of ANSI/IEEE Std 802.3-1985 and its generation is defined in section 7.2.4.6.

See RFC 1623 Ref. 2.5 dot3StatsSQETestErrors.

This statistic is not recorded for ports operating at 1000 Mbit/s.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Tx Octets

The total number of bytes in all good frames transmitted on the port. A good frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was 64 to 1535 bytes inclusive, and
- had no CRC error, no late or excessive collisions, no carrier loss, and no underrun.

See IEEE Std 802.3 1996; section 5.2.2.1.8.

Good Tx Frames

The total number of good frames transmitted on the port. A good frame is defined to be:

- any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was 64 to 1535 bytes inclusive, and
- had no CRC error, no late or excessive collisions, no carrier loss, and no underrun.

See IEEE Std 802.3 1996; section 5.2.2.1.2.

Multiple-Collision Tx Frames

The total number of frames transmitted on the port that experienced multiple collisions. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was 64 to 1535 bytes inclusive, and
- had no carrier loss and no underrun, and
- experienced 2 to 15 collisions before being transmitted successfully. None of the collisions were late.

CRC errors have no effect upon this statistic.

See RFC1623 Ref. 2.4 dot3StatsMultipleCollisionFrames.

Single-Collision Tx Frames

The total number of frames transmitted on the port that experienced exactly one collision. Such a frame:

- ❑ was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❑ was 64 to 1535 bytes inclusive, and
- ❑ had no carrier loss and no underrun, and
- ❑ experienced one collision before being successfully transmitted. The collision was not late.

CRC errors have no effect upon this statistic.

See RFC1623 Ref. 2.3 dot3StatsSingleCollisionFrames.

Deferred Tx Frames

The total number of frames transmitted on the port that first experienced deferment. Such a frame:

- ❑ was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❑ was 64 to 1535 bytes inclusive, and
- ❑ had no carrier loss and no underrun, and
- ❑ experienced no collisions before being successfully transmitted, and
- ❑ found the medium busy when transmission was first attempted, so had to wait.

CRC errors have no effect upon this statistic

See RFC1623 Ref. 2.6 dot3StatsDefferredTransmissions.

Carrier Sense Errors

The total number of frames that experienced carrier loss on the port. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was 64 to 1535 bytes inclusive, and
- had no underrun, and
- did not experience excessive or late collsions before being transmitted successfully, and
- the carrier-sense condition was lost or never asserted when attempting to transmit the frame. The statistic is incremented once per transmission attempt, even if the carrier-sense condition fluctuated during that attempt.

However, for a gigbit port, a frame that has fluctuations of the carrier sense before and including the first 512 bytes (slot time with tolerant bit not set in port control) will not be recorded as a port carrier sense error.

Also, it can be observed that CRS may fluctuate during the slot time, but CRS must be present for the gigabit port after the the slot time has elapsed for transmission to continue. That is, as soon as the CRS is lost after the slot time, the transmit frame is purged.

CRC errors have no effect upon this statistic.

See RFC1623 Ref. 2.10 dot3StatsCarderSenseErrors.

Excessive Collisions

The total number of frames for which transmission was abandoned due to excessive collisions. Such a frame:

- was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- was 64 to 1535 bytes inclusive, and
- had no carrier loss and no underrun, and
- experienced 16 collisions before abandoning all attempts at transmitting the frame. None of the collisions was late.

CRC errors have no effect upon this statistic.

See RFC1623 Ref. 2.8 dot3StatsExcessiveCollisions.

Late Collisions

The total number of frames on the port for which transmission was abandoned because they experienced a late collision. Such a frame:

- ❑ was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❑ was 65 to 1535 bytes inclusive, and
- ❑ had no carrier loss and no underrun, and
- ❑ experienced a collision later than 512 bit-times into the transmission. There may have been up to 15 previous (not 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 have no effect upon this statistic.

See RFC1623 Ref. 2.7 dot3StatsLateCollisions.

Multicast Tx Frames

The total number of good multicast frames transmitted on the port. A good multicast frame is defined to be:

- ❑ any data or MAC control frame destined for any multicast address other than 0xFFFFFFFFFFFF, and
- ❑ was of length 64 to 1535 bytes inclusive, and
- ❑ had no CRC error, no late or excessive collisions, no carrier loss, and no underrun.

See IEEE Std 802.3-1996; section 5.2.2.1.18.

Broadcast Tx Frames

The total number of good broadcast frames transmitted on the port. A good broadcast frame is defined to be:

- ❑ any data or MAC control frame destined for address 0xFFFFFFFFFFFF only, and
- ❑ was of length 64 to 1535 bytes inclusive, and
- ❑ had no CRC error, no late or excessive collisions, no carrier loss, and no underrun

See IEEE Std 802.3 1996; section 5.2.2.1.19.

Tx Data Errors/Tx Queued Frames

This statistic can record one of two different events, depending upon the ***stmap*** bit in ***StatControl*** The options are:

- ***stmap*** = 0. Transmit data errors.

  The total number of frames transmitted on the port that experienced data errors. Such a frame:

  - was any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
  - was of length 64 to 1535 bytes inclusive, and
  - had either a CRC error or an underrun.

A CRC error can occur only if the frame has been corrupted within the switch since frames received with CRC errors (or alignment errors) are discarded during reception.

An underrun can occur only if the switch fails keep the transmitting MAC supplied with frame data. Under normal circumstances this should never occur.

Pause frames cannot underrun or contain a CRC error because they are created in the transmitting MAC.

- ***stmap*** = 1. Frames submitted to port's transmit queue.

  The total number of frames ever submitted to the port's transmit queue. These may not yet have been transmitted as they may still be queued in external memory, or may be being moved to the MAC for transmission. Such frames are (or were):

  - any data or MAC control frames destined for any unicast, broadcast, or multicast address, and
  - of length 64 to 1535 bytes inclusive.

Because of the latency between frame reception and frame queuing, there is no coherency between this statistic and the various received frames statistics.

CRC errors, collisions, carrier loss, and underrun have no effect upon this statistic.

Filtered Rx Frames

The total number of frames received on the port that the IALE decided should be discarded. Such a frame:

- ❑ was any data frame (not MAC control frame) destined for any unicast, broadcast, or multicast address, and
- ❑ was of length 64 to 1518† bytes inclusive, and
- ❑ did not experience any CRC error, alignment error, code error, or overrun, and
- ❑ the IALE decided, based upon its address records and frame-routing algorithm, that the frame should be discarded. This will include frames discarded because of security violations.

MAC control frames are discarded in the MAC, so they cannot appear in this statistic.

For ports that support pretagging, if ***pretag*** = 1 in ***PortxControl*** and a frame is received with a directed format pretag, this statistic will not be incremented, even if the ***portvector*** within the pretag was all zeros.

Rx Overruns

The total number of frames received on the port that subsequently overran. An overrun frame is defined to be:

- ❑ any data or MAC control frame destined for any unicast, broadcast, or multicast address, and
- ❑ was any length (including <64 bytes and >1518† bytes), and
- ❑ the switch was unable to receive it because it did not have the resources (e.g., memory, Rx FIFO space, or bandwidth) to receive it.

CRC errors, alignment errors, and code errors have no effect upon this statistic.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Collisions

This statistic records the total number of times that the port was required to send a jam sequence. This occurs under two circumstances:

- When a transmit data or MAC control frame:
  - was destined for any unicast, broadcast, or multicast address, and
  - was 65 to 1535 bytes inclusive, and
  - had no carrier loss and no underrun, and
  - experienced a collision. A jam sequence is sent for every collision, so this statistic will increment on each occasion if a frame experiences multiple collisions.

CRC errors have no effect upon this statistic.

- When the port is in half-duplex mode, flow control is active, and a frame reception begins.

See RFC1757 Ref. 1.10 etherStatsCollisions.

Pause Tx Frames

This statistic indicates the number of IEEE Std 802.3x pause frames transmitted by the port.

Pause frames cannot underrun or contain a CRC error because they are created in the transmitting MAC, so these error conditions have no effect upon the statistic.

Because pause frames are transmitted only in full duplex, carrier loss and collisions have no effect upon this statistic.

Transmitted pause frames are always 64-byte multicast frames, so they will appear in the the *Tx Multicast Frames* and *64-Byte Frames* statistics.

See IEEE Std 802.3z; section 30.3.4.2.

Pause Rx Frames

The total number of IEEE Std 802.3x pause frames received by the port. Such a frame:

- ❏ was destined for the multicast address 01.80.C2.00.00.01 (hex) or the unicast address in ***DevNode***, and
- ❏ contained the length/type field value 88.08 (hex) and the opcode 0x0001, and
- ❏ was of length 64 to 1518† bytes inclusive, and
- ❏ had no CRC error, alignment error, or code error, and
- ❏ pause frames had been enabled/negotiated on the port.

The port could have been in either half- or full-duplex mode.

See the *Rx Align/Code Errors* and *Rx CRC Errors* statistic descriptions for definitions of alignment, code, and CRC errors. Overruns have no effect upon this statistic.

See IEEE Std 802.3z; section 30.3.4.3.

Security Violations

The total number of frames received on the port that the IALE decided were a violation of security and should be discarded. Such a frame:

- ❏ was a data frame (not MAC control frame) destined for any unicast address, and
- ❏ was of length 64 to 1518† bytes inclusive, and
- ❏ did not experience any CRC error, alignment error, code error or overrun, and
- ❏ the lALE decided, based upon its address records, that the frame should be discarded. It will do this if the source address has been marked as ‘‘secure” on this port.

A frame that is counted in this statistic also will be counted in *Rx Filtered Frames*.

This statistic is not completely independent of the ***secvio*** interrupt, i.e., it is not necessary to enable this interrupt for this statistic to operate.

However, if the ***secvio*** interrupt is enabled, further ***secvio*** violations will not be recorded until the interrupt is cleared. The statistic records only the number of interrupts issued.

† 1531 bytes if ***maxlen*** = 0 and ***long*** = 1 in ***StatControl***.

Unknown Unicast Destination Addresses

A count of the number of frames that used the ***UnkUniPorts*** register during the frame-route determination process. Data frames causing CRC errors, alignment errors, code errors, short frames, fragments, and overruns are not recorded in this statistic. MAC control frames do not affect this statistic.

Unknown Multicast Destination Addresses

A count of the number of frames that used the ***UnkMultiPorts*** register during the frame-route determination process. Data frames causing CRC errors, alignment errors, code errors, short frames, fragments, and overruns are not recorded in statistic. MAC control frames do not affect this statistic.

Unknown Source Addresses

A count of the number of frames with unknown source addresses that used the ***UnkSrcPorts*** register during the frame-route determination process. Data frames causing CRC errors, alignment errors, code errors, short frames, fragments, and overruns are not recorded in this statistic. Although ***UnkSrcPorts*** is used when ***cnotify*** = 1 in ***SysControl***, and a known source address changes port, these are not recorded in this statistic. MAC control frames do not affect this statistic.

Appendix B

Hardware-Level I/O Interface Definitions

Each implementation of software routines to read or write a byte to the TNETX4020 DIO port is specific to the hardware-interface technique used to connect the DIO to the host CPU. TI's demonstration software, TSMAN, is a Windows™ 95 program that uses the computer's printer port. The printer port was chosen because it provided satisfactory performance and would allow TSMAN to run on nearly any computer without installing any special hardware. TSMAN could be ported to any other interface technique if code were generated to support the function prototypes listed here. The most basic functions are InByte and OutByte in TSIFHW.H – the rest of these routines are built upon those two.

Topic **Page**

B.1 TSIFHW.H B-2

B.2 TSIF.H B-3

B.1 TSIFHW.H

```
//--------------------------------------------------------------------------
// tsifhw.h – Thunderswitch hardware interface functions
//--------------------------------------------------------------------------

// Thunderswitch host register addresses
// These values asserted onto (SAD_0, SAD_1, SAD_2) to decode the registers
#define TS_ADR_LO       0x0000
#define TS_ADR_HI       0x0001
#define TS_DATA         0x0002
#define TS_DATA_INC     0x0003

// Thunderswitch innternal registers addresses
#define TS_SIO_XCTRL    0x00A1
#define     ECLOK       0x04
#define     ETXEN       0x02
#define     EDATA       0x01

// PC parallel port definitions
#define  Port_BaseAddr     0x378
#define  Port_Data         Port_BaseAddr
#define     SAD_0           0x01
#define     SAD_1           0x02
#define     SAD_2           0x04
#define     MCLK            0x10     // MII management data clock
#define     MDATA           0x20     // MII data line
// MTXEN controls direction of MDATA
// if 1=MDATA is an output, and we are going to drive it
// if 0=MDATA is in input, which means we get off of it and let the
// Phy drive it
#define     MTXEN           0x40
#define     SRNW            0x80     //1=read cycle, 0=write cycle

#define  Port_Status       Port_BaseAddr+1
#define     SRDY            0x80     // SRDY# signal on DIO interface
#define     PERROR          0x20

#define  Port_Control      Port_BaseAddr+2
#define     SCS             0x02     // DIO Chip Select Signal
#define     LATCH           0x04     // Latch
#define     PORT_INPUT      0x20     // 0=write, 1=read

// Function prototypes
extern int InByte( WORD wPort, BYTE* pbData );
extern int OutByte( WORD wPort, BYTE bData );

extern int InMii(WORD wPort_Addr, BYTE* pbData);
extern int OutMii(BYTE b);
```

B.2 TSIF.H

```
//--------------------------------------------------------------------------
//  tsif.h – Thunderswitch hardware interface functions
//--------------------------------------------------------------------------
//  This module contains the definitions and declarations required for
//  a program to interface and use the ThunderSwitch interface functions.
//--------------------------------------------------------------------------

typedef unsigned char  BYTE;    //this better be 8
typedef unsigned short WORD;    //this better be 16
typedef unsigned long  DWORD;   //this better be 32

// Function prototypes
extern int     TsDioRdByte( WORD, BYTE* );
extern int     TsDioWrByte( WORD, BYTE* );
extern int     TsDioRd( WORD, int, BYTE* );
extern int     TsDioWr( WORD, int, BYTE* );
extern int     TsDioDMARd( WORD, int, BYTE* );
extern int     TsDioDMAWr( WORD, int, BYTE* );

extern int     TsRdramWr( DWORD, int, BYTE* );
extern int     TsRdramWr( DWORD, int, BYTE* );

extern int     MiiRdWord( int, int, int, WORD* );
extern int     MiiWrWord( int, int, int, WORD );

extern int     TsEeRd( WORD wAddr, int nCount, BYTE* pbData);
extern int     TsEeWr( WORD wAddr, int nCount, BYTE* pbData);
extern int     TsEeRdByte( WORD, BYTE* );
extern int     TsEeWrByte( WORD, BYTE );
```