SPWS028B – APRIL 1997 – REVISED SEPTEMBER 1997

- **Provides up to 2K-Address Matching System**
- **VLAN Support for Single or Multiple ThunderSWITCH™ Devices**
- **Provides Glueless External-Address Match Interface to the TNETX3150/TNETX3150A**
- **Uses Standard Off-the-Shelf SRAMs**
- **EEPROM Interface for Auto-Configuration, No CPU Needed**
- **Automatic Aging Through User-Selectable Aging (AGE) Threshold**
- **Fabricated in 3.3-V Low-Voltage Technology**
- **Requires 3.3-V and 5-V Power Supplies**
- **5-V Tolerant I/Os**
- **Provides Direct Input/Output (DIO) Interface for Management Access and Control of the Address-Lookup Table**
- **Address Lookups, Adds, and Deletes Are Automatically Performed in Hardware**
- **User-Selectable Interrupts Simplify the Management Operations**
- **Provides Spanning-Tree Support**
- **Secure Addresses From Changing Ports**
- **MII Data I/O (MDIO) PHY Management Interface**
- **Management Access to Statistic Registers**
- **Packaged in 144-Terminal Plastic Quad Flatpack**
- **JTAG Compliant**

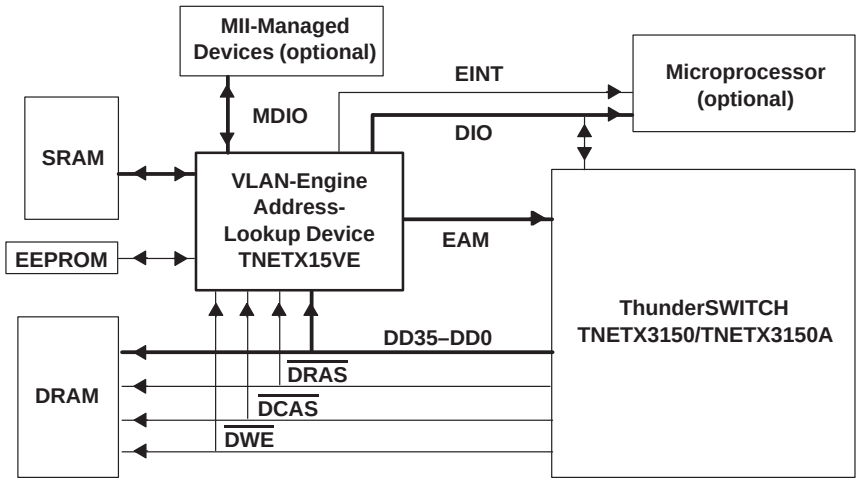
description

The TNETX15VE performs the VLAN-engine address-lookup function for the TNETX3150/TNETX3150A Ethernet™ switch using a glueless interface. The TNETX15VE device determines the addresses to use and match from the TNETX3150/TNETX3150A DRAM bus. The address-lookup table is maintained in external SRAM. The frame-matching and routing information is given to the TNETX3150/TNETX3150A through the external address-matching (EAM) interface.

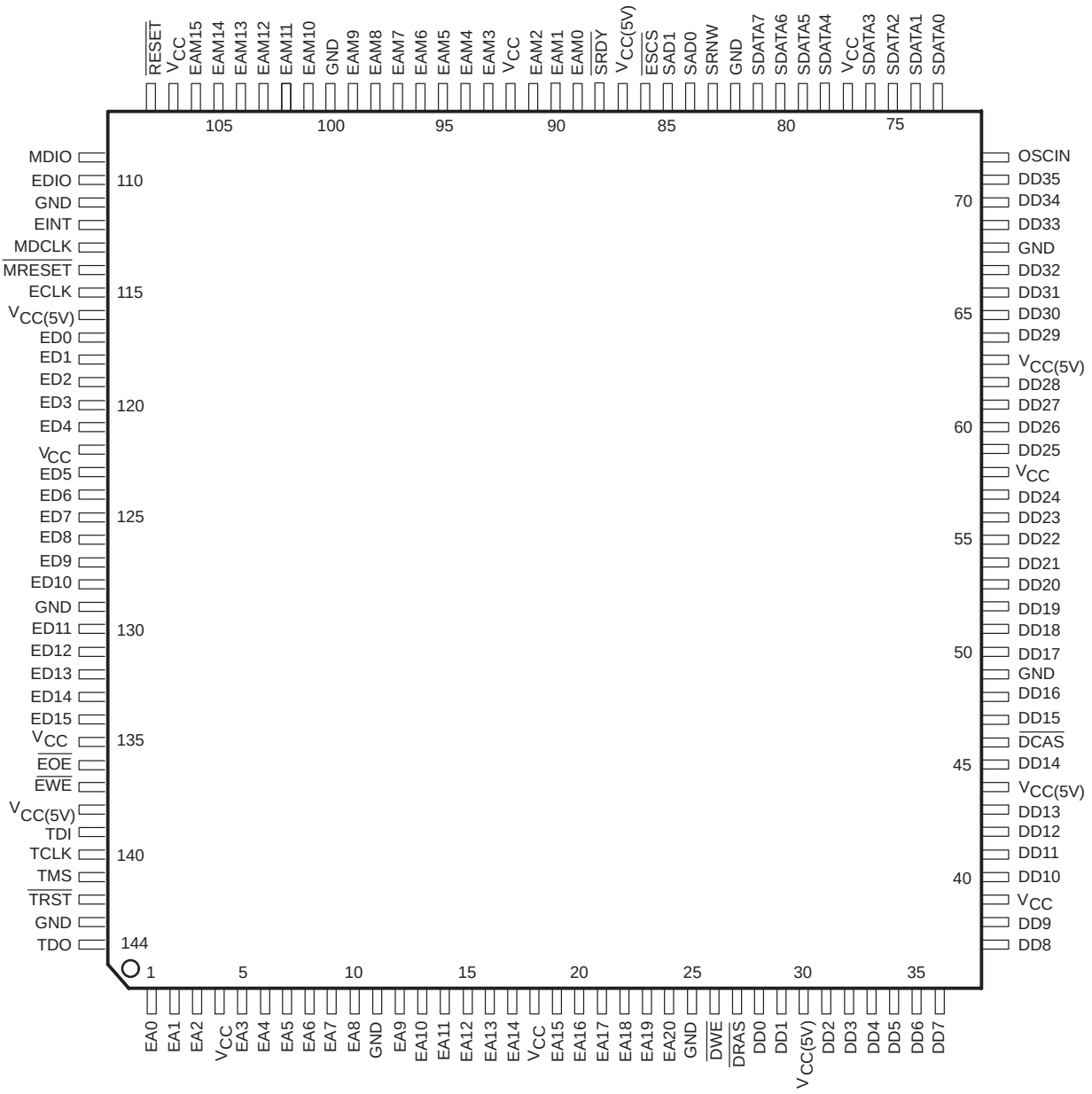
The TNETX15VE is designed to work in either unmanaged or managed mode. Unmanaged operation is accomplished through EEPROM support. Startup options are auto-loaded into the TNETX15VE registers using the EEPROM interface. If the TNETX15VE is initialized by automatic loading from the EEPROM, it begins operation automatically.

All functions of this device are fully controllable by management through a direct input/output (DIO) interface. In addition, this device can interrupt the external management processor with user-selectable interrupts.

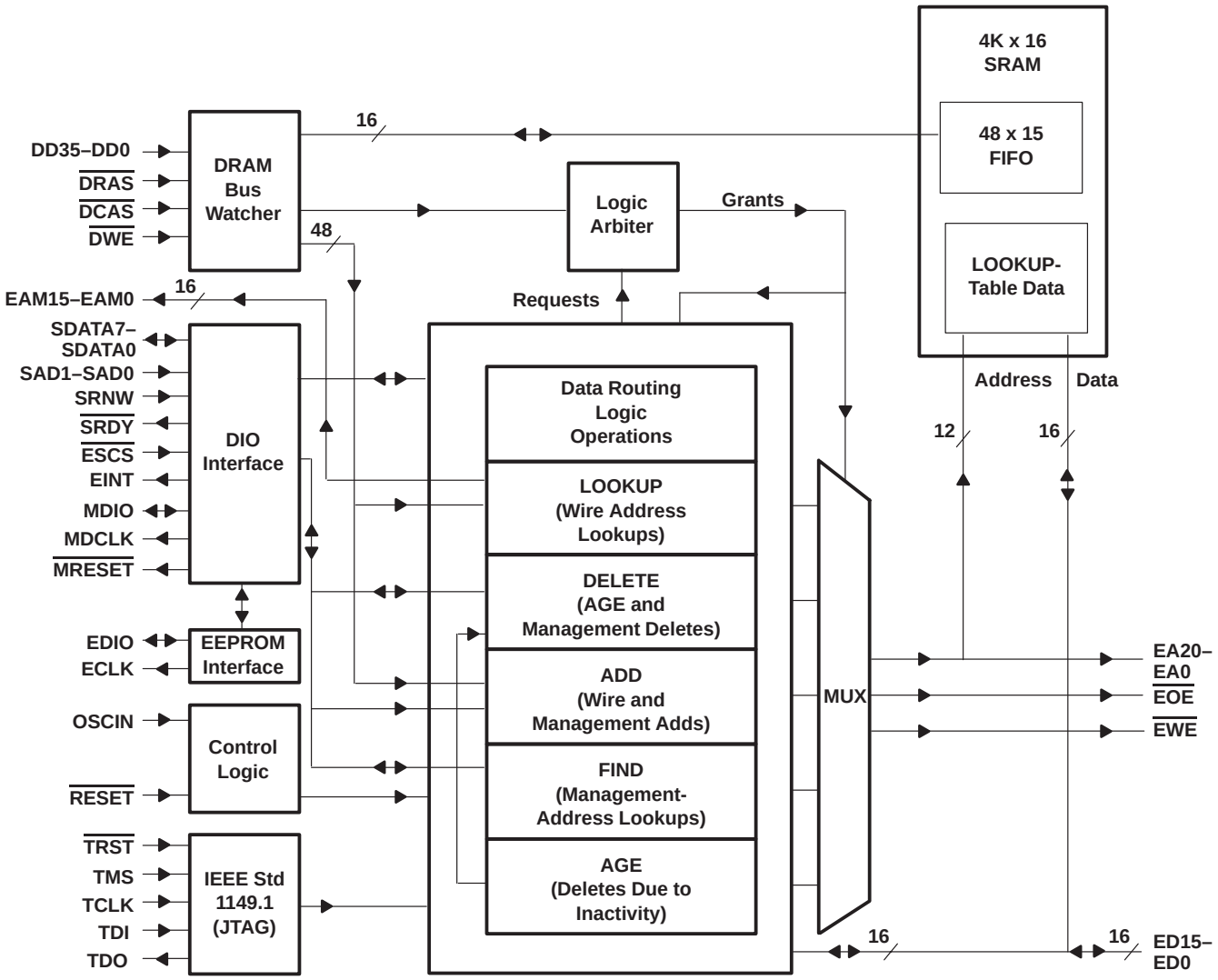
This device also provides support for easy management control of IEEE Std 802.3u media-independent interface (MII) managed devices. A typical application is shown in the following:



PGE PACKAGE
(TOP VIEW)



functional block diagram



general description

bus watcher

The bus watcher interfaces to the TNETX3150/TNETX3150A DRAM interface and extracts destination, source addresses, VLAN information, and the originating port number. The bus watcher is responsible for identifying the frame's start-of-frame and end-of-frame tags. It also interfaces to the arbiter and the TNETX15VE logic to perform off-the-wire address LOOKUPs and ADDs.

DIO interface

The DIO interface enables an attached microprocessor to access the TNETX15VE internal registers. The DIO interface is used to select control modes, read statistics, receive interrupts, read/write to attached MII devices, read/write to an attached EEPROM, and perform management LOOKUPs, ADDs, and DELETEs.

EEPROM interface

The EEPROM interface allows accesses to the EEPROM. It also interfaces with the EEPROM for automatic loading of selected registers from the EEPROM at startup or reset.

arbiter

The arbiter manages the SRAM access for the TNETX15VE logic operations by assigning priorities to the logic operations. Wire lookups have the highest priority, followed by DELETEs, ADDs, management LOOKUPs, and AGEs. The individual logic operations request the bus by asserting a request signal. The arbiter grants the SRAM bus by controlling the SRAM-bus address/data multiplexer.

SRAM address/data multiplexer (MUX)

The address/data MUX is controlled by the arbiter and selects the logic operation that has ownership of the SRAM bus.

TNETX15VE logic operations

The TNETX15VE logic operations consist of LOOKUP, DELETE, ADD, FIND, and AGE operations. Each logic operation is assigned a priority on the SRAM bus and is controlled by the arbiter. The LOOKUP operation has the highest priority and is responsible for wire lookups. The DELETE operation is responsible for either deletes from the AGE operation or for management-delete requests. The ADD operation is responsible for wire adds as well as for management-add requests. The FIND operation is responsible for management searches of the lookup table. The AGE operation is responsible for deleting addresses that have no activity in a fixed time period.

internal 4K × 16 SRAM

The TNETX15VE integrates an internal 4K × 16 SRAM that is used to store the following:

- A one-address-per-port FIFO as part of the bus watcher
- A 32-address FIFO for bridge protocol data unit (BPDU) sources during spanning-tree resolution
- A two-word data entry for each address it has learned

The addresses that the TNETX15VE recognizes are stored as a series of pointers in external (×16) static RAM. The size of the internal RAM imposes a hard limit to the number of addresses that the TNETX15VE can support.

IEEE Std 1149.1 test port (JTAG)

The test-access port is composed of five terminals that are used to interface serially with the device and the board on which it is installed for boundary-scan testing. The TNETX15VE is fully JTAG compliant, with the exception of requiring external pullup resistors on terminals TDI, TMS, and $\overline{\text{TRST}}$.

Contents

Terminal Functions

EAM interface

<table>
<tr><th colspan="2">TERMINAL</th><th>I/O</th><th>DESCRIPTION</th></tr>
<tr><th>NAME</th><th>NO.</th><th></th><th></th></tr>
<tr><td>EAM15†</td><td>106</td><td rowspan="16">O</td><td rowspan="16">External address match single-/multiple-port routing code select. When EAM15 is high, the EAM-interface terminals contain a single-port routing code. When EAM15 is low, the EAM-interface terminals contain a multiple-port routing code (VLAN).
External address match port routing code select. When EAM15 is high, the EAM14–EAM0 terminals are placed in the single-port mode. In this mode EAM14–EAM0 terminals encode a single port to which the frame is routed.
EAM14–EAM5 are don't-care bits and are set to 0. The single-port codes (EAM15–EAM0 terminals) are shown in the following:</td></tr>
<tr><td>EAM14</td><td>105</td></tr>
<tr><td>EAM13</td><td>104</td></tr>
<tr><td>EAM12</td><td>103</td></tr>
<tr><td>EAM11</td><td>102</td></tr>
<tr><td>EAM10</td><td>101</td></tr>
<tr><td>EAM9</td><td>99</td></tr>
<tr><td>EAM8</td><td>98</td></tr>
<tr><td>EAM7</td><td>97</td></tr>
<tr><td>EAM6</td><td>96</td></tr>
<tr><td>EAM5</td><td>95</td></tr>
<tr><td>EAM4</td><td>94</td></tr>
<tr><td>EAM3</td><td>93</td></tr>
<tr><td>EAM2</td><td>91</td></tr>
<tr><td>EAM1</td><td>90</td></tr>
<tr><td>EAM0‡</td><td>89</td></tr>
</table>

TNETX3150/TNETX3150A	EAM15–EAM0															
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Port 00 (uplink)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0 b
Port 01	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1 b
Port 02	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0 b
Port 03	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1 b
Port 04	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0 b
Port 05	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1 b
Port 06	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0 b
Port 07	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1 b
Port 08	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0 b
Port 09	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1 b
Port 10	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0 b
Port 11	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1 b
Port 12	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0 b
Port 13	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1 b
Port 14	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0 b
Reserved	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1 b
No-operation code	1	x	x	x	x	x	x	x	x	x	x	1	x	x	x	x b

The no-operation code signals that the EAM device does not take part in forwarding decisions on this packet. The TNETX15VE does not put out this code in normal operations. To successfully disable the TNETX15VE, it is reset and not started. This causes the EAM interface to float, and external pullup resistors are required on EAM15 and EAM04 (minimally) to supply the no-operation code to the TNETX3150/TNETX3150A device. Erratic operation occurs if the TNETX3150/TNETX3150A address matching is not enabled when the TNETX15VE is disabled. The TNETX15VE, when enabled, overrides the operations of the TNETX3150/TNETX3150A matching circuitry; but, it is strongly advised to disable the TNETX3150/TNETX3150A address matching if the TNETX15VE is active. This prevents two logic blocks from attempting to control port suspension and other functions.

† Most-significant bit
‡ Least-significant bit

Terminal Functions (Continued)

EAM interface (continued)

<table>
<tr><th colspan="2">TERMINAL
NAME NO.</th><th>I/O</th><th>DESCRIPTION</th></tr>
<tr><td colspan="2"></td><td></td><td>
When EAM15 is low, the EAM14–EAM0 terminals encode the multiple ports to which the data frame is routed. Each bit on the EAM14–EAM0 bus is coded as follows:
– A 1 on the bit signifies that the data frame is routed to that port.
– A 0 on the bit signifies that the data frame is not routed to that port.
Terminal number assignments have a one-to-one correspondence to the port numbers as shown in the following:
<table>
<tr><th></th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>EAM bus</td><td>Port 14</td><td>Port 13</td><td>Port 12</td><td>Port 11</td><td>Port 10</td><td>Port 09</td><td>Port 08</td><td>Port 07</td><td>Port 06</td><td>Port 05</td><td>Port 04</td><td>Port 03</td><td>Port 02</td><td>Port 01</td><td>Port 00</td></tr>
</table>
For example, to multicast a frame to ports 08, 05, 02, and 00 a multiport code of 0000 0001 0010 0101 or 0x0125 is sent.
Some packets, originally destined for multiple ports, need to be sent only to a single port after restrictions are applied (VLAN, one of the destination ports = source port, etc.). TNETX15VE uses single-port coding (EAM15 = 1) to reduce the operations required by the TNETX3150/TNETX3150A.
To discard a frame, a code of 0x0000 is output (multiple routing code with no ports selected):
<table>
<tr><th>TNETX3150/TNETX3150A</th><th>EAM14–EAM0</th></tr>
<tr><td>Discard Frame</td><td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td></tr>
</table>
</td></tr>
</table>

Terminal Functions (Continued)

DRAM interface

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
DD35†	71		
DD34	70		
DD33	69		
DD32	67		
DD31	66		
DD30	65		
DD29	64		
DD28	62		
DD27	61		
DD26	60		
DD25	59		
DD24	57		
DD23	56		
DD22	55		
DD21	54		
DD20	53		
DD19	52		
DD18	51		
DD17	50	I	DRAM data bus. Data bus is sourced by the TNETX3150/TNETX3150A.
DD16	48		
DD15	47		
DD14	45		
DD13	43		
DD12	42		
DD11	41		
DD10	40		
DD9	38		
DD8	37		
DD7	36		
DD6	35		
DD5	34		
DD4	33		
DD3	32		
DD2	31		
DD1	29		
DD0‡	28		
$\overline{\text{DCAS}}$	46	I	DRAM column address select. $\overline{\text{DCAS}}$ is sourced by the TNETX3150/TNETX3150A.
$\overline{\text{DRAS}}$	27	I	DRAM row address select. $\overline{\text{DRAS}}$ is sourced by the TNETX3150/TNETX3150A.
$\overline{\text{DWE}}$	26	I	DRAM write enable. $\overline{\text{DWE}}$ is sourced by the TNETX3150/TNETX3150A.

† Most-significant bit
‡ Least-significant bit

Terminal Functions (Continued)

external SRAM interface

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
EA20	24	O	EA20 indicates an internal or external RAM access. 0 = external, 1 = internal.
EA19†	23	O	External address bus. EA20–EA0 is the external SRAM address bus. This bus is always driven except during reset.
EA18	22		
EA17	21		
EA16	20		
EA15	19		
EA14	17		
EA13	16		
EA12	15		
EA11	14		
EA10	13		
EA9	12		
EA8	10		
EA7	9		
EA6	8		
EA5	7		
EA4	6		
EA3	5		
EA2	3		
EA1	2		
EA0‡	1		
ED15†	134	I/O	External data bus. ED15–ED0 is the external SRAM data bus.
ED14	133		
ED13	132		
ED12	131		
ED11	130		
ED10	128		
ED9	127		
ED8	126		
ED7	125		
ED6	124		
ED5	123		
ED4	121		
ED3	120		
ED2	119		
ED1	118		
ED0‡	117		
$\overline{\text{EOE}}$	136	O	External output enable. $\overline{\text{EOE}}$ is the active-low external SRAM output-enable signal.
$\overline{\text{EWE}}$	137	O	External write enable. $\overline{\text{EWE}}$ is the active-low external SRAM write-enable signal.

† Most-significant bit
‡ Least-significant bit

Terminal Functions (Continued)

DIO interface statistics interface

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
EINT	112	0	TNETX15VE interrupt. EINT is an interrupt request from the TNETX15VE to an optional attached microprocessor. Its purpose is to reduce the polling that is required for the processor to be current on the operational states of the TNETX15VE. The interrupt cause is determined by reading the interrupt register (0x28, 0x29). It is not required to acknowledge or act on the interrupts (unmanaged operation). EINT is active high and level sensitive; it is high when any of the interrupt triggers are active.
$\overline{\text{ESCS}}$	86	I	TNETX15VE chip select. When $\overline{\text{ESCS}}$ is active (low), a port access is valid for the TNETX15VE device. $\overline{\text{ESCS}}$ must not be tied to any other chip-select signal (such as the TNETX3150/TNETX3150A $\overline{\text{SCS}}$).
SAD1 SAD0	85 84	I	DIO address bus. SAD1–SAD0 selects the TNETX15VE host registers. SAD1 / SAD0 / DESCRIPTION 0 / 0 / DIO address low 0 / 1 / DIO address high 1 / 0 / DIO data 1 / 1 / DIO data auto-increment
SDATA7† SDATA6 SDATA5 SDATA4 SDATA3 SDATA2 SDATA1 SDATA0‡	81 80 79 78 76 75 74 73	I/O	DIO data bus. SDATA7–SDATA0 is a byte-wide bidirectional DIO bus.
$\overline{\text{SRDY}}$	88	O	DIO ready. When $\overline{\text{SRDY}}$ is active low, the following conditions occur: – When SRNW = 1, the interface host is told that data is valid for reading by the host. – When SRNW = 0, the interface host is told that data has been received by the TNETX15VE. When $\overline{\text{ESCS}}$ is inactive (high), $\overline{\text{SRDY}}$ is disabled [high-impedance (Z) state]. For this reason, $\overline{\text{SRDY}}$ should have a 10-K pullup resister to avoid engaging the hosts ready logic during the time when TNETX15VE is not addressed.
SRNW	83	I	DIO read/not write. SRNW is a read- or write-select signal. – When SRNW is high, the read operation is performed by the host. – When SRNW is low, the write operation is performed by the host.

† Most-significant bit
‡ Least-significant bit

DIO interface MII

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
MDCLK	113	O	MII management data clock. MDCLK is a clock from the TNETX15VE for the serial MII management data MDIO. MDCLK can be disabled [high-impedance (Z) state] by MDIOEN [a bit in the SIO register at 0x0A (DIO)].
MDIO	109	I/O	MII management data input/output. MDIO is the data terminal for serial MII management data. MDIO requires an external pullup resistor for proper operation. MDIO can be disabled [high-impedance (Z) state] by MDIOEN [a bit in the SIO register at 0x0A (DIO)].
$\overline{\text{MRESET}}$	114	O	MII management reset. $\overline{\text{MRESET}}$ is a reset signal for the serial MII management interface. $\overline{\text{MRESET}}$ can be disabled [high-impedance (Z) state] by MDIOEN [a bit in the SIO register at 0x0A (DIO)].

Terminal Functions (Continued)

DIO interface
EEPROM interface

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
ECLK	115	O	EEPROM clock. ECLK is the serial EEPROM data clock and requires an external pullup register.
EDIO	110	I/O	EEPROM data input/output. EDIO is the serial EEPROM data I/O signal. This terminal requires an external pullup (see EEPROM data sheet) for EEPROM operation.

control logic interface

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
OSCIN	72	I	Oscillator input. OSCIN is the clock input that drives the internal state machines. For no-glue interface to the TNETX3150/TNETX3150A, OSCIN is connected to the TNETX3150/TNETX3150As OSCIN terminal (AC12).
$\overline{\text{RESET}}$	108	I	Reset. $\overline{\text{RESET}}$ (active low) resets the TNETX15VE. $\overline{\text{RESET}}$ must remain active for at least 25 ms after power and the clocks have stabilized on power up. After the power-up cycle, $\overline{\text{RESET}}$ must remain active for a minimum of 3 μs for recognition.

JTAG interface

TERMINAL NAME	TERMINAL NO.	I/O	DESCRIPTION
TCLK	140	I	Test clock. TCLK clocks state information and test data into and out of the device during operation of the test port.
TDI	139	I	Test data input. TDI serially shifts test data and test instructions into the device during operation of the test port.
TDO	144	O	Test data out. TDO serially shifts test data and test instructions out of the device during operation of the test port.
TMS	141	I	Test mode select. TMS controls the state of the test-port controller.
$\overline{\text{TRST}}$	142	I	Test reset. $\overline{\text{TRST}}$ asynchronously resets the test-port controller when set low.

power interface

TERMINAL NAME	TERMINAL NO.	DESCRIPTION
GND	11, 25, 49, 68, 82, 100, 111, 129, 143	Ground. GND is the 0-V reference for the device.
V_{CC}	4, 18, 39, 58, 77, 92, 107, 122, 135,	Supply voltage. VCC = 3.3 ± 0.3 V. V_{CC} is the main-logic power supply.
$V_{CC(5V)}$	30, 44, 63, 87, 116, 138	Supply voltage. $V_{CC(5V)} = 5 \pm 0.5$ V. $V_{CC(5V)}$ is the clamp rail voltage on I/O terminals that gives TTL capability.

register descriptions

general notation notes

Hexadecimal notation: Leading 0, followed by lower case x, then hex digits, with space or punctuation delimiters on both ends. A...F are equivalent to a...f (example: 0x12bc, 0x12BC)

Binary notation: Trailing lower-case b after string composed of only 1 or 0, with space or punctuation delimiters on both ends (example: 101011b).

Decimal notation: A number composed of 0...9 digits with no prefix or suffix, with a space or punctuation delimiters on both ends (example: 23965)

The least-significant bit in a register is the lowest number.

The least-significant byte of a multibyte value is the lowest number.

node address format (Ethernet native data format)

The node addresses are kept inside TNETX15VE in Ethernet native data format. This is the same format used when the address appears in the frame transmitted on the wire. Ethernet first transmits the least-significant bit of a data byte on the wire. This makes the group/specific address appear in bit 40 (least-significant bit of most-significant byte). TNETX15VE address storage format is shown in the following:

BIT															
BYTE (most-significant byte)				BYTE		BYTE		BYTE		BYTE		BYTE			
47	42	41	40	39	32	31	24	23	16	15	8	7	0		
		LOCAL/UNIVERSAL ADDRESS	GROUP/SPECIFIC ADDRESS												

DIO register access

There are only four byte-wide DIO registers that are hardware interfaced to the DIO interface. These registers are treated like a pointer structure to locate internal registers, some of which are interfaced to external terminals of the device (see Figure 1). The small number of registers actually on the DIO interface give the TNETX15VE a small I/O map requirement when designing systems; it also allows additional registers to be defined without requiring the hardware interface to change.

To access an internal register, post both halves of the address to the DIO address-low and DIO address-high registers and read or write the DIO data register. If the host performs a read, the internal state machine enables the internal register at the posted address to drive the bus when the read cycle is active. If the host performs a write to a TNETX15VE register, the value written to the data register address is transferred to the posted address in the DIO address-low and DIO address-high registers. $\overline{\text{SRDY}}$ going low on a host read indicates that the internal fetch was accomplished. A delay of up to 60 ns after cycle start is encountered if the resource being requested is currently being used by an internal state machine. There are dedicated cycles for DIO to prevent lockout, but they are allocated as 2 of 5, and DIO is asynchronous to internal processes. $\overline{\text{SRDY}}$ going low on a host write indicates that the value written to the data register is captured and is written to the resource addressed. For maximum throughput, $\overline{\text{SRDY}}$ should be part of the interface; if not, add 60 ns to the minimum DIO cycle times.

If a range of consecutive (rising in order) accesses are needed, reading or writing to the DIO data auto-increment register causes a post increment of the DIO address low and DIO address high registers after each access. Reads and writes can be mixed in a string of accesses; the post increment is by one after each cycle.

Since the DIO address high register is not likely to need the upper bits, one of them is overloaded; that is, it is given a special use. A complete hardware reset is accomplished by writing 0x40 to the DIO address-high register. The DIO address-low register is a don't care during this action.

DIO register access (continued)

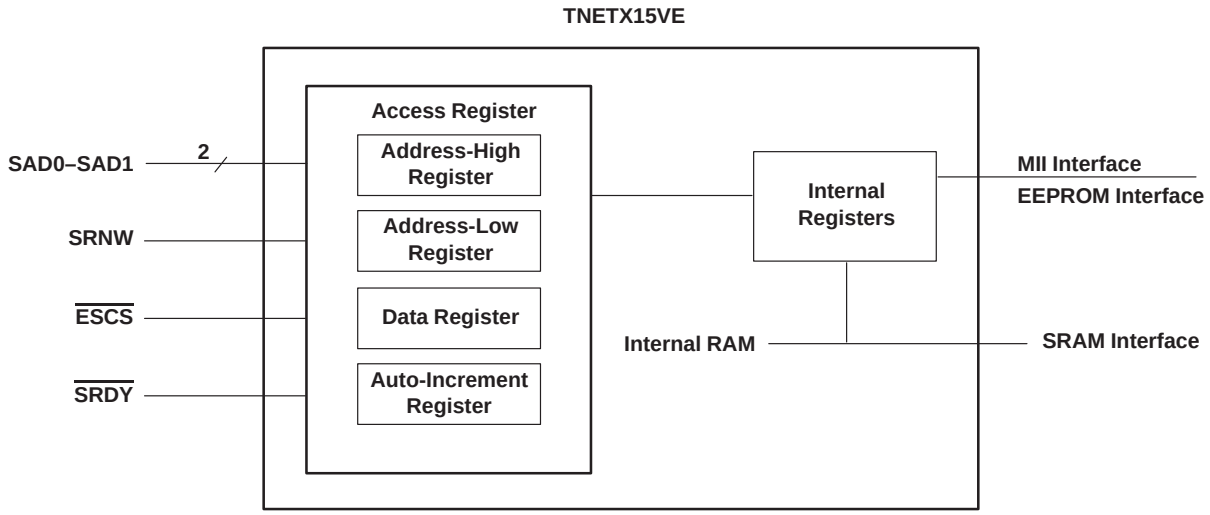


Figure 1. DIO Access (Pointer) Registers

All registers are set to their default values on a hardware reset (setting the $\overline{\text{RESET}}$ terminal high or by writing 0x40 to DIO address high). All registers, except the control register, are also set to their default values on a software reset (setting the RESET bit high in the control register).

SAD1	SAD0	DESCRIPTION
0	0	DIO address low
0	1	DIO address high
1	0	DIO Data
1	1	DIO data auto-increment

DIO address register DIOADR at SAD1–SAD0 = 00b and 01b (host)

DIO ADDRESS HIGH								DIO ADDRESS LOW							
15†	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0‡
DIOADR															

BIT	NAME	FUNCTION
15–0	DIOADR	DIO address. DIOADR contains the internal DIO address used on subsequent accesses to the DIO data or DIO data auto-increment registers. This field auto-increments (by one) on all accesses to the DIO data auto-increment register. Writing a value of 0x40 to DIO address high (DIO address low is ignored) places the TNETX15VE in a hardware reset state. This action is equivalent to resetting the TNETX15VE by setting the $\overline{\text{RESET}}$ terminal low.

† Most-significant bit
‡ Least-significant bit

DIO data register DIODATA at SAD1–SAD0 = 10b (DIO)

The DIO data register address allows indirect access to internal TNETX15VE registers. This is a temporary staging register for data being written through to an internal register (write) or fetched from an internal register (read).

DIO data auto-increment register DIODATAINC at SAD1–SAD0 = 11b (DIO)

The DIO data auto-increment register address allows indirect access to internal TNETX15VE registers. This is a temporary staging register for data being written through to an internal register (write) or fetched from an internal register (read). Accesses to this register cause a post-increment of the DIO address register.

internal register map

The DIO addresses for the internal registers are shown in the following:

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Agingtimer (see Note 1)		RAMsize (see Note 1)	Revision	0x00
		NLRNports (see Note 1)		0x04
FindVLANID	SIO	Control (see Note 1)		0x08
Findnode23–Findnode16	Findnode31–Findnode24	Findnode39–Findnode32	Findnode47–Findnode40	0x0C
Findport		Findnode7–Findnode0	Findnode15–Findnode8	0x10
SECVIOctr	Findcontrol	Findnodeage		0x14
Unkmultictr		Unkunictr		0x18
		Numnodes		0x1C
MANtest	RAMaddr			0x20
		RAMdata		0x24
Intmask		Int		0x28
		AddVLANID	Adddelcontrol	0x2C
Newnode23–Newnode16	Newnode31–Newnode24	Newnode39–Newnode32	Newnode47–Newnode40	0x30
Newport		Newnode7–Newnode0	Newnode15–Newnode8	0x34
Addnode23–Addnode16	Addnode31–Addnode24	Addnode39–Addnode32	Addnode47–Addnode40	0x38
Addport		Addnode7–Addnode0	Addnode15–Addnode8	0x3C
Agednode23–Agednode16	Agednode31–Agednode24	Agednode39–Agednode32	Agednode47–Agednode40	0x40
AgedVLAN	Agedport	Agednode7–Agednode0	Agednode15–Agednode8	0x44
Delnode23–Delnode16	Delnode31–Delnode24	Delnode39–Delnode32	Delnode47–Delnode40	0x48
Delport	DelVLANID	Delnode7–Delnode0	Delnode15–Delnode8	0x4C
VLANmask1 (see Note 1)		VLANmask0 (see Note 1)		0x50
VLANmask3 (see Note 1)		VLANmask2 (see Note 1)		0x54
				0x58
				0x5C
				0x60
				0x64
				0x68
				0x6C

NOTE 1: These registers are automatically loaded from the attached EEPROM when the LOAD bit in the control register is set or when the TNETX15VE is hardware reset by either deasserting the $\overline{\text{RESET}}$ terminal or by setting DIO address high to 0x40.

internal register map (continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
TagVLAN6/TagVLAN7 (see Notes 1 and 2)	TagVLAN4/TagVLAN5 (see Notes 1 and 2)	TagVLAN2/TagVLAN3 (see Notes 1 and 2)	TagVLAN0/TagVLAN1 (see Notes 1 and 2)	0x70
TagVLAN14/TagVLAN15 (see Notes 1 and 2)	TagVLAN12/TagVLAN13 (see Notes 1 and 2)	TagVLAN10/TagVLAN11 (see Notes 1 and 2)	TagVLAN8/TagVLAN9 (see Notes 1 and 2)	0x74
			VLANID1 (see Note 1)	0x78
				0x7C
			VLANID2 (see Note 1)	0x80
				0x84
			VLANID3 (see Note 1)	0x88
				0x8C
			VLANID4 (see Note 1)	0x90
				0x94
			VLANID5 (see Note 1)	0x98
				0x9C
			VLANID6 (see Note 1)	0xA0
				0xA4
			VLANID7 (see Note 1)	0xA8
				0xAC
			VLANID8 (see Note 1)	0xB0
				0xB4
			VLANID9 (see Note 1)	0xB8
				0xBC
			VLANID10 (see Note 1)	0xC0
				0xC4
			VLANID11 (see Note 1)	0xC8
				0xCC
			VLANID12 (see Note 1)	0xD0
				0xD4
			VLANID13 (see Note 1)	0xD8
				0xDC
			VLANID14 (see Note 1)	0xE0
				0xE4
				0xE8
				0xEC
	Mirrorport		UPLINKport	0xF0
RXblock		TXblock		0xF4
			CPUTX	0xF8

NOTES: 1. These registers are automatically loaded from the attached EEPROM when the LOAD bit in the control register is set or when the TNETX15VE is hardware reset by either deasserting the $\overline{\text{RESET}}$ terminal or by setting DIO address high to 0x40.
2. Each byte contains two VLANIDs, one for each offset (x/y) specified in the name.

default register values at reset

The reset values for the default registers are shown in the following:

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
0x0000		0x00	0xab (†)	0x00
0x0000		0x0000		0x04
0x00	MDIO*(0x10) + EDIO*(0x01)	0x0000		0x08
0x00	0x00	0x00	0x00	0x0C
0x0000		0x00	0x00	0x10
0x00	0x00	0x0000		0x14
0x0000		0x0000		0x18
0x0000		0x0000		0x1C
0x00	0x000000			0x20
0x0000		SRAM data at address 0		0x24
0x0000		0x0000		0x28
0x0000		0x00	0x00	0x2C
0x00	0x00	0x00	0x00	0x30
0x0000		0x00	0x00	0x34
0x00	0x00	0x00	0x00	0x38
0x0000		0x00	0x00	0x3C
0x00	0x00	0x00	0x00	0x40
0x00	0x00	0x00	0x00	0x44
0x00	0x00	0x00	0x00	0x48
0x00	0x00	0x00	0x00	0x4C
0x7FFF		0x7FFF		0x50
0x7FFF		0x7FFF		0x54
0x00000000				0x58
0x00000000				0x5C
0x00000000				0x60
0x00000000				0x64
0x00000000				0x68
0x00000000				0x6C
0x00	0x00	0x00	0x00	0x70
0x00	0x00	0x00	0x00	0x74
0x000000			0x00	0x78
0x00000000				0x7C
0x000000			0x00	0x80
0x00000000				0x84
0x000000			0x00	0x88
0x00000000				0x8C
0x000000			0x00	0x90
0x00000000				0x94
0x000000			0x00	0x98
0x00000000				0x9C

† a = major version, b = minor version. These bits are hard coded on every device but change as changes are made to TNETX15VE and its derivatives.

default register values at reset (continued)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
0x000000			0x00	0xA0
0x00000000				0xA4
0x000000			0x00	0xA8
0x00000000				0xAC
0x000000			0x00	0xB0
0x00000000				0xB4
0x000000			0x00	0xB8
0x00000000				0xBC
0x000000			0x00	0xC0
0x00000000				0xC4
0x000000			0x00	0xC8
0x00000000				0xCC
0x000000			0x00	0xD0
0x00000000				0xD4
0x000000			0x00	0xD8
0x00000000				0xDC
0x000000			0x00	0xE0
0x00000000				0xE4
0x00000000				0xE8
0x00000000				0xEC
0x00	0x00	0x00	0x00	0xF0
0x0000		0x000000		0xF4
0x000000			0x0F	0xF8

EEPROM auto-configuration from an external x24C02 EEPROM

The flash EEPROM interface is provided so the system-level manufacturers can provide an optional configured system to their customers. Customers can change or reconfigure their system and retain their preferences between system power downs.

The flash EEPROM contains configuration and initialization information, which is accessed infrequently (typically at power up and reset).

The TNETX15VE uses the 24C02 serial EEPROM device (2048 bits organized as 256×8). This device uses a two-wire serial interface for communications.

EEPROM programming can be done while in the system through the TNETX15VE EDIO port using suitable driver software.

The organization of the EEPROM data is shown in Table 1. The last register loaded is the control register. This allows a complete initialization by downloading the contents of the EEPROM into the TNETX15VE. During the download, no DIO operations are permitted. The LOAD and RESET bits in the control register cannot be set during a download, preventing a download loop.

The TNETX15VE has reserved space in the register map with corresponding reserved space in the EEPROM. This is to support additional features in future revisions of the device with minimal impact on software drivers written for this device. Although the TNETX15VE does not use the values in the reserved EEPROM locations to load the reserved locations in the register map, the auto-loader reads every address from the beginning of the EEPROM to the CRC to calculate the CRC. That is, changing a loaded value or a reserved value requires a new CRC for the EEPROM.

The TNETX15VE detects the presence/absence of the EEPROM. If it is not installed, the EDIO terminal should be tied low. For EEPROM operation, the terminal requires an external pullup (see EEPROM data sheet). When no EEPROM is detected, the TNETX15VE assumes default register values at power up and is halted. Downloading a configuration from the EEPROM terminals is disabled when no EEPROM is present.

The first bit written to or read from the EEPROM is the most-significant bit of the byte, i.e., data (7). Therefore, writing the address 0xC0 is accomplished by writing a 1 and then 1, 0, 0, 0, 0, 0, 0. For details of reading, writing, or programming a 24C02 device, refer to its data sheet.

The TNETX15VE expects data to be stored in the EEPROM in a specific format. The range from 0x00 to 0xA2 in the EEPROM is reserved for use by the adapter. The contents of the remaining bytes are user defined. The EEPROM can be read/written by driver software through the SIO register.

A 32-bit CRC value must be calculated from the EEPROM data and placed in the EEPROM in the location following the bytes loaded into the internal registers. The TNETX15VE uses this 32-bit CRC to validate the EEPROM data. If the CRC fails, the TNETX15VE registers are set to their default (hardwired) values. This is the same state the TNETX15VE is in if no EEPROM is present. Without an EEPROM, a management CPU is required to load all the registers and set the start bit to 1 in the upper one-half of the control/status register (0x09). The CRC calculation on the EEPROM bytes is the same as the IEEE Std 802.3u for the packet CRC calculation on a byte-by-byte basis. For reference, the equation is:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

EEPROM auto-configuration from an external x24C02 EEPROM (continued)

The TNETX15VE EEPROM register assignments are shown in Table 1:

Table 1. EEPROM Register Assignments (Addresses Not Listed Are Reserved)

DIO ADDRESS (All values in hex)	EEPROM MEMORY ADDRESS (All values in hex)	REGISTERS
0x0001	0x00	RAMsize
0x0002–0003	0x01–0x02	Agingtimer
0x0004–0005	0x03–0x04	NLRNports
0x0050–0051	0x05–0x06	VLANmask0
0x0052–0053	0x07–0x08	VLANmask1
0x0054–0055	0x09–0x0A	VLANmask2
0x0056–0057	0x0B–0x0C	VLANmask3
0x0070	0x25	TagVLAN0/TagVLAN1
0x0071	0x26	TagVLAN2/TagVLAN3
0x0072	0x27	TagVLAN4/TagVLAN5
0x0073	0x28	TagVLAN6/TagVLAN7
0x0074	0x29	TagVLAN8/TagVLAN9
0x0075	0x2A	TagVLAN10/TagVLAN11
0x0076	0x2B	TagVLAN12/TagVLAN13
0x0077	0x2C	TagVLAN14/TagVLAN15
0x0078	0x2D	VLANID1
0x0080	0x35	VLANID2
0x0088	0x3D	VLANID3
0x0090	0x45	VLANID4
0x0098	0x4D	VLANID5
0x00A0	0x55	VLANID6
0x00A8	0x5D	VLANID7
0x00B0	0x65	VLANID8
0x00B8	0x6D	VLANID9
0x00C0	0x75	VLANID10
0x00C8	0x7D	VLANID11
0x00D0	0x85	VLANID12
0x00D8	0x8D	VLANID13
0x00E0	0x95	VLANID14
0x0008–0009	0x9D–0x9E	Control
	0x9F–0xA2	CRC

internal register descriptions

revision register, revision at 0x00 (DIO)

MOST-SIGNIFICANT BIT			BIT				LEAST-SIGNIFICANT BIT
7	6	5	4	3	2	1	0
Product Code				Revision			
Initial Values After Reset							
0	0	1	0	0	0	0	0

This register contains the revision code for the device. The initial revision code is hardwired to 0x20, which indicates the version 2.0 device. This register is read only, and writes to it are ignored.

SRAM size-select register, RAMsize at 0x01 (DIO)

The RAMsize register can be written to only when the START bit in the control register is set to 0. This register is automatically loaded from the EEPROM when the $\overline{\text{RESET}}$ terminal is asserted low, when DIO address high register contains 0x40, or when LOAD in the control register is set.

BIT							
7	6	5	4	3	2	1	0
Reserved				RSIZE			
Initial Values After Reset							
0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
7–4	Reserved	Writes to this location are ignored and are read as 0.
3–0	RSIZE	Ram size select. This field indicates the size of the external SRAM, and has a bearing on the number of addresses that the TNETX15VE supports. The TNETX15VE uses this field to determine how many tables to initialize as part of the reset sequence by indicating how many address lines to use. All the intermediate pointers used for address decode are stored in external SRAM, which means external SRAM is required. The end points of an address lookup are stored in the internal RAM for speed. The internal SRAM is used also for the one-address/port FIFO, and the 32-location FIFO used to record source addresses for frames that override the receive block (spanning-tree BPDU support). This field is written only when the START bit is 0. This field also is loaded from the EEPROM interface during auto-load. Code values are as follows: **RAMsize REGISTER** / **RAM SIZE** / **WORST CASE** 0x00 / 640 × 16 / 2 0x01 / 832 × 16 / 2 0x02 / 1K × 16 / 3 0x03 / 2K × 16 / 7 0x04 / 4K × 16 / 14 0x05 / 8K × 16 / 28 0x06 / 16K × 16 / 59 0x07 / 32K × 16 / 123 0x08 / 64K × 16 / 251 0x09 / 128K × 16 / 507 0x0A / 256K × 16 / 1019 0x0B / 512K × 16 / 1954 0x0C / 1M × 16 / 1954 0x0D to 0x0F / Reserved Caution should be taken not to specify more RAM than is actually present. The state machines allocating space treat this value literally. Unusual effects concerning addresses can be observed if this space is smaller than indicated. For instance, addresses can appear more than once with the first few bits changed to 1s.

age-deletion time-select register, agingtimer at 0x02–0x03 (DIO)

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Agingtimer															
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The agingtimer register is 16 bits wide and is used to control the aging process. There are two aging modes, and the modes are selected according to the value of this register:

- When agingtimer is 0 or 0xFFFF, the TNETX15VE performs table-full aging. TNETX15VE ages out the oldest address (only when the lookup table becomes full).
- When agingtimer is not 0 or 0xFFFF, the TNETX15VE performs threshold aging. The value in the Agingtimer is the time threshold in seconds. All addresses that are older than this time are aged out.

Aging does not delete addresses that have been secured, and multicast addresses also are not aged. Aging is disabled when the NAUTO bit in the control register is set. It is management's responsibility in NAUTO mode to manage the lookup table.

All bits in this register are read/writeable and default to 0x0 during reset. This field also is automatically loaded from the EEPROM when the $\overline{\text{RESET}}$ terminal is asserted low, when the DIO address high register contains 0x40 or when LOAD in the control register is set to 1.

learning-disable register, NLRNports at 0x04–0x05 (DIO)

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	NLRNports														
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The NLRNports register disables learning for the ports whose corresponding bit is 1. Frames are still received and routed but they cause no additions to the lookup table. This register is automatically loaded from the EEPROM in a hardware reset ($\overline{\text{RESET}}$ = 0 or when the DIO address-high register contains 0x40) or when the LOAD bit in the control register is set. It is read/write at any time.

mode-control status register, control at 0x08–0x09 (DIO)

BYTE 1								BYTE 0							
BIT															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RESET	LOAD	START	INITD	NEEPM	NAUTO	Reserved		NIOB	Reserved	NCRC	MIRR	Reserved			
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Values With No EEPROM Detected															
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Values When Auto-Loading Fails															
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

The control register is automatically loaded from an EEPROM when the $\overline{\text{RESET}}$ terminal is asserted low, when the DIO address high register is written with 0x40, or when the LOAD bit is set. Only selected bits in this register are loaded from the EEPROM. RESET and LOAD bits are not loaded to prevent automatic loading loops. The two status bits, INITD and NEEPM, also are not loadable. If automatic loading fails due to EEPROM malfunction or CRC error, the control register RESET bit is set.

mode-control status register, control at 0x08–0x09 (DIO) (continued)

BIT	NAME	FUNCTION
15	RESET	Reset. Writing a 1 to RESET places the TNETX15VE in a hardware reset state. This function sets all internal logic operations to a known state, and clears all registers (except for control). (All data from the lookup table is lost.) This bit is not automatically loaded from the EEPROM. If EEPROM automatic loading fails, then the RESET bit is set to 1. This bit must be cleared by the host for the part to exit the reset state.
14	LOAD	Load system. Writing a 1 to LOAD starts the automatic loading of registers from the attached EEPROM. This bit is not automatically loaded from the EEPROM. The EEPROM automatic loader clears this bit to 0 (writing 0 to this bit has no effect). This is another way to initialize registers besides the auto-load (if EEPROM present) after reset.
13	START	Start system. Writing a 1 to START causes the TNETX15VE to begin operation. While the SRAM tables are initialized, no address checking is performed. (Writing a 0 to this bit has no effect.)
12	INITD	RAM-initialization-done signal. INITD becomes high when the lookup table SRAM is initialized. The TNETX15VE begins learning/matching addresses after this signal goes high. This is a read-only bit. Expect this to take 40 μs (internal RAM) + 40 ns × RAMsize.
11	NEEPM	No external EEPROM. NEEPM indicates when an internal EEPROM is detected. If this bit is 1, no EEPROM is present, or the TNETX15VE is unable to detect it. If this bit is set to 0, an EEPROM is detected. This is a read-only bit.
10	NAUTO†	NOT automatically add-address mode select. NAUTO selects the manner in which addresses are added to the lookup table. In NAUTO mode, the aging logic operation is disabled. It is management's responsibility to manage the lookup table in this mode. • When set to 1, the TNETX15VE adds addresses only to the lookup table when a DIO ADD command is given to it. • When set to 0, the TNETX15VE automatically adds unknown addresses to its lookup table.
9–8	Reserved	Writes to these locations are ignored and read as 0.
7	NIOB	Not-in-order broadcast coding. NIOB must be set to 0. The obsolete mode requested by setting this bit to 1 does not properly support multicast, VLAN, or spanning tree. NIOB must be matched by the TNETX3150/TNETX3150A mode bit that is inverted (IOBMOD = 1) from the TNETX15VE mode bit.
6	Reserved	Writes to these locations are ignored and read as 0.
5	NCRC†	No CRC check. NCRC enables/disables the add-on-only-good CRC function. • When set to 1, the TNETX15VE adds frames immediately after the source address is found on the DRAM bus. A good CRC is not required to add the source address to the table. • When set to 0, the TNETX15VE waits until the EOB/EOF and a good CRC indication before adding addresses.
4	MIRR	Mirror-port mode. MIRR enables port-mirroring capabilities for the TNETX15VE. The port to be mirrored is written in the mirrorport register. • When set to 1, all frames received on this port or sent to this port are copied to the port specified in the UPLINKport register. • When set to 0, the TNETX15VE does not perform port mirroring.
3–0	Reserved	Writes to these locations are ignored and read as 0.

† These signals should not be allowed to change state after the TNETX15VE begins operation (by writing a 1 to START).

serial interface I/O register, SIO at 0x0A (DIO)

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
NMRST	MCLK	MTXEN	MDATA	MDIOEN	ECLOK	ETXEN	EDATA
Initial Values After Reset							
0	0	0	MDIO	0	0	0	EDIO

The SIO register contains the control bits for two serial interfaces to TNETX15VE. Setting the outbound signals changes the state of the external terminal. Sampling the inbound signals gives the state of the external terminal. The MII to the PHY devices has a serial management portion. This interface can have multiple masters, and allows for two way data between master and slave. The interface to the EEPROM is simpler because it is a dedicated path between one TNETX15VE and its EEPROM, although it is also serial and two way.

BIT	NAME	FUNCTION
7	MNRST	MII NOT reset. The state of MNRST directly controls the state of the $\overline{\text{MRESET}}$ line (MII reset) • If MNRST is set to 0: The $\overline{\text{MRESET}}$ line is set to 0. • If MNRST is set to 1: The $\overline{\text{MRESET}}$ line is set to 1. This bit is not self-clearing and must be manually deasserted. It can be set low and then immediately set high. Since every PHY attached to the MII may not have a reset terminal, you need to do MNRST and also individually reset each PHY through MII commands. The default state of this bit is 0 (MII is in reset). The MDIOEN bit must be set to drive $\overline{\text{MRESET}}$.
6	MCLK	MII SIO clock. MCLK controls the state of the MDCLK terminal. The MDIOEN bit must be set to drive MDCLK. • When MCLK is set to 1, MDCLK is set to 1. • When MCLK is set to 0, MDCLK is set to 0.
5	MTXEN	MII SIO transmit enable. MTXEN is used with the MDATA bit to read/write information from/to the MDIO terminal. The MDIOEN bit must be set to drive MDIO. • When MTXEN is set to 1, the MDIO terminal is driven with the value in the MDATA bit. • When MTXEN is set to 0, MDATA is loaded with the value in the MDIO terminal.
4	MDATA	MII SIO data. MDATA is used with MTXEN to read/write information from/to the MDIO terminal. • When MTXEN is set to 1, MDIO is driven with the value in this bit. • When MTXEN is set to 0, this bit is loaded with the value on MDIO. The MDIOEN bit must be set to drive MDIO. The default value of this bit is the value driven on the MDIO terminal.
3	MDIOEN	MII SIO data terminal enable. MDIOEN controls the high-Z state of the MDIO, MDCLK, and $\overline{\text{MRESET}}$ terminals. • Setting MDIOEN to 1 enables MII. • Setting MDIOEN to 0 places MII in a high-Z state. The default state of this bit is 0 (MII disabled)
2	ECLOK	EEPROM SIO clock. ECLOK controls the state of the ECLK terminal. • When ECLOK is set to 1, ECLK is set to one. • When ECLOK is set to 0, ECLK is set to 0.
1	ETXEN	EEPROM SIO transmit enable. ETXEN controls the direction of the EDIO terminal. • When ETXEN is set to 1, EDIO is driven with the value in the EDATA bit. • When ETXEN is set to 0, EDATA is loaded with the value on the EDIO terminal.
0	EDATA	EEPROM SIO data. EDATA is used to read or write the state of the EDIO terminal. • When EXTEN is set to 1, EDIO is driven with the value in this bit. • When EXTEN is set to 0, EDATA is loaded with the value on EDIO. The default value of this bit is the value driven on the EDIO terminal.

management-table lookup interface at 0x0B–0x16 (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
FindVLANID				0x08
Findnode23–Findnode16	Findnode31–Findnode24	Findnode39–Findnode32	Findnode47–Findnode40	0x0C
Findport		Findnode7–Findnode0	Findnode15–Findnode8	0x10
	Findcontrol	Findnodeage		0x14

The management-table lookup registers allow the management entity to find information about the node addresses contained in the table.

These registers are part of the address lookup/inspection mechanism supported through the DIO (host) interface. An address is fully described by the six bytes of findnode (0xC to 0x11), and one byte of findVLANID (0xB). Duplicate MAC addresses are handled as long as they are made unique by residing on different VLANs. Data about an address is returned in findport (0x12). FIND commands are issued through the findcontrol (0x16) register. They are discussed later in more detail in numerical order.

To cause a FIND operation to execute, the host writes to the findcontrol register (0x16). If a generalized FIND command is executed (findfirst), the target address is returned in two parts; the MAC address is returned in six bytes of findnode registers, and the VLANID of that address (actually the VLANID of the port that the addresss resides on) is returned in findVLANID. If a successive FIND command is executed (findnext), the starting point to continue the search is the six bytes of findnode and the one byte of findVLANID. The results are returned in the same registers, overwriting the original values. These registers are read/write between FINDS; the search picks up with the previous FIND (which loaded these registers), or values left by writes from the host interface. If a lookup on an address to check parameters is executed, the data returned corresponds to the value in the six bytes of findnode and one byte of findVLANID when the lookup command was given; the address parameters are posted in the findport register (0x12, 0x13). The FIND operation completion is detected when the command bit is written back to 0 (poll) or when the FNDCPLT interrupt is given. The find register block is discussed later in further detail in numerical order.

lookup VLAN tag register, findVLANID at 0x0B (DIO)

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved						VLANID	
Initial Values After Reset							
0	0	0	0	0	0	0	0

The findVLANID register returns the associated VLANID code for the node contained in the findnode registers on a lookup logic operation. On a FIND operation with the VLAN modifier active, the FIND logic operation uses this register's VLANID for searches.

FindVLANID register is a read/write register. When the FIND logic operation is operating, this register is locked and writes are ignored.

lookup-node address register, findnode at 0x0C–0x11 (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Findnode23–Findnode16	Findnode31–Findnode24	Findnode39–Findnode32	Findnode47–Findnode40	0x0C
		Findnode7–Findnode0	Findnode15–Findnode8	0x10

The findnode registers are used to exchange addresses between the TNETX15VE and the management CPU. The node address in findnode is kept in Ethernet native data format. The function of findnode depends on the bits set in findcontrol:

- On FIRST operations, this register shows the first address in the lookup table. The data in findnode is valid only when the FOUND bit in findcontrol is a 1.
- On NEXT operations, this register shows the next address in the lookup table. The data in findnode is valid only when the FOUND bit in findcontrol is a 1.
- On LKUP operations, the lookup logic operation looks up the address stored in this register. If found, the FOUND bit in findcontrol is set to a 1.
- On NEW operations, this register shows the first address in the lookup table that has the NEW flag set. The data in findnode is valid only when the FOUND bit in findcontrol is a 1.
- On NBFR operations, this register shows the first location of the NBLCK RX FIFO. This FIFO stores source address and port information on frames received where the destination address information in the table is tagged with the NBLCK (no block) bit. This FIFO is intended to allow the reception of, and response to, BPDUs during the early states of spanning tree when all other traffic is blocked, and can be used for other operations since any address can have its NBLCK bit set. The data in findnode is valid only when the FOUND bit in findcontrol is a 1.

This register is read/writeable. When the FIND logic operation is operating, this register is locked and no writes can be performed to it.

lookup routing-code register, findport at 0x12–0x13 (DIO)

The findport register is a read-only register, except for the portcode field, which returns port assignment information for the node address contained in the findnode register after a lookup. The data structure for the findport register depends on the type of address stored in the findnode register or whether the NBCLK FIFO was searched.

If findnode contains a unicast address, findnode (40) = 0 (bit 40 of the Ethernet adddress is 0):

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBLCK	SECURE	LOCKED	CUPLNK	PORTCODE				NEW	Reserved						
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15	NBLCK	Not blocked indication. NBLCK overrides the RXblock register. • If NBLCK is 1, frames destined for this node are forwarded even if the corresponding RXblock bit for the port that originated the frame is set. • If NBLCK is 0, frames are forwarded only if the RXblock bit for the port that originated the frame is not set.
14	SECURE	Secured address indication. SECURE shows the security level for the address contained in the findnode register. Secure addresses are not aged out and cannot move ports. If a station with a secured address moves ports, a security violation interrupt is given to the host, and the address is locked.
13	LOCKED	Locked address indication. LOCKED shows the lock status for the address contained in the findnode register. Locked addresses output a discard code on the EAM interface; frames destined for LOCKED addresses are not delivered.
12	CUPLNK	Copy frames to uplink indication. CUPLINK shows the copy uplink status for the address contained in the findnode register. The address tagged by this bit adds the port specified in the uplinkports register to the routing code. This can affect performance of the TNETX3150/TNETX3150A, as each packet must be linked to at least two transmit queues.
11–8	PORTCODE	Current port for node. PORTCODE holds the current port for the unicast address shown in the findnode register. On findport operations, the FIND logic operation uses the port in this field for port-based lookups. This field is read/writeable. When the FIND logic is operating, this register is locked and no writes can be performed to it.
7	NEW	New address indication. NEW shows that the node contained in the findnode registers has been added, or its port has changed, or it has been part of a security violation after the last access by management, or had its NEW bit explicitly set as part of a management add/edit. The FIND state machine clears this bit when finding the node.
6–0	Reserved	Writes to this location are ignored and read as 0.

lookup routing-code register, findport at 0x12–0x13 (DIO) (continued)

If findnode is a multicast address, findnode (40) = 1 (bit 40 of the Ethernet address is 1):

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBLCK	PORTFLAG														
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15	NBLCK	Not blocked indication. NBLCK overrides the RXblock register. • If NBLCK is 1, frames destined to this node are forwarded even if the corresponding RXblock bit for the port that originated the frame is set. • If NBLCK is 0, frames are forwarded only if the RXblock bit for the port that originated the frame is not set.
14–0	PORTFLAG	Port-bit vector for multicast. PORTFLAG shows the port-bit vector for the multicast address contained in the findnode register. The bit values in this field correspond one-to-one with the TNETX3150/TNETX3150A port assignment.

If the FIND operation used the NBFR bit, findport is defined:

The findport register returns the following information when performing NBFR operations (reading the RX FIFO for frames whose destination address is tagged by the NBLCK bit).

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TAG				PORT CODE				Reserved							
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15–12	TAG	Frame's tag field. TAG shows the frame's tag for the frame shown in the findnode register as decoded from the DRAM bus. The TAG field is used by the TNETX15VE to determine the VLAN for a frame coming from port 00.
11–8	PORT CODE	Port code for source address. PORT CODE shows the source port for the frame coming from the address in the findnode register.
7–0	Reserved	Writes to this location are ignored and are read as 0.

node's age-stamp register, findnodeage at 0x14–0x15 (DIO)

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NODEAGE														
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The findnodeage register is a read-only register, which holds the current 16-bit age-time stamp of the address contained in the findnode register and findVLANID register after a FIND operation. Age units are seconds. The time stamp indicates when this address was last seen.

lookup table search control register, findcontrol at 0x16 (DIO)

The management logic uses the findcontrol register to scan the lookup table for addresses. Five commands are available: findfirst (FIRST), findnext (NEXT), findnew (NEW), find (LKUP), and findFIFO (NBFR). Variations on these commands can be achieved by the use of the PORT and VLAN modifier bits for all find commands except findFIFO and find. Only one command is valid at one time. Example: a findfirst and a findnext command cannot be issued at the same time (0x06). The TNETX15VE ignores all multiple commands. This register is a read/write register. When the FIND logic is operating, this register is locked and writes are ignored, allowing the DIO cycle to complete. The bits remain as written while running. Polling this register or using a FNDCPLT interrupt determines the completeness of the find command (if the find command is complete, the bits used to start the command execution are cleared). The PORT and VLAN modifiers can be used at the same time to increase the pool of available match candidates.

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
FOUND	NEW	NBFR	PORT	VLAN	FIRST	NEXT	LKUP
Initial Values After Reset							
0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
7	FOUND	Address found. If the address contained in the findnode register is found in the table, FOUND is asserted. The data contained in the find interface (findVLANID, findnode, findport, and findnodeage) is valid only when this bit is 1. This is a read-only bit.
6	NEW	New address finds. When NEW is 1, the TNETX15VE scans the address table for the first address, which is marked as NEW. After successful finds, the NEW flag is cleared. All searches for addresses with the NEW bit set start at the beginning of the table.
5	NBFR	Nonblocked FIFO read. Setting the NBFR bit requests information from the top of the NBLOCK FIFO (if present). If there is an entry in the FIFO, the information is returned in the findnode and findport locations, and the FOUND bit is set to 1. If there is not an entry in the FIFO, the FOUND bit is set to 0. The NBFR bit stays set until the command finishes. There is an interrupt available to signal when the NBLCK FIFO is not empty and should be searched with this command bit.
4	PORT	Find modifier – find by port. PORT is used with the FIRST, NEXT, NEW, and LKUP bits to find the unicast address whose port assignment matches the port contained in the portcode field of the findport register.
3	VLAN	Find modifier – find by VLAN ID. VLAN is used with the FIRST, NEXT, NEW, and LKUP bits to find addresses whose VLAN ID matches the VLAN ID contained in the findVLANID register.
2	FIRST	Lookup first address. When FIRST is 1, the TNETX15VE scans the address table for the first valid address. It returns this address to the findnode register.
1	NEXT	Lookup next addresses. When asserted, the TNETX15VE scans the address table for the next available address after the address currently in the findnode registers (0xC–0x11). It returns this next address to the findnode register.
0	LKUP	Address lookup. When asserted, the TNETX15VE scans the address table for the address contained in the findnode register. If found, the FOUND bit reads 1; if not found, it reads 0.

statistics registers at 0x17–0x1D (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Secvioctr				0x14
Unkmultictr		Unkunictr		0x18
		Numnodes		0x1C

All registers in this field are read only and their default value after reset is 0.

security violation counter, secvioctr at 0x17 (DIO)

The secvioctr field contains the number of times that a secured address attempts to move ports. This register generates a stat interrupt (statistics overflow interrupt) when it is one-half full (most-significant bit in the field is 1). Reading this register automatically clears it and the default value of this register is 0x00.

unknown unicasts counter, unkunictr at 0x18–0x19 (DIO)

The unkunictr register counts the number of times that the TNETX15VE broadcasts a frame that has a unicast destination address. These frames are used when the TNETX15VE is not able to find the destination address in its lookup table. This register generates a stat interrupt (statistic overflow interrupt) when it is one-half full (most-significant bit in the field is 1). The default value of this register is 0x0000. Reading the least-significant byte loads the most-significant byte into a buffer register that is shared at the most-significant byte address for unkunictr, unkmultictr, and numcodes. The least-significant byte should be read first, followed by the most-significant byte, without initiating a read of another statistics counter. Reading the least-significant byte also clears the counter.

unknown multicasts counter, unkmultictr at 0x1A–0x1B (DIO)

The unkmultictr register counts the number of times that the TNETX15VE broadcasts a frame that has a multicast destination address. Multicast destination addresses are broadcast when the TNETX15VE is not able to find the destination address in its lookup table. This register generates a stat interrupt (statistics overflow interrupt) when it is one-half full (most-significant bit in the field is 1). The default value of this register is 0x0000. Reading the least-significant byte loads the most-significant byte into a buffer register that is shared at the most-significant byte address for unkunictr, unkmultictr, and numcodes. The least-significant byte should be read first, followed by the most-significant byte, without initiating a read of another statistics counter. Reading the least-significant byte also clears the counter.

lookup table node count, numnodes at 0x1C–0x1D (DIO)

The numnodes counter register contains the number of addresses currently in the lookup table. This register is read only and its value at reset is 0x0000. Reading the least-significant byte loads the most-significant byte into a buffer register that is shared at the most-significant byte address for unkunictr, unkmultictr, and numcodes. The least-significant byte should be read first, followed by the most-significant byte, without initiating a read of another statistics counter.

SRAM address register, RAMaddr at 0x20–0x22 (DIO)

The RAMaddr register is used with the RAMdata (at 0x24, 0x25) register to access the TNETX15VE internal or external SRAM. The SRAM accessed (internal or external) is selected through the RSEL bit.

BYTE 2								BYTE 1								BYTE 0							
BIT																							
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INC	Reserved		RSEL	RAM ADD																			
Initial Values After Reset																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
23	INC	Address auto-increment. When INC is 1, this increments the RAM ADD field to access the next location in the SRAM. The address is incremented by one after every time a read or write is performed on the most-significant byte of the RAMdata register.
22, 21	Reserved	Writes to this location are ignored and read as 0.
20	RSEL	SRAM access select. RSEL selects whether the SRAM to be accessed is the internal or the external SRAM. • If RSEL is 1, accesses are to the internal SRAM. • If RSEL is 0, accesses are to the external SRAM.
19–0	RAM ADD	RAM address. This 21-bit field holds the address of the SRAM location, which is read or written. The data that is to be read or written is placed in the RAMdata register. The internal SRAM is 4K × 16 in size. The unused RAM ADD bits are 0 when posting an address for either internal or external access. This interface is read/write at all times except during reset.

manufacturing test register, MANtest at 0x23 (DIO)

This register is reserved for Texas Instruments manufacturing test. It is written to 0x0 for normal operation. All reset mechanisms leave 0x0 in this register, and it is not loaded from the EEPROM. It is written to only when the START bit is not set to 1.

It contains mechanisms, for example, that increment all the counters at the same time (or run the aging algorithm quicker than usual to decrease the time required to test the part at manufacture). Setting any of these bits results in abnormal operation. To allow improvement to the testing, these functions are subject to change.

SRAM data register, RAMdata at 0x24–0x25 (DIO)

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RAM DATA															
Data at SRAM (0x000000)															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The RAMdata register is used to access the SRAM address held in the RAM ADD field of the RAMaddr register. This field is 16-bits wide. The SRAM accessed (internal or external) depends on the RSEL bit in the RAMaddr register.

- Writes to the RAM are executed when the MS byte of the RAMdata register is written to by the host. When depositing values for transfer to SRAM, write the least-significant byte first.
- Reads are accomplished by reading the data from either byte in the RAMdata register.
- The SRAM address to be accessed should be placed in the RAMaddr register. If the INC bit in the RAMaddr register is 1, the address to be accessed is increased after each time the most-significant byte of RAMdata is accessed.

interrupt register, int at 0x28–0x29 (DIO)

The int register is used with the intmask register to provide interrupts to the attached CPU. When TNETX15VE EINT terminal is 1, this register gives the reason for the interrupt. Specific interrupts can be masked out by setting the appropriate bit in intmask register. This register is read only with the exception of the int bit. All bits in a byte are automatically cleared when the byte is read, even if more then one was set.

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NEW	NEWM	CHNG	CHNGM	SECVIO	SECVIOM	AGE	AGEM	INT	FNDCPLT	Reserved			RXFIFO	STAT	FULL
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15	NEW	New-node interrupt. NEW indicates that a new node has been added to the lookup table using a wire ADD. The node address is given in the newnode register and the node's port is given in the newport register.
14	NEWM	Missed-new-node interrupt indication. NEWM indicates that a new node interrupt was given, but the information was not placed in the newnode registers because the registers have not been released by a host read of the most-significant byte of the newport register.
13	CHNG	Node-port change interrupt. CHNG indicates that there has been a change in port assignment for a node that exists in the lookup table. The node address is given in the newnode register and the node's new port is given in the newport register.
12	CHNGM	Missed-node-port change interrupt indication. CHNGM indicates that a node-port change interrupt was issued, but the information was not placed in the newnode registers because the registers have not been released by a host read of the most-significant byte of the newport register.
11	SECVIO	Security-violation interrupt. SECVIO indicates that a node (which has been secured) has attempted to move port assignments. The node address is given in the newnode registers. The location that the node attempted to move to is contained in the newport register.
10	SECVIOM	Missed-security-violation interrupt indication. SECVIOM indicates that a node-port change interrupt was issued, but the information was not placed in the newnode registers because the registers have not been released by a host read of the most-significant byte of the newport register.
9	AGE	Age-out interrupt. AGE indicates that a node has been aged-out (deleted from the lookup table). The node address is issued in the agednode register. The node's assigned port is given in the agedport register.
8	AGEM	Missed-age-out interrupt indication. AGEM indicates that an age-out interrupt was issued, but the information was not placed in the agednode registers because the registers have not been released by a host read of the most-significant byte of the agedVLAN register.
7	INT	Test interrupt request. Asserting INT gives a test interrupt to the attached CPU.
6	FNDCPLT	Find completion interrupt. FNDCPLT indicates that the FIND logic operation has completed a search operation. The data in the find registers is now valid.
5–3	Reserved	Writes to this location are ignored and read as 0.
2	RXFIFO	Receive FIFO has become not empty. The RXFIFO bit indicates that the RXFIFO has moved from being empty to not empty. To set again, RXFIFO must be emptied and receive data again.
1	STAT	Statistics overflow interrupt. STAT indicates that a counter in the statistics is one-half full (most-significant bit in the counter is a 1). This is an indication to the CPU to read the statistic counters (thereby clearing them).
0	FULL	SRAM full interrupt. FULL indicates that a condition existed (or exists) that prevented the adding of another address to the TNETX15VE address-lookup table. Either the external RAM (table pointers), or the internal RAM (table entries) filled up. If NAUTO (bit 10, control 0x9) is set, the host is responsible for adding addresses. The host-initiated ADD operation did not complete, and the FULL interrupt was asserted. The host must delete addresses until the ADD completes, or further ADDs are stalled. If NAUTO is not set, the internal state machines have taken care of deleting the necessary oldest addresses until the ADD operation completes, even if the ADD was initiated by the host. In this case, the interrupt is an indication that the table is operating near peak capacity.

interrupt masking register, intmask at 0x2A–0x2B (DIO)

The intmask register is used with the int register to select the type of interrupts that should be given to the attached CPU. Bit definitions in the intmask register agree one-to-one to the bit definitions in the int register. Only those fields with the bit set to 1 generate an interrupt to the CPU. Setting the mask bit for a condition prevents only the external terminal from moving to a logic 1 when the appropriate condition is true. The appropriate bit in the interrupt register is always a 1 if the condition is true, and the bit has not been cleared by being read. This register is read/writeable.

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NEW	NEWM	CHNG	CHNGM	SECVIO	SECVIOM	AGE	AGEM	INT	FNDCPLT	Reserved			RXFIFO	STAT	FULL
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15	NEW	New-node interrupt mask. When NEW is set, a new-node interrupt is posted if the NEW bit in the int register is set.
14	NEWM	Missed-new-node interrupt mask. When NEWM is set, a missed new-node interrupt is posted if the NEWM bit in the int register is set.
13	CHNG	Node-port change interrupt mask. When CHNG is set, a node-port change interrupt is posted if the CHNG bit in the int register is set.
12	CHNGM	Missed-node-port change interrupt mask. When CHNGM is set, a missed-node-port interrupt is posted if the CHNGM bit in the int register is set.
11	SECVIO	Security-violation interrupt mask. When SECVIO is set, a security-violation interrupt is posted if the SECVIO bit in the int register is set.
10	SECVIOM	Missed-security-violation interrupt mask. When SECVIOM is set, a missed-security-violation interrupt is posted if the SECVIOM bit in the int register is set.
9	AGE	Age-out interrupt mask. When AGE is set, an age-out interrupt is posted if the AGE bit in the int register is set.
8	AGEM	Missed-age-out interrupt mask. When AGEM is set, a missed age-out interrupt is posted if the AGEM bit in the int register is set.
7	INT	Test interrupt mask. When INT is set, a test interrupt is posted if the INT bit in the int register is set.
6	FNDCPLT	Find completion mask. When FNDCPLT is set, a find completion interrupt is posted if the FINDCPLT bit in the int register is set.
5–3	Reserved	Writes to this location are ignored and read as 0.
2	RXFIFO	Receive FIFO not empty. When RXFIFO is set to 1, the receive frame address buffer is not empty. During spanning tree, this signals the arrival of a BPDU.
1	STAT	Statistics overflow interrupt mask. When STAT is set, a statistics interrupt is posted if the STATS bit in the int register is set.
0	FULL	SRAM full interrupt mask. When FULL is set, a memory-full interrupt is posted if the FULL bit in the int register is set.

new-add aged-del register set

The add/delete register set is used by the state machines (logic operations) to report on stations' addresses that have been added, deleted, or involved in port security violations as executed by the internal state machines, or to request such actions be taken on the specified address on behalf of the host.

The newnode and agednode set of registers send status information from the state machines to the host as a result of an interrupt. Addnode and delnode registers are host information/actions to be applied to the TNETX15VE table. The status groups (new, aged) are covered in detail first, followed by the host command groups (add, delete).

REGISTER SET AND ADDRESSES				
BYTE 3	**BYTE 2**	**BYTE 1**	**BYTE 0**	**DIO ADDRESS**
		AddVLANID	Adddelcontrol	0x2C
Newnode23–Newnode16	Newnode31–Newnode24	Newnode39–Newnode32	Newnode47–Newnode40	0x30
Newport		Newnode7–Newnode0	Newnode15–Newnode8	0x34
Addnode23–Addnode16	Addnode31–Addnode24	Addnode39–Addnode32	Addnode47–Addnode40	0x38
Addport		Addnode7–Addnode0	Addnode15–Addnode8	0x3C
Agednode23–Agednode16	Agednode31–Agednode24	Agednode39–Agednode32	Agednode47–Agednode40	0x40
AgedVLAN	Agedport	Agednode7–Agednode0	Agednode15–Agednode8	0x44
Delnode23–Delnode16	Delnode31–Delnode24	Delnode39–Delnode32	Delnode47–Delnode40	0x48
Delport	DelVLANID	Delnode7–Delnode0	Delnode15–Delnode8	0x4C

new-node/port-change/security-violation interrupt interface at 0x30–0x37 (DIO)

BYTE 3	**BYTE 2**	**BYTE 1**	**BYTE 0**	**DIO ADDRESS**
Newnode23–Newnode16	Newnode31–Newnode24	Newnode39–Newnode32	Newnode47–Newnode40	0x30
Newport		Newnode7–Newnode0	Newnode15–Newnode8	0x34

The new-node/port-change/security-violation interrupt interface is used with the int and intmask registers to exchange information relating to a wire event, such as a new source address seen, a source address seen on a new port, or a source address seen on a port other than the one where it was secured, regardless of the state of NAUTO. These registers are valid on a NEW, CHNG, or SECVIO interrupt and can be used to find new addresses when the host processor is managing the address table. These registers are read only and default to 0 on reset.

When NAUTO = 1, this interface with the NEW interrupt reports an address seen on the wire that is not in the table. Since the host is managing the table when NAUTO = 1, this could be used by the host to initiate a management ADD.

To avoid the data in these registers from being changed if another interrupt occurs while reading the current data, the data in these registers is locked until the most-significant byte of the newport register is read.

The bit definition follows:

new-address register, newnode at 0x30–0x35 (DIO)

The newnode registers contain the node address (in Ethernet native data format) for which the interrupt was given. The default value of this register after reset is 0x00.00.00.00.00.00.

new-address port register, newport at 0x36–0x37 (DIO)

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				PORTCODE				Reserved		VLAN		OLDPORT			
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15–12	Reserved	Writes to this location are ignored and read as 0.
11–8	PORTCODE	Current port for node. The PORTCODE field holds the assigned port number for the address contained in the newnode register.
7–6	Reserved	Writes to this location are ignored and read as 0.
5–4	VLAN	Port VLAN assignment. VLAN shows the VLAN code for the port contained in the newnode register. This is a table lookup based on the port that the packet with this address was observed on via the port VLAN assignment registers at 0x78–0xE7.
3–0	OLDPORT	Old port for address. When an address moves port locations, OLDPORT contains the old port location for the address. When a security-violation interrupt is asserted by the TNETX15VE, the SECVIO bit is set in the int register. OLDPORT shows the port where the node attempted to move.

agednode interrupt interface group at 0x40–0x46 (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Agednode23–Agednode16	Agednode31–Agednode24	Agednode39–Agednode32	Agednode47–Agednode40	0x40
AgedVLAN	Agedport	Agednode7–Agednode0	Agednode15–Agednode8	0x44

The agednode interrupt interface is used with the int and intmask registers to pass information to the management agent about addresses that have been deleted from the lookup table due to the internal aging process. The information placed in these registers is written in Ethernet native data format and is only valid when the AGE bit in the int register is set to a 1. These registers are read only and default to 0 after reset.

To prevent the data in these registers from being changed if another interrupt occurs while reading the current data, the data in these registers is locked until the agedVLAN register is read.

aged-address register, agednode at 0x40–0x45 (DIO)

On an age interrupt, the agednode registers contain the address of the node that has been aged out from the lookup table. This is a read-only register and defaults to 0x00.00.00.00.00.00 after reset. A different interrupt is issued if this information remains unread long enough that another node is aged out before the agednode registers are released.

aged-address port assignment, agedport at 0x46 (DIO), byte 2

BIT							
7	6	5	4	3	2	1	0
Reserved				PORTCODE			
Initial Values After Reset							
0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
7–4	Reserved	Writes to this location are ignored and read as 0.
3–0	PORTCODE	Agednode's port assignment. PORTCODE displays the assigned port for the aged-out address contained in the agednode register.

aged-address VLAN assignment, agedVLAN at 0x47 (DIO), byte 3

BIT							
7	6	5	4	3	2	1	0
Reserved				VLANDID			
Initial Values After Reset							
0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
7–4	Reserved	Writes to this location are ignored and read as 0.
3–0	VLANID	Agednode's VLAN assignment. VLANID displays the assigned VLANID for the address contained in the agednode register.

management add/edit address interface group at 0x2D and 0x38–0x3F (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
		AddVLANID	Adddelcontrol	0x2C
Addnode23–Addnode16	Addnode31–Addnode24	Addnode39–Addnode32	Addnode47–Addnode40	0x38
Addport		Addnode7–Addnode0	Addnode15–Addnode8	0x3C

The management add/edit address registers are used with the ADD bit in the adddelcontrol register to perform CPU adds and edits to the lookup table. These registers are read/write registers. When the ADD logic is operating, these registers are locked and no writes can be made to them.

lookup table add/delete command register, adddelcontrol at 0x2C (DIO), byte 0

BYTE 0							
BIT							
7	6	5	4	3	2	1	0
Reserved				DELP	DELV	ADD	DEL
Initial Values After Reset							
0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
7–4	Reserved	Writes to this location are ignored and are read as 0.
3	DELP	See the delete process bits under *lookup table add/delete command register*
2	DELV	See the delete process bits under l*ookup table add/delete command register*
1	ADD	Address add. When ADD is asserted, the TNETX15VE uses the information contained in the management add/edit address interface to add or edit an address in the lookup table. ADD remains asserted until the add process is complete.
0	DEL	See the delete process bits under *lookup table add/delete command register*

add address VLAN tag register, addVLANID at 0x2D (DIO), byte 1

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved						ADDVLANID	
Initial Values After Reset							
0	0	0	0	0	0	0	0

The addVLANID register is used with the addnode and addport registers to add/edit a node's VLAN assignment. This register is a read/write register. When the ADD logic is operating, this register is locked and no writes can be made to it.

add address register, addnode at 0x38–0x3D (DIO)

The addnode register is a read/write register. When the ADD logic is operating, this register is locked and no writes can be made to it. The unicast or multicast address in this register (written in Ethernet native data format) is added to the lookup table when the ADD bit in the adddelcontrol register is set to 1. The default value of this register after reset is 0x00.00.00.00.00.00.

add routing code register, addport at 0x3E–0x3F (DIO)

The addport register is used to change port and flag assignments for the node address contained in the addnode register. The definition for the addport register depends on whether the address stored in the addnode register is a unicast or multicast address.

This register is a read/write register. When the ADD logic is operating, this register is locked and no writes can be made to it.

This interface is used to enter broadcast/multicast addresses to the TNETX15VE table. Wire ADDs operate on packet-source addresses, and these are all defined by IEEE to be unicast addresses. Therefore, bit 47 of the address is stored as 0.

If addnode is a unicast address, (addnode (40) = 0), addport is defined as follows:

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBLCK	SECURE	LOCKED	CUPLNK	PORTCODE				NEW	Reserved						
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15	NBLCK	Not-blocked indication. NBLCK sets the RXblock override function. • If NBLCK is set, frames destined for this node are forwarded even if the corresponding RXblock bit for the port is set. • If NBLCK is not set, frames are forwarded only if the RXblock bit for the port is not set.
14	SECURE	Secured address flag. SECURE is used to change the security level for the address contained in the addnode register.
13	LOCKED	Locked address flag. LOCKED locks/unlocks the address contained in the addnode register on an add operation. Locked addresses output a discard code on the EAM interface.
12	CUPLINK	Copy frames to uplink flag. CUPLINK sets the copy uplink status for the address contained in the addnode register. Addresses tagged by this bit add the port (specified in the uplinkports register) to the routing code.
11–8	PORTCODE	Current port code for address. The value in PORTCODE becomes the port to which packets (sent to this address) are forwarded. In addresses that are added automatically to the table, the source port goes to this field, and the VLAN assigned to that port becomes part of the address (see principles of operation, SRAM allocation, lookup table SRAM allocation). This field is writeable, and can be used to redirect traffic. However, losing the VLANid to port synchronization results in leaky VLANs.
7	NEW	New-address indication. NEW changes the NEW field for the node contained in the addnode register. This address can now be found easily by a NEW find command.
6–0	Reserved	Writes to this location are ignored and read as 0.

If addnode is a multicast address, (addnode (40) = 1), addport is defined as follows:

BYTE 3								BYTE 2							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBLCK	PORTFLAG														
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
15	NBLCK	Not-blocked flag. NBLCK sets the RXblock register override for the multicast address contained in the addnode register. • If NBLCK is set, frames destined for this multicast address are forwarded even if the corresponding RXblock register bit for the port that originated the frame is set. • If NBLCK is not set, frames are forwarded only if the RXblock register bit for the port that originated the frame is not set.
14–0	PORTFLAG	Current PORTFLAG for multicast. PORTFLAG changes the port assignment for the multicast address contained in the addnode register. The bit values in this field correspond one-to-one with the TNETX3150/TNETX3150A port assignment. It is the user's responsibility not to put a value in this field that would cause leaky VLANs.

management delete address interface group at 0x2C, 0x48–0x4F (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
			Adddelcontrol	0x2C
				0x30
				0x34
				0x38
				0x3C
				0x40
				0x44
Delnode23–Delnode16	Delnode31–Delnode24	Delnode39–Delnode32	Delnode47–Delnode40	0x48
Delport	DelVLANID	Delnode7–Delnode0	Delnode15–Delnode8	0x4C

The management delete address interface is used with the DEL bit in the adddelcontrol register to perform management deletions from the lookup table.

lookup table add/delete command register, adddelcontrol at 0x2C (DIO), byte 0

BYTE 0							
BIT							
7	6	5	4	3	2	1	0
Reserved				DELP	DELV	ADD	DEL
Initial Values After Reset							
0	0	0	0	0	0	0	0

BIT	NAME	FUNCTION
7–4	Reserved	Writes to this location are ignored and are read as 0.
3	DELP	Delete all addresses on port. When DELP is asserted, the TNETX15VE deletes all addresses whose port assignment matches the delport register. DELP remains asserted until the delete process is complete. A DELP operation does not delete multicast addresses or addresses that have been marked as secure. This bit can be asserted at the same time as the DELV bit but not if the DELV bit is previously set (writing a 0x0C is permitted, but not 0x08 if the DELV bit is previously a 1). Asserting both bits kills all addresses that meet both conditions.
2	DELV	Delete all addresses on VLAN. When DELV is asserted, the TNETX15VE deletes all addresses whose VLAN assignment matches the delVLANID register. DELV remains asserted until the delete process is complete. A DELV operation does not delete multicast addresses or addresses that have been marked as secure. This bit can be asserted at the same time as the DELP bit but not if the DELP bit is previously set (writing a 0x0C is permitted, but not 0x04 if the DELV bit is previously a 1). Asserting both bits kills all addresses that meet both conditions.
1	ADD	See the add process bit under *lookup table add/delete command register*
0	DEL	Address delete. When DEL is asserted, the TNETX15VE uses the information contained in the management delete address interface to delete an address from the lookup table. DEL remains asserted until the delete process is complete.

delete-node address register, delnode at 0x48–0x4D (DIO)

The delnode register is used with the DEL bit in the adddelcontrol register to allow for management deletion of an address in the lookup table. To delete an address, the address to be deleted is placed in Ethernet native data format in this register, the VLAN in the delVLANID register, and the DEL bit is asserted. This register defaults to 0x00.00.00.00.00.00 after reset. When the DEL state machine is operating, this register is locked and no writes are performed to it.

delete VLANID register, delVLANID at 0x4E (DIO), byte 2

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved						VLANID	
Initial Values After Reset							
0	0	0	0	0	0	0	0

When the DEL bit in the adddelcontrol register is set, the delVLANID register contains VLANID code for the node contained in the delnode registers. When the DELV bit is set, all addresses corresponding to this VLAN are deleted. In this case, the delnode and delport registers are don't cares.

This register is read/writeable. When performing a DEL or DELV operation, this register is locked and no writes are made to it. A DELP operation does not lock this register.

delete port register, delport at 0x4F (DIO), byte 3

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved				Delport			
Initial Values After Reset							
0	0	0	0	0	0	0	0

The delport register is used with the DELP bit in the adddelcontrol register to delete all addresses assigned to this port. In this mode, the delnode and delVLANID registers are don't cares. This register is read/writeable. This register is locked when in a DELP operation. No writes are made to this register while in the locked state. A DEL or DELV command does not lock this register.

VLAN registers

The VLAN registers are used to link individual ports together into VLANs. With VLANs active, if a packet needs to be broadcast, the state machine must know which ports are included in the sourcing port's VLAN (VLAN routing mask registers). To enforce VLANs after setup, it is important to identify the VLAN of the incoming source address because the VLAN is considered part of the address during lookup (port VLAN assignment registers). To do this, the state machines need a table entry for each port that identifies which VLAN it is part of, and a table of vectors (or masks of bits) that indicates which stations are included in that VLAN. The other table in this group is the port 00 tag VLAN array. This array supports VLANs across a cascaded port 00 by allowing the state machines to identify individual ports across the cascade to a second TNETX3150/TNETX3150A device.

With these tables set up and coordinated, nonleaky VLANs are created. Broadcast/multicast packets are delivered only to the stations in the same VLAN as the originating station. The vector associated with the broadcast/multicast address is ANDed with its VLANID mask, has the source port removed, and is ANDed with the inverse of the TX block register (no forwarding to blocked ports) to determine which ports transmit the packet. Broadcast/multicast traffic can even be confined to ports known to contain servers by excluding bits in the VLAN routing mask registers that contain only clients. Unicast traffic is confined to the destination station address on the same VLAN. It is the designer's task to ensure that these tables are coordinated if nonleaky VLANs are needed. No checks are made to see that VLAN routing mask registers and the port VLAN assignment registers are essentially reciprocals of each other.

If VLANs are not supported, leave the port VLAN assignment registers, and the port 00 tag VLAN array set to their default of 0x0, and the VLAN routing mask registers set to their default of 0x7FFF. This effectively creates one VLAN with all stations included.

If VLANs are supported to other stations in systems linked through port 00 of the TNETX3150/TNETX3150A (to which this TNETX15VE is connected), the index into the port 00 tag VLAN array is the pretag field on the inbound packet on the TNETX3150/TNETX3150A port 00. This value is passed to the TNETX15VE as the frame data is stored to DRAM. If VLANs are supported across port 00 of the attached TNETX3150/TNETX3150A, the presence of the pretag on port 00 is required. If VLANs are not supported, the pretag is unimportant only if all the entries in the port 00 tag VLAN array are left at their default value of 0. Thus, any value that is clocked off the front of the packet yields an index at 0 to the routing mask register.

VLAN routing mask registers, VLANmask at 0x50–0x6F (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
VLANmask1		VLANmask0		0x50
VLANmask3		VLANmask2		0x54
Reserved				0x58
Reserved				0x5C
Reserved				0x60
Reserved				0x64
Reserved				0x68
Reserved				0x6C

Each VLANmask register corresponds one-to-one to a VLAN value. The TNETX15VE currently supports four VLANs, but space is reserved for future inclusion of 16 VLANs. The VLANmask register has the following format:

BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	VLANmask														
Initial Values After Reset															
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The VLANmask registers are used in two ways.

- When an address is not found, the VLANmask register (corresponding to the VLAN assigned to the source port) is used as an unknown routing mask. Frames are forwarded to the ports specified in the registers.
- When a multicast address is found in the lookup table, the lookup table port bit map is masked using the VLANmask register corresponding to the VLAN assigned to the source port. The VLANmask masks destination ports on a VLAN. The vectors for each VLAN allow the user to restrict broadcast traffic on that VLAN to only the ports in that VLAN that require it.

port 00 tag VLAN array, tagVLAN at 0x70–0x77 (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
TagVLAN6/TagVLAN7	TagVLAN4/TagVLAN5	TagVLAN2/TagVLAN3	TagVLAN0/TagVLAN1	0x70
TagVLAN14/TagVLAN15	TagVLAN12/TagVLAN13	TagVLAN10/TagVLAN11	TagVLAN8/TagVLAN9	0x74

The format for the tagVLAN register is (x/y):

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved		VLANID – Offset x		Reserved		VLANID – Offset y	
Initial Values After Reset							
0	0	0	0	0	0	0	0

The tagVLAN register array is used with the tag field in the forward pointer portion of the DRAM interface to chose the VLANID. This array is only valid when a frame comes in from port 00 (uplink).

Each byte contains two VLANIDs, one for each offset (x/y) specified in the name. All VLAN IDs default to 0x0 on a reset. These registers are automatically loaded from the EEPROM in a hardware reset ($\overline{\text{RESET}}$ = 0 or when DIO address high is 0x40) or when the LOAD bit in the control register is set.

VLANIDs can be interpreted in several ways. In a cascaded situation, the VLANID is a copy of the pretag that the other system put in front of the packet, and it is used to get the VLANID of the source port for the packet as it entered the remote system. The value that is prepended to the packet is used as an index to this table to retrieve the VLANID. In this case, VLANID15 is not likely to be used because the TNETX3150/TNETX3150A cannot have a packet sourced from a port that does not exist.

If this system is connected to a larger system, then the value prepended to the packet is an external indication of the VLAN for the incoming packet. Functioning as a pointer to this table, any value in the 4-bit field is allowed. The TNETX15VE device only supports four VLANIDs in this array (two least-significant bits of a 4-bit field), and any value in the pretag field can be used to index to the table. For this reason it is important to fill in all the values in the table unless the higher entity can be restricted to four values of the pretag for four supported VLANIDs. Future versions of the address-lookup engine might support all four bits or 16 VLANs.

If port 00 is connected to an entity that does not control the pretag field, VLANs cannot extend out of this TNETX3150/TNETX3150A and TNETX15VE combination. If the packet comes from port 00, the value clocked off the wire is used as an index to this table. All the table entries must represent the VLANID assigned to port 00 because any value can be clocked off the wire.

port VLANID assignment registers, VLANID at 0x78–0x7F (DIO)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
			VLANID1	0x78
				0x7C
			VLANID2	0x80
				0x84
			VLANID3	0x88
				0x8C
			VLANID4	0x90
				0x94
			VLANID5	0x98
				0x9C
			VLANID6	0xA0
				0xA4
			VLANID7	0xA8
				0xAC
			VLANID8	0xB0
				0xB4
			VLANID9	0xB8
				0xBC
			VLANID10	0xC0
				0xC4
			VLANID11	0xC8
				0xCC
			VLANID12	0xD0
				0xD4
			VLANID13	0xD8
				0xDC
			VLANID14	0xE0
				0xE4

The VLANID registers are used to assign a particular VLAN to ports 01 through 14. Eight bytes are reserved for each port. Their format for the VLANID registers is:

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved						VLANID	
Initial Values After Reset							
0	0	0	0	0	0	0	0

All VLANIDs default to 0x0 on a reset. These registers are automatically loaded from the EEPROM in a hardware reset ($\overline{\text{RESET}}$ = 0 or when DIO address high is 0x40) or when the LOAD bit in the control register is set.

monitor registers

In addition to the NMON monitoring provided by the TNETX3150/TNETX3150A, using the TNETX15VE adds software-configurable modes of monitoring traffic, which are described as follows.

Address monitoring: Unicast traffic is sent to a second port regardless of VLAN by modifying the table entry in the TNETX15VE SRAM memory using the edit function on the ADD command. If the table entry is modified to set the CUPLINK bit, any packet destined for that address is queued for transmit on the port specified by uplinkport. Multicast addresses do not have such a bit in their entry, but multicast addresses can include any port as a monitoring port by changing the portflag vector to include a new port that is subject to that vector being ANDed with that port's VLANmask register. Both of these modes require an external host to perform the modification to the address table value. Multiple addresses can be monitored at the same time. Both sides of a suspect transaction are monitored by setting the CUPLINK bit for both destination addresses that are monitored. All the traffic going to the destination addresses (so marked) is forwarded to the port specified in the uplinkport register.

Port monitoring: All traffic to or from a specified port is copied to another port of choice. The port to be sampled is specified in the mirror port-select register; the output goes to the port selected in the uplinkport register, and the activity is gated on and off by setting the MIRR bit in the control register (0x8, 0x9). The traffic includes packets that would not normally be routed through the TNETX3150/TNETX3150A (when both the source and destination addresses are on the same port, the inbound packet is normally dropped). However, a second copy of the packet is not generated if not necessary. For example, a packet inbound on the uplinkport (destined for the mirrorport) is not copied back to the uplinkport because it has already been seen there.

Both port and address monitoring can be active at the same time, but only one destination port is used at one time; both monitoring modes direct traffic to the port specified in the uplinkport register (be careful in setting up extensive monitoring). If the streams being monitored are running in full-duplex mode, it is possible to queue more traffic per second to the monitoring port than can be transmitted there. When the transmit queue at the output port fills up, packets are dropped.

uplink routing register, uplinkport at 0xF0 (DIO)

BIT							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved				Uplink			
Initial Values After Reset							
0	0	0	0	0	0	0	0

The uplinkport register contents are used in both port monitoring and address monitoring to specify the output port for the data being monitored. The 4-bit field uplink specifies the destination port number for monitored frames. The Uplink field is read/write. Changing uplink changes the port that subsequent packets are queued to. Changing uplink does not move frames queued to the previous port value.

mirror port select register, mirrorport at 0xF2 (DIO)

BIT							
7	6	5	4	3	2	1	0
Reserved				Mirrorport			
Initial Values After Reset							
0	0	0	0	0	0	0	0

The mirrorport register selects the TNETX3150/TNETX3150A port that is monitored when the MIRR bit in the control register (0x8, 0x9) is on.

blocking registers

The blocking registers and the CPUTX register support the various states of the spanning-tree algorithm.

The TXblock register allows the host to stop packets from being queued for output (forwarded or transmitted) on a port-by-port basis. Any port with its corresponding bit set does not have any frames added to its transmit queue, except by the transmit-override mechanism. The CPUTX register is used to activate the transmit-override mechanism. When the TX routing code is not 0xF, the next frame from the CPU port is queued and transmitted on the TX routing code port, regardless of the state of the TXblock bit for that port. The CPU port value specifies the source port of a specially treated packet, and the TX routing code value specifies the destination port of a specially treated packet.

The RXblock register allows the host to not accept packets on a port-by-port basis. The bit numbers correspond to the port being blocked. Packets that arrive from any station on a port with its corresponding bit set are ignored. They are not counted in any of the statistics counters. This barrier is overridden on an address-by-address basis by setting the NBLCK bit (bit 15 of the addport register) via an ADD/EDIT operation. Addresses with this bit set are received normally.

Using the CPUTX register to queue a frame for transmission bypasses the normal TNETX15VE table lookup on the destination address, and overrides the TXblock barrier to transmit. This mechanism can be used at any time to steer one packet from the CPU port to the TX routing code port. The CPUTX register is reset to 0xF, a nonfunctional value, after one packet is handled under its influence.

The CPUTX register is used to reply to received spanning-tree BPDU packets. The multicast address port vector for BPDUs is set to point only to the port where the management entity was connected – this would ensure that the BPDUs are sent to the right place. The NBFR bit (bit 5) of the findcontrol register (0x16) would be used to discover the source port for each inbound BPDU. When the reply is composed, the CPUTX register is used to steer the packet to the right port, past the TXblock barrier.

forwarding block register, TXblock at 0xF4–0xF5 (DIO)

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	TXblock														
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The TXblock register blocks forwarding (transmitting) on ports that have their corresponding bit set. Packets destined for ports that are forwarding blocked are discarded. The TXblock register can be overridden on one frame at a time using the CPUTX registers.

receive block register, RXblock at 0xF6–0xF7 (DIO)

BYTE 1								BYTE 0							
BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	RXblock														
Initial Values After Reset															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The RXblock register blocks frame reception, depending on the port from which the frame originated. Frames arriving on ports whose corresponding bit is set to on1 are discarded. The RXblock register can be overridden if the packet destination address has the NBLCK bit set to 1.

attached microprocessor's define/routing register, CPUTX at 0xF8 (DIO)

BIT							
7	6	5	4	3	2	1	0
CPU PORT				TX ROUTING CODE			
Initial Values After Reset							
0	0	0	0	1	1	1	1

BIT	NAME	FUNCTION
7–4	CPU PORT	Attached microprocessor port assignment. CPU PORT specifies the port where the CPU is attached. Code 0xF is reserved and is ignored.
3–0	TX ROUTING CODE	Transmit routing code. TX ROUTING CODE specifies where frames coming from the CPU port are routed. This field has the following behavior: • If this field is 0xF, the CPU routing function is disabled and frames are routed according to the TNETX15VE lookup table. • If this field is not 0xF, frames that come from the port specified in CPU PORT are routed to the port encoded in TX ROUTING CODE, by passing any routing from the destination address lookup. After the frame has been transmitted using this register, it disables itself by writing a 0xF to this field.

The CPUTX register allows the programmer to specify where frames coming from the CPU should be forwarded. The CPU port is specified in the CPU PORT field and frames are routed to the port specified in the TX ROUTING CODE. While useful in replying to BPDUs in spanning tree when ports are normally blocked, this mechanism can be used at any time to specifically route one packet.

PRINCIPLES OF OPERATION

The purpose of this section is to assist in the development of management operational code for the TNETX15VE. Management operational code is drastically simplified when using the TNETX15VE since it is designed to operate without the need for a management CPU. Many of the functions that previously were performed by a CPU are now integrated on the TNETX15VE.

This section discusses the following topics:

- A background on the TNETX15VE architecture, including the data-storage algorithm and the arbitration between logic operations
- How to access the TNETX15VE registers through the DIO interface, and how these registers can be used to access the internal/external SRAM, the EEPROM, and any MII-managed devices
- How to access the TNETX15VE through the DIO interface and through a hardware reset, and what steps are needed to bring the TNETX15VE to an operational state
- How to use the logic operations to perform management-based lookups, adds, and deletions, and how to set the forwarding option for VLANs.
- How to use the TNETX15VE interrupt support to create event-driven management operational code.

internal operation

This section describes the TNETX15VE hardware and how it affects the programmer and the TNETX3150/TNETX3150A performance. The functionality described in this section is transparent to the user, but it is included to help the user become familiar with the TNETX15VE architecture and how it interfaces to the TNETX3150/TNETX3150A device. Refer to the latest TNETX3150/TNETX3150A documentation for additional information.

EAM codings and in-order broadcasts (IOB)

The TNETX15VE uses two styles of EAM codings – single-port codes and multiple-port codings. The TNETX3150/TNETX3150A treats these two types of coding differently.

Single-port codings are those that forward frames to a single port. The TNETX3150/TNETX3150A queues these frames to the port queue.

Multiple-port codings forward frames to multiple ports. The TNETX3150/TNETX3150A creates an IOB list structure to queue this frame to multiple-port queues. IOB lists use more bandwidth than a regular list because IOB lists require the use of an extra 64-byte buffer to contain all queue pointers for other ports. If 64-byte frames are sent to multiple ports, IOB structures can then use much of the TNETX3150/TNETX3150A's available bandwidth.

The TNETX15VE uses single-port codings when possible, to maximize performance. The TNETX15VE inspects all multiple-port codings after masking by the VLAN vector of source port, removal of the source port, and removal of TX-blocked ports to determine if they can be replaced by a single-port coding. If this is possible, then The TNETX15VE outputs the single-port coding.

For a more complete description of IOB lists, refer to the latest TNETX3150/TNETX3150A specification.

PRINCIPLES OF OPERATION

TNETX15VE DRAM and EAM interface

The TNETX15VE takes its frame input through the TNETX3150/TNETX3150A DRAM bus. It must recognize a start-of-frame indication (SOF) on the first flag byte of the frame. After the SOF is found, the TNETX15VE latches the first 32 bits of the destination address on the next DRAM cycle. From this time, it must complete a lookup cycle, decide on the appropriate EAM code, and output this code in 440 ns, or less. Figure 2 shows the lookup timing. Check the data set for the TNETX3150/TNETX3150A for field definitions beyond those used by the TNETX15VE.

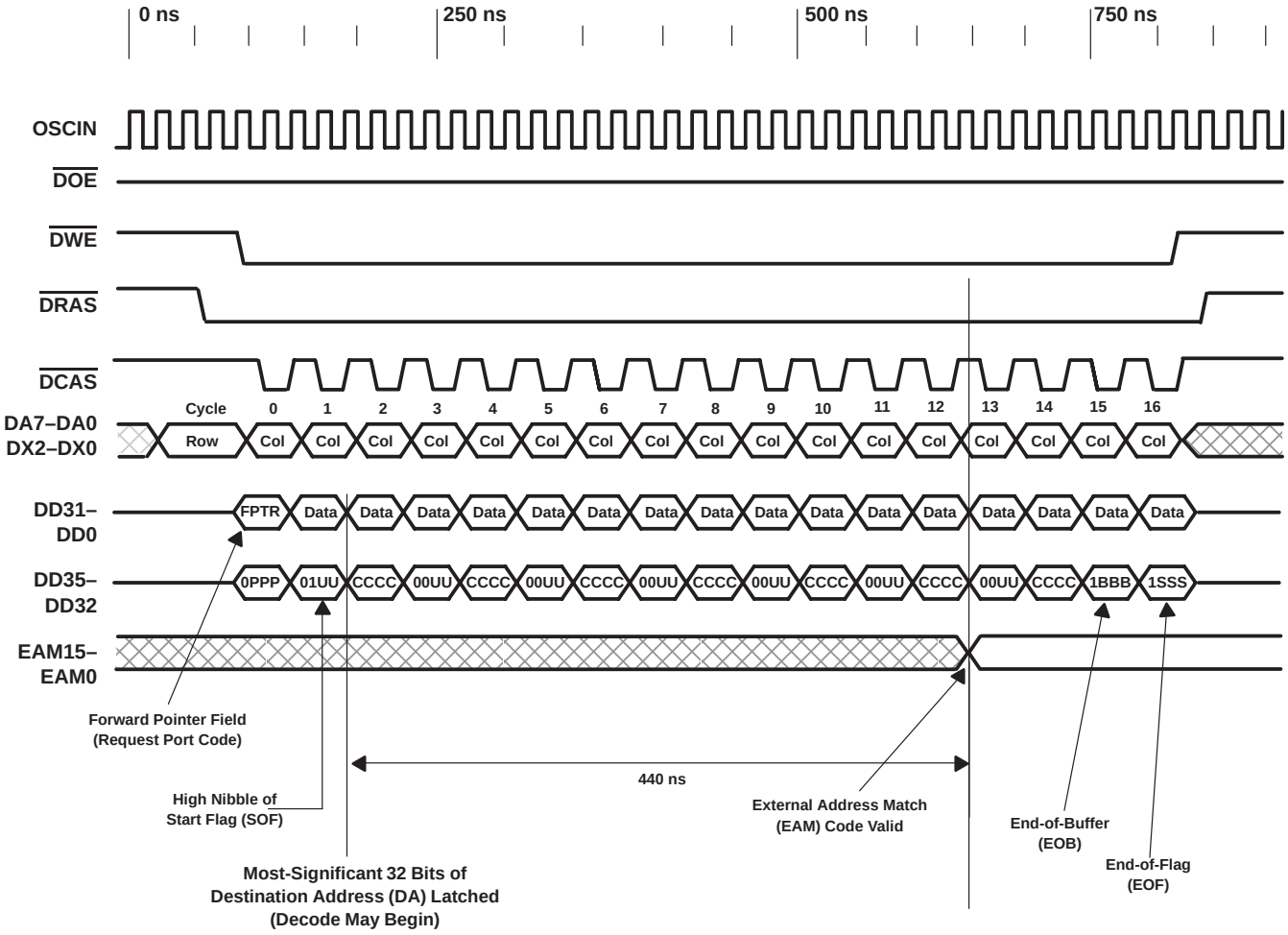


Figure 2. Lookup Cycle Timing

The data on each cycle has the following format. The TNETX15VE must determine that the data being written to memory is the first buffer of a data frame and not an IOB index buffer. The buffer contains packet data if the IOB bit is 0. The port number and VLAN assignment for the frame is latched from the channel code. The channel code is the source port.

PRINCIPLES OF OPERATION

TNETX15VE DRAM and EAM interface (continued)

The TNETX15VE must determine the start of frame by looking at the flag in cycle 1. The flag is given in the DD35–DD32 terminals. The SOF is shown in the following table in cycle 1 at bit (35–34) = 01b.

CYCLE	BIT															
	35	34	33	32	31	28	27	24	23	16	15	0				
0	IOB	Parity			Tag		Channel code		Forward pointer†							
1	0	1	Reserved		Most-significant 32 bits of destination address (DA)											
2	Channel code				Most-significant 16 bits of source address (SA)						Least-significant 16 bits of DA					
3	0	0	Reserved		Least-significant 32 bits of SA											
≈ ⋮ ≈	Flags			≈	Data											≈
N–1	EOB	Valid bytes			Data											
N	EOF	Frame status			CRC											

† These bits are ignored by the TNETX15VE.

The TNETX15VE cycle begins when it latches the partial destination address to begin the table lookup, and ends when it outputs an EAM code within the allocated 440 ns after the SOF condition is met.

The channel-code field in the flags of the body of the frame is exactly the same as that in the forward pointer. The TNETX15VE uses the channel code from the forward pointer.

tag field

The tag field in the forward pointer comes from the port 00 pretag; that is, this field is defined only if the packet source port is 00. Figure 3 shows the tag in relation to the rest of the frame. Please consult the latest TNETX3150/TNETX3150A documentation for additional information.

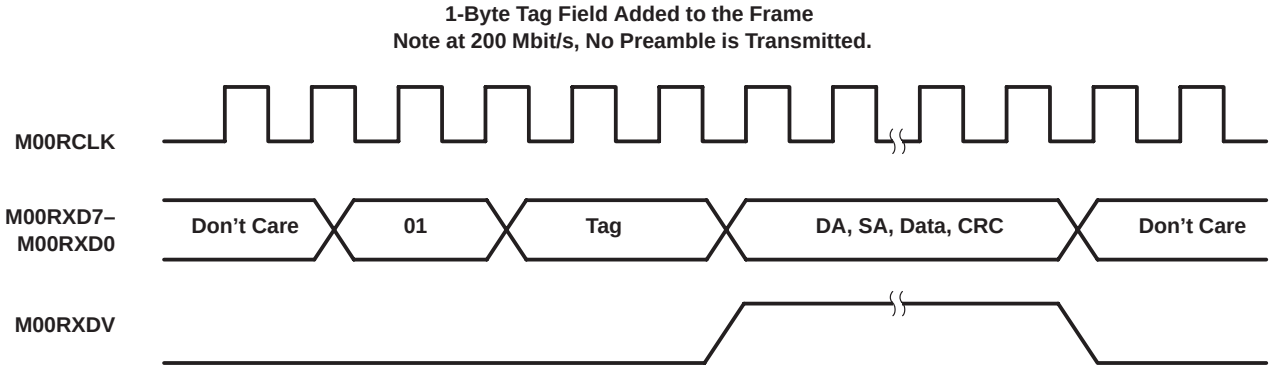


Figure 3. Field Format for Uplink-Port Receive (RX) Frames

The pretag byte format is as follows:

BIT							
7	6	5	4	3	2	1	0
Reserved				Source port number			

PRINCIPLES OF OPERATION

tag field (continued)

The intent of the tag field is to allow each TNETX15VE and TNETX3150/TNETX3150A combination in a cascaded situation through port 00 to know the sourcing port on the other side of the cascade connection. With this visibility, VLANs can be enforced with stations on both sides of the cascade connection. The pretag field also can be used to support VLANs if a TNETX15VE/TNETX3150/TNETX3150A combination is functioning as an edge device through port 00 for other equipment. By putting a tag in front of the packet, the TNETX15VE gets a VLANID from a table indexed by the tag field. The TNETX3150/TNETX3150A is responsible for recovering the pretag and including it in the first block transfer for a packet being written to memory during cycle 0. The tag field is used only by TNETX15VE if the channel code is 00. Interpretation of the tag is shown in Table 2.

Table 2. Source Port Number Codes

SOURCE PORT NUMBER	PORT
0000	Reserved
0001	Port 01 (10/100 Mbit/s)
0010	Port 02 (10/100 Mbit/s)
0011	Port 03 (10 Mbit/s)
0100	Port 04 (10 Mbit/s)
0101	Port 05 (10 Mbit/s)
0110	Port 06 (10 Mbit/s)
0111	Port 07 (10 Mbit/s)
1000	Port 08 (10 Mbit/s)
1001	Port 09 (10 Mbit/s)
1010	Port 10 (10 Mbit/s)
1011	Port 11 (10 Mbit/s)
1100	Port 12 (10 Mbit/s)
1101	Port 13 (10 Mbit/s)
1110	Port 14 (10 Mbit/s)
1111	Reserved

PRINCIPLES OF OPERATION

decoding the destination address (DA) and source address (SA) from the DRAM bus

The TNETX3150/TNETX3150A writes data to the DRAM buffers in a specific format (refer to the TNETX3150/TNETX3150A data sheet for additional information on this format). The TNETX15VE recognizes this format to determine the correct destination and source addresses. The format for the DA and the SA is shown in the following:

CYCLE	BIT			
	31 25	24 16	15 8	7 0
1	DA23–DA16	DA31–DA24	DA39–DA32	DA47–DA40
2	SA39–SA32	SA47–SA40	DA7–DA0	DA15–DA8
3	SA7–SA0	SA18–SA8	SA23–SA16	SA31–SA24

The node addresses are transmitted on the DRAM bus in the Ethernet native data format. This is the same format in which the address appears in the frame. Ethernet first transmits the least-significant bit of a data byte on the wire. This makes the specific/group bit appear in bit 40 (least-significant bit of most-significant byte). For example, a destination address of 0x12.34.56.78.9A. BC and a source address of 0xDE.F1.23.45.67.89 is seen on the DD terminals as follows:

CYCLE	BIT			
	31 25	24 16	15 8	7 0
1	0x78	0x56	0x34	0x12
2	0xF1	0xDE	0xBC	0x9A
3	0x89	0x67	0x45	0x23

CRC checking and valid frames

The TNETX15VE determines the status of the frame when the end of buffer (EOB) followed by an end of frame (EOF) is detected. CRC checking is determined from the frame-status field. The code for a good CRC is frame status = 000b. All other frame-status codings indicate that the TNETX3150/TNETX3150A aborts the frame due to either a CRC error, a FIFO overflow, or a network error. The frame status is checked when the option of adding addresses to TNETX15VE tables on a good CRC is indicated rather than immediately.

operating logic arbitration

The lookup table is contained in both the external and internal SRAM. All of the TNETX15VE logic operations share access to this SRAM. An arbitration scheme is implemented to give all logic operations fair access to the SRAM while meeting the lookup-timing requirements.

The TNETX15VE contains seven logic operations that require the use of the SRAM bus. They are: the RAM initialization logic operation (INIT), the lookup logic operation (LKUP), the delete logic operation (DEL), the add logic operation (ADD), the management address-lookup logic operation (FIND), the RAM registers RAMaddr and RAMdata (REG), and the aging logic operation (AGE).

The arbiter assigns a priority to each logic operation. The highest priority is assigned to the INIT logic operation to initialize the TNETX15VE after a reset. After initialization, LKUP becomes the logic operation with the highest priority. LKUP has the highest priority on the bus because it is the logic operation that is the most time critical. The next priority level is shared by ADD and DEL. Register-based accesses (REG) are followed by the FIND logic operation. AGE becomes the lowest priority. Figure 4 shows the priorities of the TNETX15VE logic operations.

PRINCIPLES OF OPERATION

operating logic arbitration (continued)

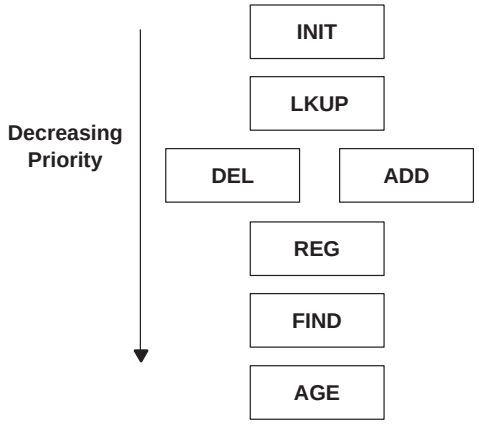


Figure 4. Logic-Operation Priorities

The arbiter grants the bus to the logic operation with the highest priority that is currently requesting the bus. Each logic operation requests the bus by asserting its request signal. The arbiter assigns the bus to that logic operation by asserting its grant signal. If no logic operation is requesting the bus, the arbiter grants the bus to AGE for background-aging operations.

One logic operation can interrupt a lower-priority logic operation to acquire the bus. For example, an LKUP operation interrupts an ADD operation.

For the case of ADD and DEL (with the same priority), the arbiter grants the bus to the first logic operation that requests it. It then grants an uninterruptable bus (unless by a LKUP) to that logic operation until that logic operation is completed. If both ADD and DEL request the bus at the same time, the bus is granted to ADD. This ensures that ADD is not interrupted by a DEL operation and vice versa. This hierarchy explains why a host request for access to RAM completes with some variability — there may be higher-priority operations accessing RAM at that instant.

lookup algorithm

The TNETX15VE uses a table-based lookup algorithm to provide deterministic lookups with less than 2^{48} memory words. The tables are hierarchical and are linked to the lower tables by threads. Each table can thread to several different tables in the hierarchy. The lowest table in the hierarchy (leaf) does not point to anything and contains information about the address to be matched.

Each level in the hierarchy is assigned to a specific range of bits in the address. The bits in the range are used as an offset within the table at each level. If a thread exists at that offset, the TNETX15VE follows that thread. The TNETX15VE matches an address when it finds a complete thread to a leaf. The thread structure is shown in Figure 5.

PRINCIPLES OF OPERATION

lookup algorithm (continued)

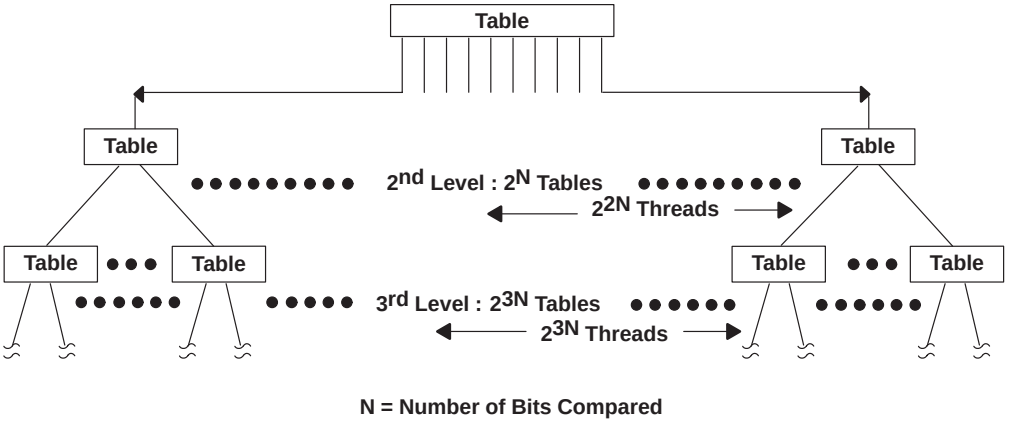


Figure 5. Lookup-Algorithm Table Hierarchy

The first level (root level) has only one table from which it can branch out to 2^N possible tables. N is the number of bits compared. Each additional table in the hierarchy branches down to 2^N other possible tables. The second level contains 2^N table and 2^{2N} threads. The third level contains 2^{2N} tables and 2^{3N} threads, and so on.

Because of this exponential growth, the threads and the amount of possible paths at each level soon overtake the number of addresses required. If this growth was unchecked with an N of 5, the third level would contain 1,024 tables and 32,768 threads. If only 1024 addresses are required, there are more tables allocated than needed, and most contain NULL pointers.

For bit groups of five (give table sizes of 32), and with five levels of table hierarchy, preassigning all possible tables at initialization requires more RAM than is possible to attach to the TNETX15VE. Therefore, table allocation occurs only when a new table is required as a result of discovering a bit pattern at any of the levels previously unseen. This has several side effects – given the same address set, entering them into the table in a different order yields different SRAM contents; pointer values are written to the table entries to point to the next level in the hierarchy since it is dynamically allocated. An address is in the table if there are valid entries at each level in the hierarchy (as each n-bit part of the address is inspected) that continue all the way down to a leaf where the characteristics of that address are stored. An address is not present in the table when there is a 0 pointer at any level of the decode. Addresses that are sequential share all the intermediate pointers down to the address of the leaf. Addresses with at least one bit different in each n-bit inspection unit occupy completely different tables. The worst-case rating on the storage capacity of the TNETX15VE SRAM assumes that each address is different from all the others by at least one bit in each inspection unit. This implies that each card is from a different manufacturer, and no serial number of any card is within a table size of any other card.

PRINCIPLES OF OPERATION

lookup algorithm (continued)

Since each table needs to compare 2^N possible combinations, it requires 2^N pointers. Each table has the following format, assuming 16-bit wide memory:

OFFSET	N (Bits to be compared)	16-BIT POINTERS
0	00000	To table x
1	00001	To table y
2	00010	To table z
⋮	⋮	⋮
$2^N - 1$	$2^N - 1$	

Only the pointers column of the above table occupies memory locations.

Each pointer points to a table in the next level. The TNETX15VE uses N bits in the address as an offset to this table and, if a pointer is found, it uses it to go to the next level. A zero pointer indicates that the entry was not found (in this case the search fails).

a graphical example of the lookup algorithm

This example uses the following method to look up the number 0xB2 (10.11.00.10b), two bits at a time (N = 2). Graphically, this number is represented in Figure 6.

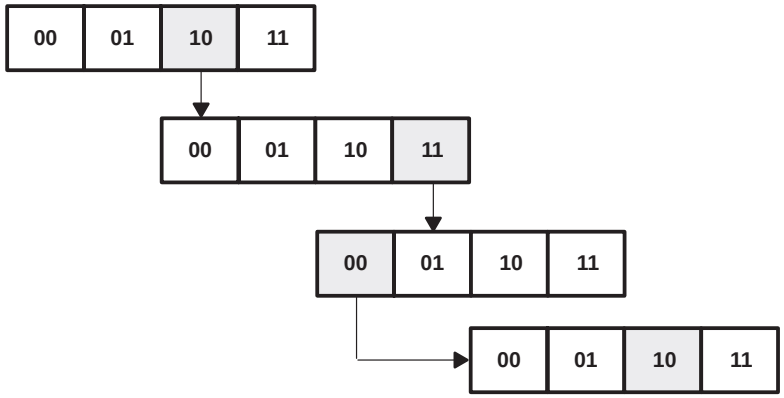


Figure 6. Example of Lookup-Number Algorithm (Matching 0xB2)

The first table is offset 0x10b points to the second level (see Figure 6). The second level uses the second set of bits (0x11b) and points to the third table. This process continues until the last two bits are matched. Matching 0xB2, two bits at a time, uses four tables with each containing four possible pointers. Not all locations in the tables are used, which potentially can lead to unused memory. Now, add 0xB0 (10.11.00.00b) to the table. Figure 7 illustrates the results.

PRINCIPLES OF OPERATION

a graphical example of the lookup algorithm (continued)

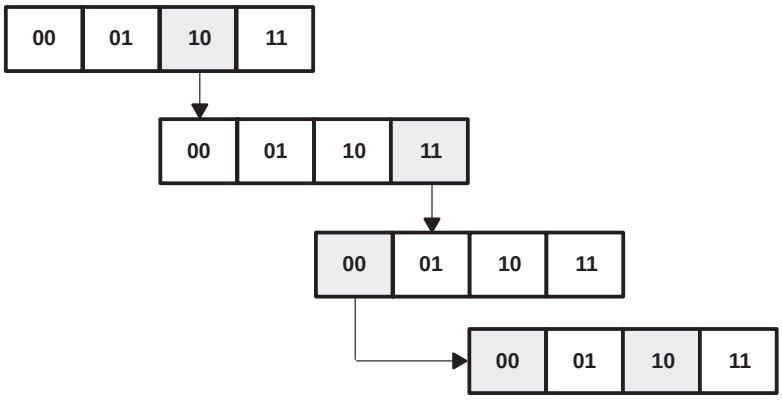


Figure 7. Example of Lookup-Number Algorithm (Adding 0xB0)

0xB0 follows exactly the same thread as 0xB2 and the only difference between the two is in the last table. 0xB0 matches offset 00b while 0xB2 matches offset 10b. There are now two numbers being represented, but there are still the same number of tables allocated (four).

Continuing this example, we can add 0xB1 and 0xB2 with the same number of tables allocated. This is called the best-case scenario since the maximum amount of addresses can be stored in the minimum amount of memory. Now add 0x22 (00.10.00.10b) to a lookup table containing only 0xB2. Figure 8 shows the results.

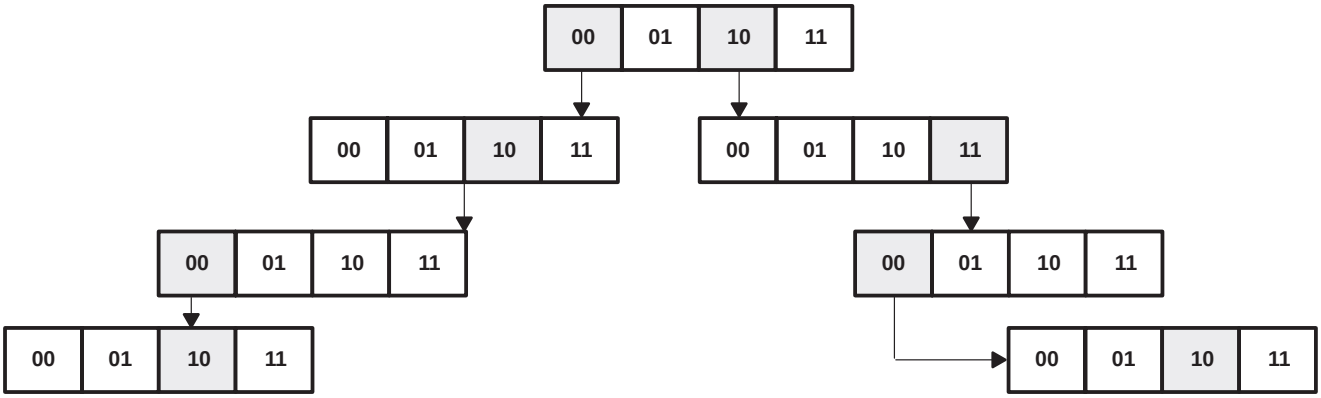


Figure 8. Example of Lookup-Number Algorithm (Adding 0x22)

Adding 0x22 requires allocating three additional tables. Seven tables are now required to hold two addresses. Compared to numbers that differ in their least-significant bits, numbers that differ in their most-significant bits require more tables. Continuing the example, adding 0x2A requires an additional three tables, as would 0xE2. This is the worst-case scenario, and it is the least-efficient way of storing addresses.

The TNETX15VE is designed to handle the worst-case address distribution. The worst-case address distribution requires a separate thread per address. A purely random distribution creates multiple threads at the top levels. The most-significant bits of a network card address are the vendor bits. However (in most networks) there are only a couple of vendors used. These cards do not have a purely random address distribution, and they all share a common set of bits that identifies the vendor. This configuration requires fewer pointers for the same number of addresses. In such a network, the tables look more like those in Figure 9.

PRINCIPLES OF OPERATION

a graphical example of the lookup algorithm (continued)

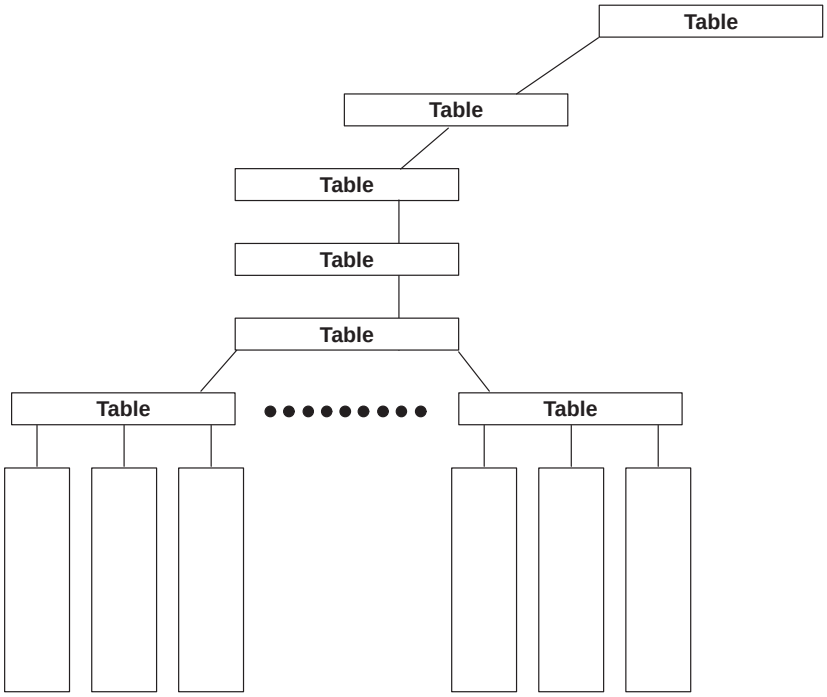


Figure 9. Address Distribution Table Hierarchy (Worst Case)

There is a need to allocate for worst case, but since the worst case is not likely to happen in a real system, the opportunity exists to include more addresses than indicated by the worst-case rating. The actual number of addresses supported depends on the nature of the nodes in the network. TNETX15VE devices in networks with nodes from one or a small number of manufacturers are able to recognize more addresses than those in a purely random-address network.

This algorithm has the primary advantage that the lookup time is independent of the number of addresses stored in the lookup table. Whether the number of addresses present is one or a million, the lookup time depends on the number of levels required to match the address.

Another side effect of this algorithm is that the table order is already sorted (smallest to largest).

The choice of N (the number of bits to consider each cycle) is a balance of:

1. The size of each table — it must be 2^N words.
2. The clocks required to process a whole address (in this case 48 IEEE addresses plus two VLAN bits) are $50 \div N$.

PRINCIPLES OF OPERATION

SRAM data storage

internal SRAM allocation

The internal 4K x 16 SRAM is divided into two 2K x 16 units. Data can be fetched from both blocks at the same time using a common address. The lower addresses in each page are devoted to the FIFO memory structures. The bus-watcher FIFO stores source addresses of packets for checking to see if that address needs to be added to the table after the destination address is looked up to route the packet. The NBLCK FIFO primarily stores the BPDU packet source address and source ports during spanning-tree resolution. Each FIFO uses the address in both pages to speed storage and retrieval.

Starting with the 0x05E/0x85E pair, information about each address present in the table is stored. This is the leaf or end of the table structure mentioned in the previous section. The lower address contains the flag and port coding word (see definition of findport register, 0x12/0x13), and the upper address contains the age of that address (see definition of *findnodeage register*, 0x14/0x15).

The range of 0x5E to 0x7FF gives room for information on 1954 addresses that are stored here. All the intermediate tables are stored in the external SRAM. Internal RAM is faster, and storing the information about an address (table leaf) internally allows several operations that conclude an address lookup, such as looking up the VLANID bits, to be overlapped to ensure completion.

Internal SRAM allocation is shown in Figure 10.

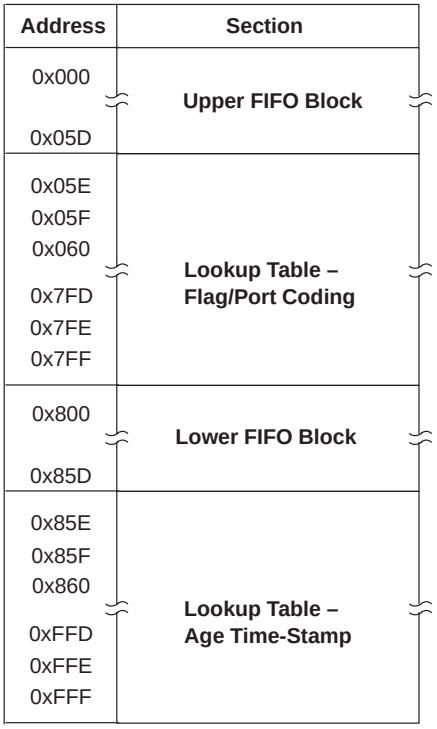


Figure 10. Internal SRAM Allocation

Three bytes of information are needed for each of the port's FIFO locations. The address-lookup table leaves are partitioned into two words for each address. The TNETX15VE places the routing codes and age time stamp in these two words. There are 1954 pairs of words that limit the address table to 1954 nodes.

PRINCIPLES OF OPERATION

lookup table SRAM allocation

The TNETX15VE uses a 5-bit version of the lookup algorithm described in the previous section. To meet our duplicate-address support goals, the TNETX15VE extends the lookup to 48 address bits plus two VLAN bits. In this way, frames with the same address (but different VLANs) are delivered to unique end stations.

When a 5-bit version of the lookup algorithm is used, each address requires 10 tables to store a 50-bit value. Tables 0 through 9 are used to match the 50 bits involved in the lookup. Table 9 (shown in Figure 11) contains a pointer to the internal SRAM where the node's data is contained.

The 16-bit-wide external SRAM is required for proper operation. A 16-bit pointer retrieved from the table coupled with the next five bits of address being inspected gives a 21-bit (2M) address capability. With the leaves of the table confined to the internal RAM, there is no value to having more than 512K $\times$ 16 of external SRAM attached to the TNETX15VE. The algorithm runs out of table leaves before running out of table space if more SRAM is installed.

Table 9 (in Figure 11) only needs an 11-bit address to reach the table leaf in internal SRAM (only 2K unique addresses). Since there is only one table-9 entry per active address in the table, some of the information for a unicast address is stored in the five bits left over. The corresponding bits for a multicast address are not used.

Each table entry requires 40 ns to access and 10 levels total 400 ns towards the lookup time. The internal SRAM data is accessed in 20 ns. The total time required (400 ns + 20 ns) gives some margin to generate the EAM code in time for the TNETX3150/TNETX3150A.

An allocated (but unused) table entry is set to 0 since 0 cannot be a valid address for a table leaf.

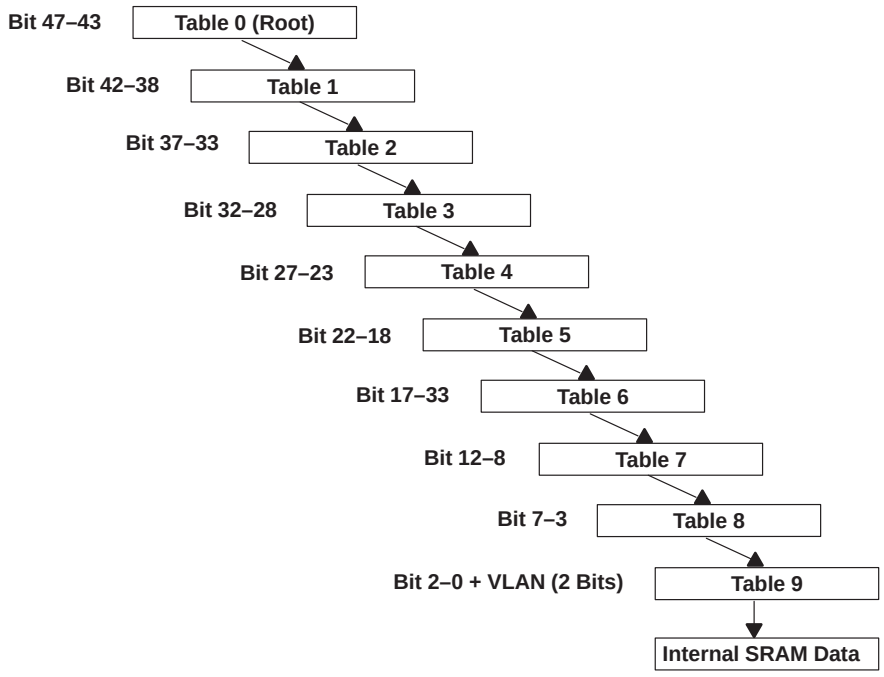


Figure 11. Address Tables

PRINCIPLES OF OPERATION

lookup table SRAM allocation (continued)

For unicast addresses, Table 9 (see Figure 11) is allocated as follows:

BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CUPLNK	Port code				Pointer to internal SRAM										

For a multicast address, Table 9 (see Figure 11) is allocated as follows:

BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved					Pointer to internal SRAM										

The internal SRAM allocation for unicasts is as follows:

WORD A	WORD B
Flags/port codings	Age-time-stamp

Word A for unicasts has the following definition:

BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBLOCK	SECURE	LOCKED	NEW	Reserved											

The internal SRAM allocation for multicasts is as follows:

WORD A	WORD B
Flags/port codings	Reserved

Word A for multicasts has the following definition:

BIT															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBLOCK	Port bit vector code														

PRINCIPLES OF OPERATION

determining the 2-bit VLANID for lookups and wire adds

The VLANID (used for lookup and wire-add operations) is the VLAN assigned to the source port. There are two different ways to determine the VLANID for the source port, and the method used depends on the source port number.

source port is 01 through 14

When the source port is 01–14, take the VLANID from the VLANID register for the source port [VLANDID 0x78–0x7F (DIO)].

source port is 00 (uplink)

If the frame comes from port 00, the tag field is used as an offset into the tagVLAN array [0x70–0x77 (DIO)]. The tagVLAN registers (16 total) support the four bits from the packet pretag field.

This offset method allows three possible cases:

- When two TNETX3150/TNETX3150As are connected through port 00 (current device is called TNETX3150/TNETX3150A No. 1 and the remote TNETX3150/TNETX3150A No. 2), the tag coming into TNETX3150/TNETX3150A No. 1 gives the source port for TNETX3150/TNETX3150A No. 2. If the tagVLAN register array is filled with the VLANIDs for the ports in TNETX3150/TNETX3150A No. 2, VLANs are transferred across port 00. This means that TNETX3150/TNETX3150A No. 1 tagVLAN array mirrors the VLANIDs of TNETX3150/TNETX3150A No. 2 and vice versa.
- When TNETX3150/TNETX3150A is connected to a proprietary integrated circuit (IC) through port 00, the third party IC uses the tag field to send across the VLANID of the frame. The TNETX15VE uses the tag as the offset to the tagVLAN register array. If the tagVLAN register array is filled with its offset number (see Table 3), the tag field corresponds to the VLANID.
- When port 00 is connected as an ordinary port (end equipment does not support pretagging), port 00's VLANID is put into all 16 locations of the tagVLAN register array. No matter what offset is latched during the pretag time, the correct VLANID is used.

PRINCIPLES OF OPERATION

Table 3. TagVLAN Registers With Offset Numbers

TAGVLAN REGISTER	OFFSET NO.
TagVLAN0	0x0
TagVLAN1	0x1
TagVLAN2	0x2
TagVLAN3	0x3
TagVLAN4	0x4
TagVLAN5	0x5
TagVLAN6	0x6
TagVLAN7	0x7
TagVLAN8	0x8
TagVLAN9	0x9
TagVLAN10	0xA
TagVLAN11	0xB
TagVLAN12	0xC
TagVLAN13	0xD
TagVLAN14	0xE
TagVLAN15	0xF

PRINCIPLES OF OPERATION

RAMsize and number of nodes supported

The address capability for the various RAM sizes is given in Table 4. The numbers are upwardly limited by the table leaves that can be stored in the TNETX15VE 4K x 16 internal SRAM. The minimum size of external SRAM that supports 1954 addresses (worst case) is 512K x 16 (RAMsize = 0x0B).

Table 4. Address Capability for RAM Sizes

RAMsize REGISTER	RAM SIZE	WORST CASE	BEST CASE
0x00	640 x 16	2	88
0x01	832 x 16	2	136
0x02	1K x 16	3	184
0x03	2K x 16	7	432
0x04	4K x 16	14	920
0x05	8K x 16	28	1912
0x06	16K x 16	59	1954
0x07	32K x 16	123	1954
0x08	64K x 16	251	1954
0x09	128K x 16	507	1954
0x0A	256K x 16	1019	1954
0x0B	512K x 16	1954	1954
0x0C	1M x 16	1954	1954
0x0D–0x0F	Reserved		

The data in Table 4 shows a large range between the worst-case performance and the best-case performance. For a 500-address system, 128K x 16 is needed. A 1-K address system requires 256K x 16. A 2-K address system requires 512K x 16. Adding more than 512K x 16 to the TNETX15VE does not increase the address storage capacity since it exceeds the internal SRAM limit.

The difference between the two numbers at each row depends on the similarity (when viewed in 5-bit groups) of the addresses entered into the table. If addresses are sequential, there is maximal table sharing, and the most addresses fit into the smallest space. If addresses are widely spaced with respect to the 5-bit groups, there is minimal table sharing, because each 32-entry table has only one entry at each level. This requires the most SRAM per address.

PRINCIPLES OF OPERATION

register spaces/external devices accessible through the TNETX15VE

Although the TNETX15VE is designed to work in a CPU-less environment, access to the internal registers is useful for the following reasons:

- Setting/changing port VLAN assignments
- Setting spanning-tree options
- Setting operational modes (startup options)
- Resetting the device (hardware and software resets)
- Management-based access and control of the lookup table
- Statistic gathering
- Diagnostic operations
- Communicating with attached PHYs through the MII
- Reading/writing to an external EEPROM

Figure 12 shows the various register spaces provided by and accessed through the TNETX15VE.

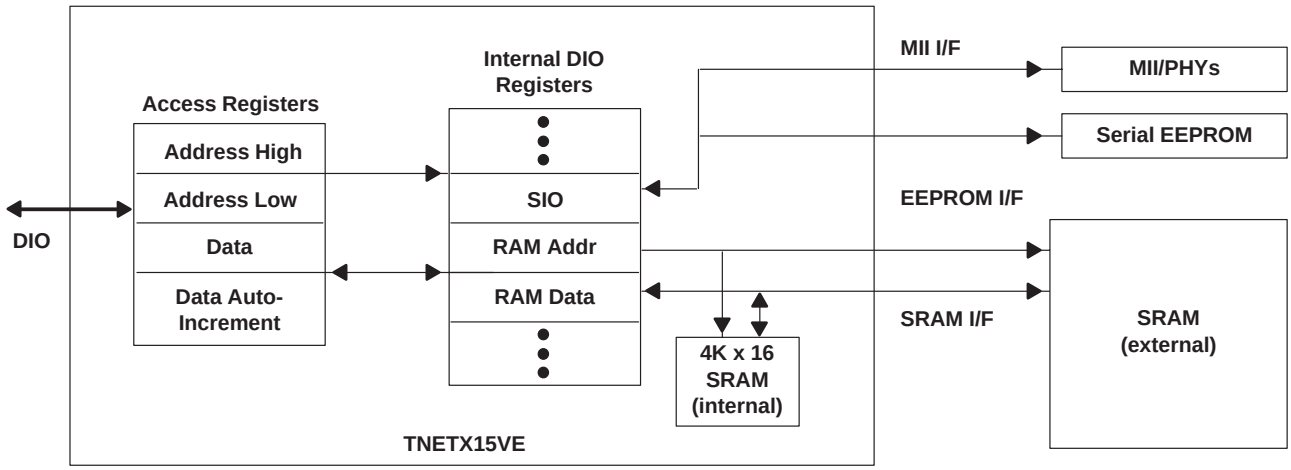


Figure 12. Register Spaces Provided and Accessed Through the TNETX15VE

PRINCIPLES OF OPERATION

DIO interface

The DIO interface is asynchronous to allow easy adaptation to a range of microprocessor devices and computer system interfaces. The DIO interface is designed to operate from the same bus as the TNETX3150/TNETX3150A DIO interface. Thus, both devices are accessed using the same DIO read and write routines. Each device is selected for DIO reads and writes through independent chip-select signals. Chip select for the TNETX3150/TNETX3150A is named $\overline{\text{SCS}}$, while chip select for the TNETX15VE is named $\overline{\text{ESCS}}$. Figure 13 shows how the TNETX15VE and the TNETX3150/TNETX3150A share the DIO interface.

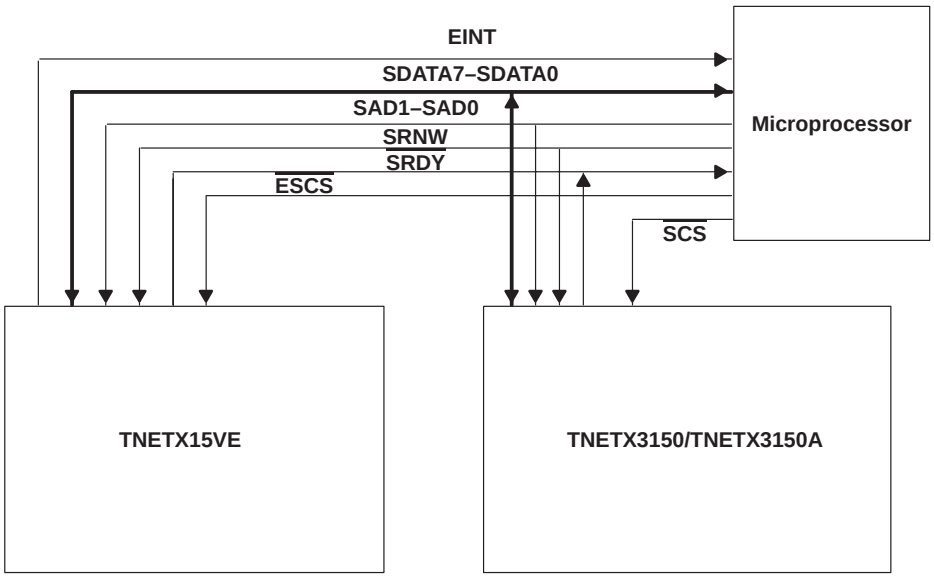


Figure 13. DIO Interface Bus

PRINCIPLES OF OPERATION

DIO write cycle

The write cycle waveforms are shown in Figure 14 and described in the following:

- The TNETX15VE host register address (SAD1–SAD0 and SDATA7–SDATA0) is asserted and SRNW goes low.
- After a setup time, $\overline{\text{ESCS}}$ goes low, initiating a write cycle.
- The TNETX15VE pulls $\overline{\text{SRDY}}$ low as the data is accepted.
- SDATA7–SDATA0, SAD1–SAD0, and SRNW signals are deasserted after the hold time has expired.
- $\overline{\text{ESCS}}$ goes high (by the host) to complete the cycle, causing $\overline{\text{SRDY}}$ to deassert and to go high for one cycle before going into the high-impedance (Z) state.

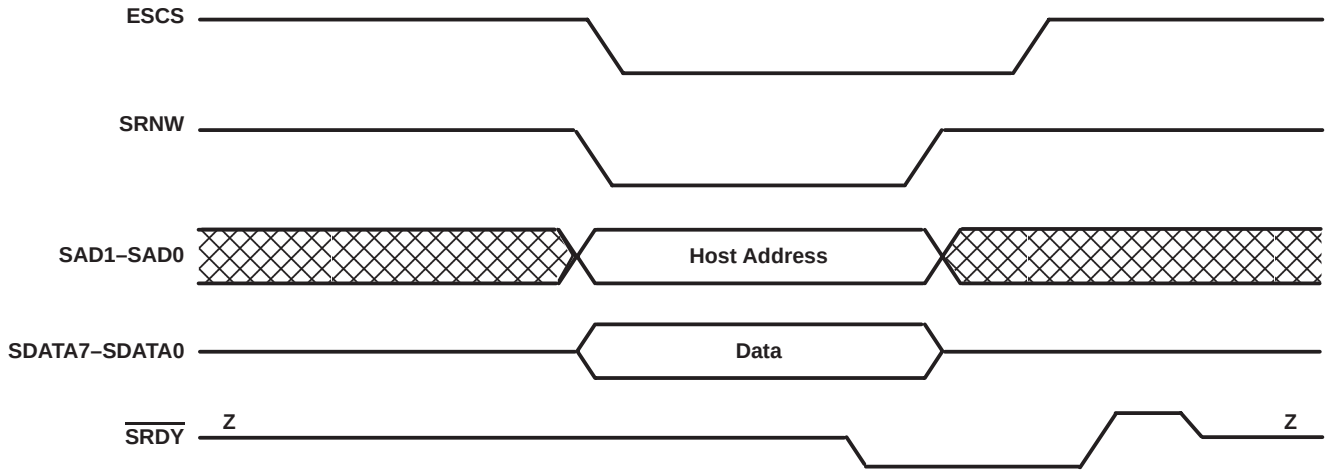


Figure 14. DIO Write Cycle Waveforms

PRINCIPLES OF OPERATION

DIO read cycle

The read cycle waveforms are shown in Figure 15 and described in the following:

- The TNETX15VE host register address SAD1–SAD0 is asserted while SRNW is held high.
- After a setup time, $\overline{\text{ESCS}}$ goes low, initiating the read cycle.
- After a delay time, with $\overline{\text{ESCS}}$ low, SDATA7–SDATA0 is released from the Z state. SDATA7–SDATA0 is driven with valid data and $\overline{\text{SRDY}}$ goes low. The host can access the data.
- $\overline{\text{ESCS}}$ goes high (by the host) to signal completion of the cycle, causing $\overline{\text{SRDY}}$ to deassert. $\overline{\text{SRDY}}$ goes high for one clock cycle before going into the Z state. SDATA7–SDATA0 also goes to the Z state.

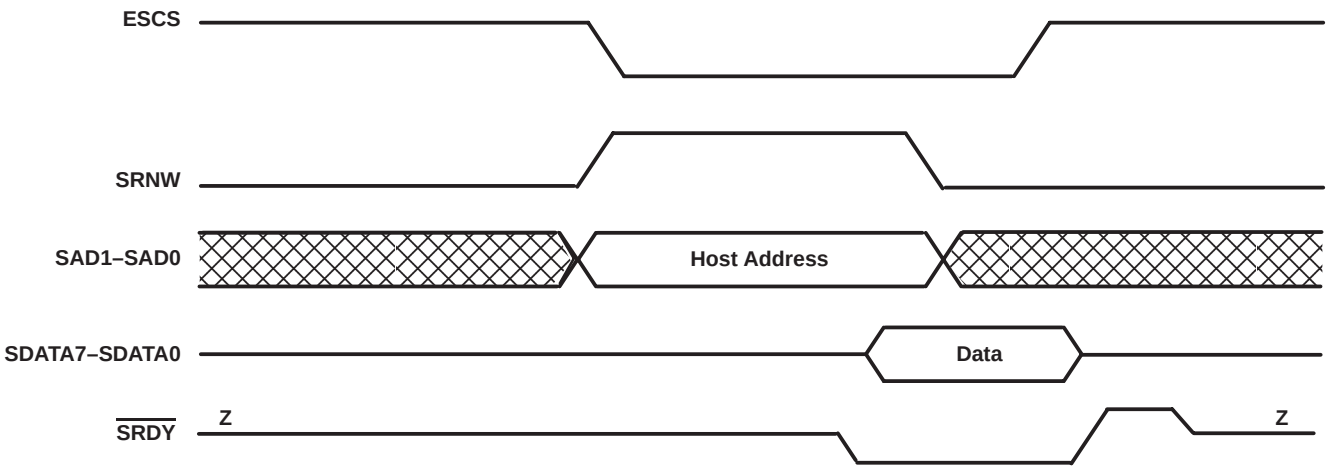


Figure 15. DIO Read Cycle Waveforms

PRINCIPLES OF OPERATION

host (access) register space

The TNETX15VE registers, SRAM (internal and external), and EEPROM are indirectly accessed through the access registers. The access registers are written/read to/through the DIO interface. There are four byte-wide access registers. They are individually selected through the SAD bus, as shown in the following, and the registers are read/written through the SDATA bus.

SAD1	SAD0	ACCESS REGISTERS
0	0	DIO address low
0	1	DIO address high
1	0	DIO data
1	1	DIO data auto-increment

Two bytes, DIO address low and DIO address high, are used to select the address (DIOADR) of the internal register being selected. DIO address high is the most-significant byte of DIOADR and DIO address low is the least-significant byte. The DIO address register is byte writeable. The user does not have to write to both DIO address locations for each access to the internal registers. This saves time in register accesses. Up to 2^{16} possible locations can be accessed through the DIO address register.

A special function (incorporated into this interface) is a hardware reset that is given to the TNETX15VE through the use of the host registers. A hard reset is achieved by writing 0x40 to DIO address high. The value in DIO address low is don't care. Writing this code is equivalent to electrically setting the $\overline{\text{RESET}}$ terminal to 0.

The next two bytes, DIO data and DIO data auto-increment, are used to read and write data to the byte-wide internal register selected in the DIOADR. Both DIO data and DIO data auto-increment can be effectively used to read and write the data, but the DIO data auto-increment register provides additional functionality over DIO data. Access to the DIO data auto-increment register provides a post-increment (by one) to the DIO address register. This is useful for reading/writing to a block of registers.

As an example, to access a single-byte-wide register such as the SIO register (DIO address = 0x0A) the operations needed are:

- Write a 0x00 to DIO address high
- Write a 0x0A to DIO address low to select DIO address 0x0A
- Read the SIO register by reading DIO data, or write to the SIO register by writing to DIO data.

Multiple-byte registers are accessed by reading/writing to their individual bytes. The control register (DIO address 0x08–0x09) is accessed in the following manner:

- Write a 0x00 to DIO address high
- Write a 0x08 to DIO address low to select DIO address 0x08
- Read the least-significant bit of the control register by reading DIO data, or write to the least-significant bit of the control register by writing to DIO data.
- Write a 0x00 to DIO address high
- Write a 0x09 to DIO address low to select DIO address 0x09
- Read the most-significant bit of the control register by reading DIO data, or write to the most-significant bit of the control register by writing to DIO data.

PRINCIPLES OF OPERATION

host (access) register space (continued)

Improvement on the preceding steps is obtained by writing a 0x00 to DIO address high and changing only DIO address low. More steps can be eliminated by using the DIO data auto-increment register to read or write to contiguous register bytes. The following shows how to use the auto-incrementing function to access the control register.

- Write a 0x00 to DIO address high
- Write a 0x08 to DIO address low to select DIO address 0x08
- Read the least-significant bit of the control register by reading DIO data auto-increment or write to the least-significant bit of the control register by writing to DIO data auto-increment. The address in DIO address auto-increments to 0x0009.
- Read the most-significant bit of the control register by reading DIO data auto-increment, or write to the most-significant bit of the control register by writing to DIO data auto-increment.

The auto-incrementing function is useful when reading or writing to a large number of adjacent registers, such as the 48-bit address registers or when reading the statistics block.

internal registers

The internal registers are used to initialize and/or software-reset the TNETX15VE, select the TNETX15VE startup and routing options, checks the number of nodes within the TNETX15VE and statistics, enable management-based operations on the lookup table, and interface with the on-chip or external SRAM, the EEPROM, and any MII-managed devices.

The internal registers are described in the detailed *internal register* section of this document. This section provides additional information on the use of internal registers to access the SRAM, MII devices, and EEPROM.

lookup-table SRAM access

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
		RAMsize		0x00
	RAM addr			0x20
		RAM data		0x24

The TNETX15VE SRAM (internal and external) can be accessed through the internal registers with the RAM addr and RAM data registers. The algorithm for reading and writing to the RAM is similar to that for reading and writing to the internal registers; the address of the location to access is placed in RAM addr and the data is read from or written to RAM data.

To select between internal or external RAM, the RSEL bit in the RAM addr register is used. This interface also has an auto-increment function, which is selected from the INC bit in RAM addr. The auto-incrementer increments by one when the most-significant bit of RAM data is read or written to.

The DIO state machine must request access to SRAM via the SRAM bus scheduler. A write to SRAM is queued when the most-significant byte of RAM data is written. A read of SRAM is queued when either byte of RAM data is read through the DIO port.

PRINCIPLES OF OPERATION

serial interface – MII-managed devices

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
	SIO			0x08

The TNETX15VE gives the programmer an easy way to implement a software-controlled bit-serial interface. This interface is most appropriate in implementing a media-independent interface (MII) management serial interface.

MII devices, which implement the management interface consisting of MDIO and MDCLK, are accessed in this way though the SIO register. In addition (for PHYs that support this) the TNETX15VE implements a third MII-management signal, $\overline{\text{MRESET}}$, to hardware-reset MII PHYs.

MDIO requires an external pullup for operation. The I/O direction is controlled by the MTXEN bit, and the external terminal state is set from or read in MDATA. The complete serial interface (MDIO, MDCLK, $\overline{\text{MRESET}}$) can be placed in a high-Z state through the MDIOEN bit in SIO. High-Z support is needed to prevent two or more devices from driving the MII bus at the same time.

The TNETX15VE does not implement any timing or data structure on its serial interface. Appropriate timing and frame format must be assured by the management software by setting or clearing bits at the right time and in the right order. Refer to the IEEE Std 802.3u specification and the data sheet for the MII-managed device for the nature and timing of the MII waveforms.

x24C02 EEPROM interface

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
	SIO	Control		0x08

The flash EEPROM interface is provided so the system-level manufacturers can provide an optional configured system to their customers. Customers can change or reconfigure the system and retain preferences between system power downs. The flash EEPROM contains configuration and initialization information that is accessed infrequently (typically at power up and reset). The host can write to the EEPROM through this SIO register interface.

The TNETX15VE implements a two-wire serial interface, consisting of EDIO and EDCLK that communicates with the EEPROM. Similar to the MII interface, the TNETX15VE does not implement any timing or data structure on its serial interface. Appropriate timing and frame format must be ensured by the management software by setting or clearing bits at the right times. There are no limits to the size or organization of the EEPROM when the host is driving this interface, but these same terminals are used by an internal state machine to load the contents of the EEPROM into internal registers. This state machine expects a 24C02 (as device 0) on the serial bus. Refer to the manufacturer's data sheet for a description of the EEPROM waveforms.

If an EEPROM is not installed, EDIO should be tied low. For EEPROM operation, EDIO and EDCLK require an external pullup (see EEPROM data sheet). TNETX15VE detects the presence or absence of the EEPROM and indicates this in the NEEPM bit of the control register.

PRINCIPLES OF OPERATION

initialization

The TNETX15VE can be designed for stand-alone use, with no need for a management CPU. It can be reset and initialized with or without a host CPU. This section deals with the steps necessary to bring the TNETX15VE up to an operating level.

TNETX3150/TNETX3150A initialization

If multiple port EAM codings are used (forwarding to multiple ports), then TNETX3150/TNETX3150A IOBMOD bit in the systcl register must be set to a 1. For proper operation, this TNETX3150/TNETX3150A setting must be matched by setting NIOB in the TNETX15VE control register (0xA) to 0. If both bits are reversed (IOBMOD in TNETX3150/TNETX3150A systcl register = 0, NIOB in TNETX15VE control register =1), the ability to send frames to multiple ports, but not all ports, is lost. Multicasts become broadcasts to all ports, regardless of VLAN. Also the broadcast is not in order with other traffic on each transmit queue. If both bits are set to 1 or 0, unpredictable behavior results.

The user must also disable the TNETX3150/TNETX3150A internal address matching when using the TNETX15VE. This is accomplished by writing a 1 to the ADRDIS bit in the port control register of each port. For detailed information on the TNETX3150/TNETX3150A, the reader is referred to the TNETX3150/TNETX3150A data sheet.

resetting the TNETX15VE

hardware reset

The TNETX15VE is hardware reset by asserting $\overline{\text{RESET}}$ low. The TNETX15VE comes out of reset when RESET goes high. The TNETX15VE can also be hardware reset through the DIO interface by writing a 0x40 to the DIO address high access register (DIO address low is a don't care). Both actions are equivalent.

During a hardware reset, no access to the internal registers is allowed. A read or write stalls, that is, $\overline{\text{SRDY}}$ does not go low until the hardware reset stimuli is removed. All access registers and internal registers are initialized to default values.

If an EEPROM is detected (after a hardware reset), the TNETX15VE begins the EEPROM auto-loading process. No DIO operations are allowed during auto-loading.

software reset

The TNETX15VE is software reset by setting the RESET bit to a 1 in the control register (0x08,0x09 DIO). The TNETX15VE remains in the reset state until this bit is cleared. All internal registers are initialized to default values during a software reset, except for the control register, which keeps its current value. Reading the internal registers is allowed during a software reset, but the user is not able to write to any register (except for the control register) ($\overline{\text{SRDY}}$ does not go low).

The EEPROM auto-loading process does not start during a software reset. The user must assert the LOAD bit in the control register for auto-loading to start.

PRINCIPLES OF OPERATION

EEPROM initialization/automatic loading (LOAD)

The TNETX15VE automatically loads selected registers from an attached 24C02 EEPROM after a hardware reset or when the LOAD bit in the control register is set. Up to eight 24C02 EEPROMs can be connected across the same serial interface. They are distinguished by separate addresses that are selectable by pulling address terminals up or down on each EEPROM. The TNETX15VE requires the automatic loading information in device number 0x00, starting at location 0x00.

The TNETX15VE determines if the EEPROM device is present. Several conditions cause the TNETX15VE to determine that a device is not present. If EDIO is pulled down, the EEPROM is assumed to be not present. If the EEPROM fails to acknowledge (Ack) on data writes, the EEPROM is not present. If the cyclic redundancy check (CRC) in the EEPROM does not match the internally calculated CRC, the EEPROM is determined to be not present.

When no EEPROM is detected, The TNETX15VE sets the NEEPM bit in the control register to a 1, and TNETX15VE is placed in a reset state (RESET and NEEPM are set in the control register) while the registers assume their default values.

The organization of the EEPROM data is roughly equivalent to the TNETX15VE registers 0x01–0x05 and 0x50–0xEF. The automatic loader reads the register values from the EEPROM and programs the TNETX15VE accordingly. The last register written is the control register.

The automatic loader can initialize and start up the TNETX15VE if the START bit in the control register is programmed in the EEPROM. This allows for initialization and startup without a host CPU.

During the automatic loading, no DIO operations are permitted. Both reads and writes stall, that is, $\overline{\text{SRDY}}$ does not come true on a pending operation until the EEPROM load cycle is over. The load cycle should take (bits per word) * (words read) * 100 kHz, or 27 * 162 * 100 kHz, or 44 ms. To sense the end, wait for a sample register read to complete, or wait 50 ms. The download bit, LOAD bit, RESET bit, and any other read only or reserved bits cannot be set during automatic loading. The CRC for the EEPROM is calculated using the information written in the EEPROM although information may not be written to the TNETX15VE. For example; a value of 0x8F or 0xFF in the EEPROM for RAMsize are both written as 0x0F in the TNETX15VE since bits 7, 6, 5, and 4 are reserved, but the calculated CRC for the EEPROM for each case is different. As another example, the byte at 0x2E is currently reserved. The value in the EEPROM is not used, but it is included in the CRC calculation.

PRINCIPLES OF OPERATION

EEPROM initialization/automatic loading (LOAD) (continued)

The last four bytes read by the automatic loader correspond to a 32-bit CRC value for the information stored in the EEPROM. The CRC value is calculated by using the following callable C routine, which assumes that the data to be examined is loaded into an array eeprom_data [ ]:

```
void ifexeeprom (char *cs)
   {
   long crc;
   int k;

   crc = 0xffffffffl;
   for (k=0;k<=158;k++)
      {
      crcbyt (eeprom_data[k], &crc);
      }

   crc ^= 0xffffffffl;
   eeprom_data[k++]=(int) ((crc >> 24) & 0x0ffl);
   eeprom_data[k++]=(int) ((crc >> 16) & 0x0ffl);
   eeprom_data[k++]=(int) ((crc >>  8) & 0x0ffl);
   eeprom_data[k++]=(int) ((crc      ) & 0x0ffl);
   }

crcbyt (dat,crc)
int dat;
long *crc;
   {
   int i;

   for (i=0;i<8;i++)
      {
      crcbit(dat>>7,crc);
      dat = dat <<1;
      }
   }

crcbit (dat,crc)
int dat;
long *crc;
   {

   if ( (((*crc>>31) & 11)^((long)dat & 11)) ==1)
      {
      *crc ^= 0x02608edbl;
      *crc = *crc <<1;
      *crc |= 0x00000001l;
      }
   else
      {
      *crc = *crc <<1;
      *crc &= 0xfffffffel;
      }
}
```

In this example, the values for which the CRC is calculated are placed in the `eeprom_data[]` array from offset 0 to 158 (0x9E). The routine `crcbyt` is called for each byte. The C routine then places the calculated CRC values in the `eeprom_data[]` array at locations 159 to 162 (0x9F to 0xA2).

PRINCIPLES OF OPERATION

TNETX15VE operational modes

The TNETX15VE operational modes are selected through the control register. They are used to control decision points in the logic operations. The modes available through the control register are NAUTO, NCRC, and MIRR. Other operating parameters are selectable through the agingtimer, NLRNports, TXblock, and RXblock registers.

NAUTO mode

The not automatically add-address (NAUTO) mode is implemented to give the management CPU complete control of the lookup table. It does so by disabling the two automatic processes that can affect the lookup-table, wire additions, and aging.

NAUTO mode disables adding addresses to the TNETX15VE tables learned from packets passing through the TNETX3150/TNETX3150A. All addresses in the TNETXVE must be added through the DIO interface (see ADD in internal register section). No interrupt is generated on addresses added through the DIO interface, even if the host sets the NEW bit in the addport register so that the address can be found with a FINDNEW command.

NEW interrupts in this mode signal that the address checking state machine has seen an address on the wire that is not in the table.

NAUTO also affects the AGE logic operation by disabling it. AGE does not delete nodes from the table even if the table is full. It is the management's responsibility in this mode to maintain the addresses in the lookup table. Table-full conditions are determined through a FULL interrupt.

NCRC mode

The no-CRC (NCRC) check mode enables the TNETX15VE to add addresses learned from packets before the CRC is checked. It is a performance-boosting feature for the ADD state machine and it allows it to perform more ADD logic operations in the same amount of time. This allows the TNETX15VE to add more efficiently and keep the aging time stamp current on nodes that do not talk frequently on the network, avoiding unnecessary aging.

The tradeoff in this mode is the possibility that corrupt addresses could be added into the lookup table. This condition should not become critical unless aging is turned off.

MIRR mode

The mirrors activity (MIRR) mode directs all of a port's traffic to a user-specified port. This, in effect, mirrors the activity to the selected port. Three types of traffic are copied: traffic that is directed to the port, traffic that is coming from the port, and traffic that stays within the port's segment and would ordinarily be discarded. MIRR mode may decrease performance in the switch since it generates additional traffic within the TNETX3150/TNETX3150A.

aging modes (agingtimer register)

The TNETX15VE implements an autonomous aging logic operation. There are two modes of aging and these modes are controlled through the agingtimer register. The two modes are time-threshold aging and table-full aging.

Time-threshold aging is set by writing a value between 0x0001 and 0xFFFE to this register. In this mode, the register value gives the time in seconds that is used by the AGE logic operation to delete addresses. This provides an aging range of 1 second to 18 hours. Addresses older than this are deleted.

Table-full aging is set by writing 0x0000 or 0xFFFF to the register. In this mode, the AGE logic operation ages out addresses only when the table is full and the ADD logic operation needs to add an address. AGE deletes the oldest address in the table.

PRINCIPLES OF OPERATION

NLRN mode

The no-learn (NLRN) mode disables the automatic wire learning of node addresses that are assigned to a port on a port-by-port basis. While learning is turned off, no inbound frames can affect time stamps for addresses already received. If the address was present in the table and aging is running, the address can age out even if frames are being received from that source address. NLRN mode does not disable a port and prevent it from receiving or transmitting frames.

TX block mode

The transmit-blocking (TX block) mode disables the transmission of frames by any port in this mode. Frames are still received and forwarded to other ports from this port with this mode set.

RX block mode

The receive-blocking (RX block) mode blocks all frames from being received by any port in this mode. Address learning from this port is not disabled in this mode.

TX block, RX block and NLRN can be used in combinations.

lookup table SRAM initialization (START)

The lookup table is automatically initialized by the TNETX15VE with no need for an external processor. The steps for initializing are:

- Write to RAMsize the size of the external SRAM. Writing to RAMsize can be performed by a CPU or the EEPROM auto-load operation.
- Assert the START bit in the control register. This is accomplished either by the CPU or EEPROM.
- The TNETX15VE indicates the completion of the lookup-table initialization by setting the INITD bit in the control register to 1.

TNETX15VE clears the lookup table by writing 0x0000 to all available locations and then initializes the data structures (FIFO, lookup tables). After these operations are done, The TNETX15VE performs lookups, adding, and aging operations as directed by mode bits.

frame forwarding-LKUP logic operation

The lookup (LKUP) logic operation is designed for two important tasks: 1) perform time-critical lookups of the wire within the TNETX3150/TNETX3150A's allotted time, and 2) forward the frame to the right ports. The LKUP logic operation works independently from all other logic operations and from the management CPU. To meet the timing requirements, this logic operation occupies the highest priority on the SRAM bus.

LKUP logic operation performs a lookup on the destination address of the frame, plus a 2-bit VLAN, and routes traffic accordingly. The destination address is found from the DRAM interface. The VLANID used is determined from the tag field and the source port, as described previously.

The lookup logic operation routes the frame differently, depending on whether the frame is found in the lookup table. Other registers and conditions also affect these routing options. The registers that affect routing options are VLANID, VLANmask, tagVLAN, control, TXblock, RXblock, mirrorport, and uplinkport. The LOCKED and CUPLNK bits contained in the lookup table also affect the routing options for unicasts. The NBLCK flag affects the routing options for unicasts and multicasts.

Nominally, if an address is found, the TNETX15VE uses the information contained in the lookup table for routing. If the address is not found, the TNETX15VE broadcasts the frame to the ports indicated in the VLANmask register for the VLAN associated with the source port. The other registers listed above block forwarding or send the packet to additional ports.

PRINCIPLES OF OPERATION

guidelines for calculating the value of VLANmask from the VLAN assignments

Addresses that are unknown are broadcast using the VLANmask register for the source-port VLAN. A simple way of determining the value of these registers is discussed. The VLANmask is calculated from the VLANID and tagVLAN register array.

If the TNETX3150/TNETX3150A/TNETX3100 and TNETX15VE chipset is stand alone, the VLANmask registers have a bit set for each port that belongs to that VLAN. All the entries in the tagVLAN array must agree, to indicate what VLAN port 00 belongs to.

If the chipset is cascaded/expanded (where port 00 is relative to other intranetworking equipment rather than end users/segments), the VLANmask register has a bit set for each port that belongs to that VLAN plus a 1 in port 00s position if stations in the VLAN are on the other side of port 00. The entries in the tagVLAN array correlate the inbound pretag with the appropriate VLAN.

The following example is for the reference side of a cascaded TNETX3150/TNETX3150A and TNETX15VE:

(Locally VLANs 0 and 2 are used)	(Remotely VLANs 0, 1, and 3 are used)
VLANID1 = 0x2	TagVLAN0 = 0x0
VLANID2 = 0x2	TagVLAN1 = 0x0
VLANID3 = 0x0	TagVLAN2 = 0x0
VLANID4 = 0x2	TagVLAN3 = 0x1
VLANID5 = 0x2	TagVLAN4 = 0x0
VLANID6 = 0x2	TagVLAN5 = 0x3
VLANID7 = 0x0	TagVLAN6 = 0x3
VLANID8 = 0x0	TagVLAN7 = 0x0
VLANID9 = 0x0	TagVLAN8 = 0x0
VLANID10 = 0x2	TagVLAN9 = 0x3
VLANID11 = 0x0	TagVLAN10 = 0x0
VLANID12 = 0x0	TagVLAN11 = 0x3
VLANID13 = 0x2	TagVLAN12 = 0x0
VLANID14 = 0x0	TagVLAN13 = 0x0
	TagVLAN14 = 0x0
	TagVLAN15 = 0x0

Then, the VLANmask registers should be as follows: VLANmask0 = 0x5B89, VLANMASK1 = 0x0001, VLANmask2 = 0x2476, AND VLANmask3 = 0x0001.

VLAN	PORT															
	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	1	0	1	1	0	1	1	1	0	0	0	1	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	1	0	0	0	1	1	1	0	1	1	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

PRINCIPLES OF OPERATION

management-based lookups (FIND)

The FIND logic operation is designed to give the programmer a simple way to find addresses within the lookup table. The FIND logic operation is controlled from the following internal registers:

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
FindVLANID				0x08
Findnode23–Findnode16	Findnode31–Findnode24	Findnode39–Findnode32	Findnode47–Findnode40	0x0C
Findport		Findnode7–Findnode0	Findnode15–Findnode8	0x10
	Findcontrol	Findnodeage		0x14

The interface provides: 48-bit read or writeable register findnode in which the address is placed; a VLANID register findVLANID in which the node's VLAN is shown; a 16-bit register findport in which routing information is placed; and a 16-bit register findnodeage, which contains the age time stamp of the node being looked up.

FIND, FINDFIRST, FINDNEXT, and FINDNEW commands

Four FIND commands can be modified by PORT or VLAN bits: FINDFIRST, FINDNEXT, FIND(LKUP), and FINDNEW. These are used to interrogate the table that the TNETX15VE built as a result of all ADDS. Without the modifiers, FIND works on all addresses; each modifier applied reduces the pool. Applying VLAN and PORT results in an address that matches that VLAN and that PORT.

Also classed as a FIND is FINDNBKR (the interrogation of the FIFO storing and address/port information for packets that overrode the receive block barrier). Showing the value on the top of the FIFO pops it off.

FINDNEW looks for addresses that still have their NEW bit set. Finding an address with the NEW bit set results in the NEW bit being cleared. Every FINDNEW operation starts at the beginning of the table. It does not affect the operation of the FINDNEW to load or leave an address in the findnode registers (0xC–0x11).

The logic performs the command given to it, and indicates that it has stopped looking for the address by asserting the FNDCPLT bit in the int register. If the FNDCPLT bit in the intmask register is set to a 1, this gives an interrupt to the host. Another way FIND indicates completion is by clearing its command bits. The findcontrol register keeps the code that the user has written until the process is complete. For example, if the user writes a 0x01 to findcontrol (LKUP) then this register reads 0x01 until the FIND logic operation is finished operating.

If an address is found as a result of the command just completed, it is indicated by asserting the FOUND bit in findcontrol. The FOUND bit indicates that the information in the findnode, findVLANID, findport, and findnodeage registers is valid.

The FIND command finds a specific user-defined address in the lookup table. The procedure for the FIND command is as follows:

- Write the 48-bit address query in the findnode register and the node's 2-bit VLANID in findVLANID.
- Set the LKUP bit in the findcontrol register. The TNETX15VE scans the lookup table for that particular address.
- Poll the findcontrol register until the LKUP bit is cleared. Since a LKUP is very quick, only a few polling cycles are required. No interrupt is given for LKUP-operations completing.
- Read the findcontrol register. If FOUND is set, the address was found and the node's information is placed in the registers. If FOUND is not set, the address was not found within the lookup table.

PRINCIPLES OF OPERATION

FIND, FINDFIRST, FINDNEXT, and FINDNEW commands (continued)

The FINDFIRST command finds the first address contained in the lookup table. The procedure for the FIND command is as follows:

- Set the FIRST bit in the findcontrol register. No write to findnode or findVLANID is required. The TNETX15VE scans the lookup table for the first address. The search is confined to a VLAN and/or PORT by setting those modifier bits in the findcontrol register.
- Wait until the FNDCPLT interrupt is asserted [FNDCPLT (bit 6) in interrupt register] or poll the findcontrol register until the FIRST bit is cleared.
- Read the findcontrol register. If FOUND is set, an address was found and the node's address and information is placed in the registers. If FOUND is not set, an address was not found and the rest of the lookup table is empty.

The FINDNEXT command finds the next address from that contained in the findnode register or the same address on a separate VLAN. The user can either write a value in findnode and findVLANID and find the next address or keep the current value and continue finding next addresses. The procedure for the FIND command is as follows:

- Write the starting address in findnode and findVLANID (if desired) or keep the currently held address and VLAN.
- Set the NEXT bit in findcontrol. The TNETX15VE scans the lookup table for the next address after the one contained in findnode or for the same address in a separate VLAN. The search is confined to a VLAN and/or PORT by setting those bits in the findcontrol register.
- Wait until the FNDCPLT interrupt is asserted. [FNDCPLT (bit 6) in interrupt register] or poll the findcontrol register until the NEXT bit is cleared.
- Read the findcontrol register. If FOUND is set, and the address was found, the node's address and information is placed in the registers. If FOUND is not set, and an address was not found, the rest of the lookup table is empty.

The commands can be combined to quickly dump the address table. All that is required is a FINDFIRST followed by FINDNEXT commands, until no more addresses are found.

Due to the TNETX15VE efficiency in adding addresses and changing ports, and since these functions occur without user intervention, a method is necessary to keep the host informed of all addresses that have changed.

The interrupt registers fulfill this function, but during the time that one interrupt is processed, many more can go by unnoticed. The management is then reduced to scanning in the table using the FIND logic operation to determine which addresses have changed. This is impractical in that a local copy of the table must be kept and then compared with the TNETX15VE table. To alleviate this problem, the NEW flag is used.

The NEW flag in the SRAM is asserted every time the TNETX15VE adds an address or changes an address port assignment. The FIND logic operation, tuned to look for this flag in the SRAM, filters out addresses that have not changed.

PRINCIPLES OF OPERATION

FIND, FINDFIRST, FINDNEXT, and FINDNEW commands (continued)

The NEW bit is asserted by the ADD logic operation. The FIND logic operation clears this bit after giving the address to management. The steps for scanning the whole table for all NEW addresses is as follows:

- Perform a FINDNEW command by:
  - Setting the NEW bit in the findcontrol register. No write to findnode is required. The TNETX15VE scans the lookup table for the first NEW address from the beginning of the table each time NEW is set.
  - Waiting until the FNDCPLT interrupt is asserted [FNDCPLT (bit 6) in interrupt register] or poll findcontrol until the register bits are cleared
  - Reading the findcontrol register. If FOUND is set, and an address was found, the node's address and information are placed in the registers. The NEW flag in the address is cleared. If FOUND is not set, an address was not found and the lookup table contains no NEW addresses.
- Since the FIND command clears the NEW bit, looking for the next NEW address means repeating the FINDNEW command until no more addresses are found. The NEW bit is cleared on all FIND commands. Always check the NEW bit (7) in the findport register (0x12–0x13 DIO) on non-NEW finds.
- NEW is automatically set with a wire add; the host can set the state of the NEW bit with an add/edit command.

VLAN and PORT find modifier bits

The VLAN and PORT bits in the findcontrol register are used with the NEW, FIRST, NEXT, and LKUP bits to modify the FIND operations.

The PORT bit limits the FINDFIRST, FINDNEXT, and FIND commands to look for only the addresses whose port assignment matches the port contained in the portcode field of findport. This command is useful when management wants to know all addresses that are associated with a port number. Since only unicast addresses can be matched to a port (multicasts use a port bit mask), using the PORT modifier limits returns to unicast addresses. To dump all unicast addresses associated with port 01, the user must:

- Write a 0x01 to the most-significant bit of findport (portcode = 0x1)
- Perform a FINDFIRST port command by:
  - Setting the PORT and FIRST bits in the findcontrol register. No write to findnode and findVLANID is required. The TNETX15VE scans the lookup table for the first address that is associated with port 01.
  - Waiting until the FNDCPLT interrupt is asserted, or poll findcontrol until the FIRST bit is cleared. The PORT bit stays set.
  - Reading the findcontrol register. If FOUND is set, and an address was found, the node's address and information are placed in the registers. If FOUND is not set, and an address was not found, there are no more addresses associated with this port.

PRINCIPLES OF OPERATION

VLAN and PORT find modifier bits (continued)

- If an address was found, perform a FINDNEXT port by:
  - Setting PORT and NEXT bits in the findcontrol register. The TNETX15VE scans the lookup table for the next address after the one contained in findnode. Since VLANs are port based, the same MAC address cannot appear more than once on the same port.
  - Waiting until the FNDCPLT interrupt is asserted, or poll findcontrol until the NEXT bit is cleared. The PORT bit stays set.
  - Reading the findcontrol register. If FOUND is set, the next address was found and the node's address and information are placed in the registers. If FOUND is not set, there are no more addresses associated with this port.
- Repeat the FINDNEXT port command until no more nodes are found.

The VLAN bit is similar to the PORT bit in that it limits the FINDFIRST, FINDNEXT, and FIND commands to look for only the addresses whose VLANID matches the VLAN contained in findVLANID. This command is useful when management wants to know all addresses that are associated with a VLAN. This function returns both unicasts and multicast addresses assigned to the VLAN. To dump addresses associated with VLAN 0x3, the user must:

- Write a 0x03 to findVLANID.
- Perform a FINDFIRST VLAN command by:
  - Setting the VLAN and FIRST bits in the findcontrol register. No write to findnode is required. The TNETX15VE scans the lookup table for the first address that is associated with VLAN 0x3.
  - Waiting until the FNDCPLT interrupt is asserted (or poll findcontrol) until the FIRST bit is cleared. The VLAN bit stays set.
  - Reading the findcontrol register. If FOUND is set, and an address was found, the node's address and information are placed in the registers. If FOUND is not set, there are no more addresses associated with this VLAN.
- If an address is found, perform a FINDNEXT VLAN by:
  - Setting the VLAN and NEXT bits in the findcontrol register. The TNETX15VE scans the lookup table for the next address associated with the VLAN, starting with the address left in the previous step.
  - Waiting until the FNDCPLT interrupt is asserted (or poll findcontrol) until the NEXT bit is cleared. The VLAN bit stays set.
  - Reading the findcontrol register. If FOUND is set, and the next address was found, the node's address and information are placed in the registers. If FOUND is not set, there are no more addresses associated with this VLAN.
- Repeat the FINDNEXT port command until no more nodes are found.

PRINCIPLES OF OPERATION

FIFO FINDS (NBFR–no broadcast FIFO reads)

In certain applications, such as spanning tree, it is important to know the source address of frames sent to a common-destination address. The TNETX15VE gives the programmer the capability of doing this, but limits this to only those frames whose destination address is marked with the NBLCK bit.

The TNETX15VE performs a destination address search on all frames going by on the DRAM bus. If the destination address matches one on the lookup table, and if the matching address on the lookup table has the NBLCK bit set, the TNETX15VE writes the source address of this frame into the NBLCK RXFIFO, regardless of whether or not the receiving port is being RX blocked. The TNETX15VE then informs the host that the FIFO contains data by asserting an RXFIFO interrupt (RXFIFO bit in the interrupt and intmask registers). The NBLCK FIFO entries are made as a result of the LKUP operation. An entry is made even if the packet has a bad CRC.

The TNETX15VE FIFO reads the NBLCK RXFIFO using the FIND state machine. The procedure for performing FIFO reads is as follows.

- Perform a FIFO read:
  1. Set the NBFR bit in the findcontrol register. No write to any other register is required. The TNETX15VE scans the FIFO for the first address.
  2. Poll the findcontrol register until the NBFR bit clears. No FNDCMPLT interrupt is given due to the speed of the FIFO read. At most, one poll is required.
  3. Read the findcontrol register. If the FOUND bit is set, then an address was found and the node's address is given in the findnode register. The frame's tag and the port the frame originated from is placed in the findport register. This address is then cleared from the FIFO. If the FOUND bit is not set, then an address was not found and the FIFO is empty.
  4. Repeat the NBFR operation until no more addresses are found.

summary of FIND operations supported

Table 5 summarizes the supported FIND operations, the bits used to select the operation, and the code written to the findcontrol register.

Table 5. FIND Operations

OPERATION	BITS USED	FINDCONTROL
FIND (LKUP)	LKUP	0x01
FINDNEXT	NEXT	0x02
FINDFIRST	FIRST	0x04
FINDNEXTVLAN	VLAN, NEXT	0x0A
FINDFIRSTVLAN	VLAN, FIRST	0x0C
FINDNEXTPORT	PORT, NEXT	0x12
FINDFIRSTPORT	PORT, FIRST	0x14
FINDNEXTPORTVLAN	PORT, VLAN, NEXT	0x1A
FINDFIRSTPORTVLAN	PORT, VLAN, FIRST	0x1C
NBLCK FIFO READ	NBFR	0x20
FINDNEW	NEW	0x40
FINDNEWVLAN	NEW, VLAN	0x48
FINDNEWPORT	NEW, PORT	0xA0
FINDNEWPORTVLAN	NEW, PORT, VLAN	0xA8

PRINCIPLES OF OPERATION

adding an address

The ADD logic operation is responsible for new address additions to the lookup table, address port changes, marking the address as NEW, modifying the information stored in the lookup table, and keeping the address time stamp current. The TNETX15VE implements a single ADD logic operation and shares it between automatic adds from the wire and host port additions. The TNETX15VE prioritizes wire adds over management adds. It completes an add request before starting another add request.

The ADD algorithm is summarized as follows:

- ADD performs a lookup to determine if the source address exists in the table.
- If the address exists, ADD verifies that the port assignment has not changed. If the port assignment changes and the address is not secured, a CHNG interrupt is issued; if the address is secured, a SECVIO interrupt is issued. If NAUTO = 0 and the address is unsecured, ADD updates the port and sets the NEW bit. In all cases, ADD updates the AGE time stamp.
- If the address does not exist, a NEW interrupt is issued. If NAUTO = 0, ADD adds the address to the table with the current time stamp and the NEW bit is set.

Adding an address may require allocating additional lookup tables. It is possible that during the adding process no more lookup tables are available for address additions. A FULL interrupt is issued to the host, and the ADD operation stalls, pending the resolution of the no-space-available situation. In this situation, the following occurs:

- If the NAUTO bit is set to 1 (host-managing table), the host deletes addresses in the table until enough are deleted to make space for the new address. Depending on the shared tables used by the addresses being deleted, and the tables that are shared by the inbound address, it is possible that many addresses may have to be deleted for the new address to fit. When enough space is available, the ADD operation completes, as denoted by the ADD bit in the adddelcontrol register, 0x2C, going to 0.
- If the NAUTO bit is set to 0 (TNETX15VE managing table), the TNETX15VE deletes as many addresses as necessary to fit the new address into the table by deleting the oldest addresses, using the AGE function.

The following modes indicate control and other options that affect the ADD logic operation.

NAUTO mode

NAUTO mode is selected by setting the NAUTO bit in the control register to 1. In NAUTO mode, the ADD logic operation does not add addresses off the wire. The only way addresses are added, moved, or edited is through the register interface. The wire ADD state machine does continue checking addresses going by on the wire, but the host processor must act on an indication of a new address to have that address placed in the table.

PRINCIPLES OF OPERATION

NLRN mode

The ADD logic operation cannot generate any indication of a new address from ports whose learning has been disabled by having its bit in the NLRNport register set. The bus watcher does not extract these addresses from the DRAM bus so no table search is possible. In this mode, the management CPU can add an address on the learning disabled ports. Since the bus watcher does not provide addresses from these ports to ADD in this mode, ADD does not perform any age touches to any addresses in the lookup table assigned to ports in this mode.

NCRC mode

The NCRC bit controls whether or not the bus watcher waits for a complete valid (CRC checked) frame before giving it to ADD. The TNETX15VE performs additions faster in NCRC mode since it does not have to wait for the good CRC indication on the bus. There is a possibility that addresses from bad (CRC checked) frames are added, but the aging process deletes them eventually.

management address adding

The ADD logic operation also can add addresses through the DIO interface management add/edit address interface registers as shown in the following:

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
		AddVLANID	Adddelcontrol	0x2C
Addnode23–Addnode16	Addnode31–Addnode24	Addnode39–Addnode32	Addnode47–Addnode40	0x38
Addport		Addnode7–Addnode0	Addnode15–Addnode8	0x3C

Management ADDs are used to perform the following functions:

- The address' flags NEW, NBLCK, SECURE, LOCKED, and the copy uplink flag, CUPLINK, can be set or cleared through management adds.
- DIO ADDs are used to change the address port assignment.
- DIO ADDs are also the only way multicast and broadcast addresses are added to the lookup table.
- DIO ADDs also write the current time to the AGE time stamp for the node.

Management add commands are given through the ADD bit in the adddelcontrol register. The steps for adding an address are as follows:

- Write the node's address in the addnode registers and its VLAN in addVLANID.
- If it is a unicast address, write the node's flag information and port assignment in addport. If it is a multicast address, write the forwarding mask.
- Set the ADD bit in the adddelcontrol register to 1.

The ADD logic operation locks the addnode, addVLANID, and addport registers to ensure that they do not change during the address ADD. Reads to these registers are still possible. The ADD bit in the adddelcontrol register remains a 1 until the ADD is complete.

Having a sticky bit (remains a 1) for ADD gives the programmer the opportunity to setup or perform other register operations without having to wait for the ADD completion. A polling method is used to find out if the ADD is finished. This involves reading the adddelcontrol register to determine if the ADD bit is low. The command bit stays set to 1 until command execution is completed.

PRINCIPLES OF OPERATION

adding unicasts and multicast addresses

There is no significant change in procedure between adding unicast and multicast addresses. There is one difference that the programmer must observe. The TNETX15VE stores different information for multicast addresses than for unicast addresses. Unicast addresses use a 4-bit code for the port number and three flag bits. Multicast addresses store a 15-bit port vector code.

Both data formats are added through the addport register. The format for this register changes, depending on bit 40 in the addnode register. If set to 1, the addport data is interpreted as multicast (if not, it is interpreted as unicast).

deleting an address

The TNETX15VE has two ways to delete addresses from the lookup table — the internal aging algorithm and a host request through the DIO interface. The DEL state machine is responsible for deleting addresses from the lookup table. DEL takes its information from the DIO registers for DIO deletes and from the AGE logic operation for aging deletes.

The TNETX15VE implements a 16-bit timer, incrementing every second for the aging process. This timer is used to write the time stamp during ADDs and for comparing ages.

aging (AGE logic operation)

The AGE logic operation is responsible for automatic address deletes. The TNETX15VE implements two styles of aging: time-threshold aging and table-full aging. The aging style is selected through the agingtimer register. A value of 0x0000 or 0xFFFF in the agingtimer register selects table-full aging. Any other value selects time-threshold aging. The AGE logic operation is disabled when the TNETX15VE is placed in NAUTO mode. The aging algorithm works as shown in Figure 16 and described in the following:

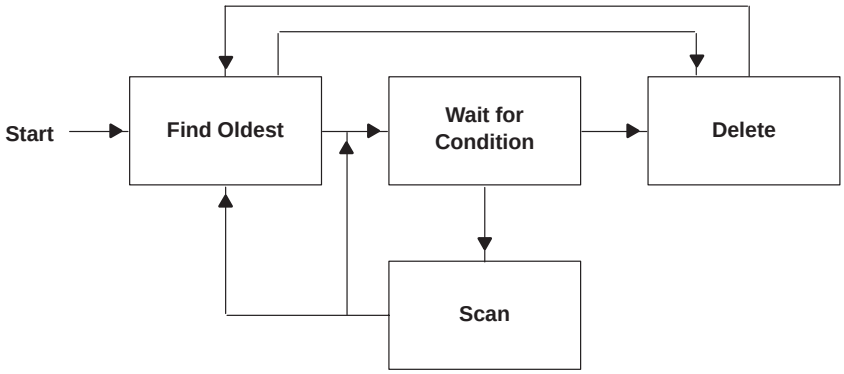


Figure 16. Aging Algorithm Block Diagram

- The operation AGE scans the table for the oldest address (state = find oldest state). AGE determines the oldest address by finding the address in the lookup table with the lowest time stamp. If more than one address has the same oldest time stamp, AGE picks the first address. The AGE scanning process omits all multicast addresses. This process also omits unicast addresses that have been secured by setting the SECURE flag. These addresses can be deleted only by a DIO-delete command.

PRINCIPLES OF OPERATION

aging (AGE logic operation) (continued)

- Once the oldest address is found, AGE keeps this address, enters a waiting state (state = wait for condition), until one of two conditions occurs:

  1. If the address table is changed by either the ADD logic operation performing an address addition/time-stamp update or by deleting (DEL) an address, AGE checks the address that ADD or DEL is working on. If ADD has changed the time stamp for the current oldest address, it must scan the table for a new oldest address (state = find oldest). If DEL has deleted the current oldest address, it again must rescan the table for a new oldest address (state = find oldest). If neither ADD nor DEL touched the current oldest address, it still remains the oldest address and AGE returns to the wait state (state = wait for condition).
  2. The aging condition is met. In this case, AGE calls on the DEL logic operation to delete the node from the table. After a successful deletion, AGE rescans the table for the next node to age (state = find oldest) and then gives an interrupt to the host to indicate the previous oldest address was deleted from the table.

- During the find-oldest process and if AGE is in a time-threshold age, AGE deletes all addresses that are over the threshold value.

The aging condition is different for time-threshold aging and table-full aging, and both are described in the following:

time-threshold aging

In time-threshold aging, the aging condition occurs when the address age is larger than the time threshold entered in the agingtimer (0x2) registers. The address age is not the time stamp written in the SRAM but the difference between current time and each time stamp. When this value becomes greater than agingtimer value, the address is deleted.

Example:

Time threshold value	= 192d seconds (0x00C0)
Current timer value	= 256d seconds (0x0100)
Address time-stamp value	= 80d seconds (0x0050)
Address aging time	= 256d – 80d = 176d seconds (0x00B0)

Conclusion: The address is not aged yet since the address aging time (176d seconds) is less than the time threshold value (192d seconds) by 16d seconds. It takes an additional 16d seconds (0x0010) for the address aging time to equal the time threshold (192d seconds) (0x00C0) and age (delete) the address.

table-full aging

Table-full aging is implemented for applications that do not want to use aging based on time, but still require aging. As its name implies, aging in this mode only happens when the lookup table is full and needs additional room to add a new address. The ADD logic operation initiates an aging request when it determines that it does not have enough tables to add the address it is working on. The time behaves differently in this mode. In table-full aging, the age timer does not increment every second, but rather when a new address is added. There must be a time difference between addresses to decide which is oldest, but a single count is adequate. Since ADD time stamps every time it sees a node come through the bus, nodes that are actively transmitting between adds quickly move up to the same (new) age level. Those nodes that do not transmit remain at the lower age stamps. It is these nodes that are deleted in table-full aging when the table is full.

PRINCIPLES OF OPERATION

DEL logic operation (management address deletions)

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
			Adddelcontrol	0x2C
Delnode23–Delnode16	Delnode31–Delnode24	Delnode39–Delnode32	Delnode47–Delnode40	0x48
Delport	DelVLANID	Delnode7–Delnode0	Delnode15–Delnode8	0x4C

The DEL logic operation is controlled through the delnode, delVLANID, and delport registers and the adddelcontrol register. Management delete commands are given through the DEL, DELP, and DELV bits in the adddelcontrol register.

The steps for deleting any specific address are as follows (multicast and secured unicast can only be deleted this way):

- Writing the node address in the delnode register and the node's VLAN in delVLANID
- Asserting the DEL bit in adddelcontrol

The TNETX15VE also can delete all addresses on a port. The steps for deleting all addresses on a port are:

- Writing the port in delport. Delnode and delVLANID are don't cares and are ignored.
- Asserting the DELP bit in adddelcontrol

As in FIND, only unicast addresses are associated with a port; only unicast unsecured addresses are deleted with a DELP command. Another multiple-delete command is used when a deletion by VLAN is desired. The TNETX15VE has the capability to delete all unsecured, unicast addresses on a particular VLAN. The steps for this command are:

- Writing the VLAN to be deleted in delVLANID. Delnode and delport are don't cares and are ignored.
- Asserting the DELV bit in adddelcontrol

The DEL logic operation locks the delnode and delVLANID registers on all delete commands to ensure that they do not change during the deletions. Reads to these registers are still possible. The DEL, DELP, AND DELV bits in adddelcontrol remain in the 1 state until the deletion is complete. If DELP and DELV are both set, both the delport and delVLANID fields are active. Unsecured unicast addresses meeting both conditions are deleted.

Much like the management adds, having a sticky bit for DEL gives the programmer the opportunity to set up or perform other register operations without having to wait for the delete completion. A polling method is used to find out if the delete is finished. This involves reading adddelcontrol to determine if the command bit used has returned to 0.

interrupts

The TNETX15VE implements interrupts to ease the management processor's tasks. The interrupts are used to indicate changes to the lookup table. It indicates that a new address is added, an address changed ports, an address changed ports and is secure, and an address is deleted due to the aging process. It also indicates that a FIND operation is completed, the RX NBLCK has data (not empty), the statistic registers are half full (and the possibility for an overflow is present) and the lookup table is full. The TNETX15VE indicates that interrupts to the CPU exist by setting to 1 its level-sensitive EINT terminal. The EINT terminal is asserted when any of the possible interrupt conditions are met.

The interrupt register is readable at all times and contains all the current TNETX15VE interrupts. The interrupt register is byte clearable. That is, the interrupt bits in a byte are cleared when that byte is read, regardless of whether or not it is currently masked.

PRINCIPLES OF OPERATION

masking interrupts

The programmer may be interested in processing some interrupts now, while leaving the others for a later time.

The TNETX15VE can mask interrupts. This is accomplished by an interrupt masking register, (intmask). The int and intmask registers have a one-to-one correspondence. The only way EINT is asserted is if both int and intmask are in a 1 state. The logic for the interrupt masking is shown in Figure 17.

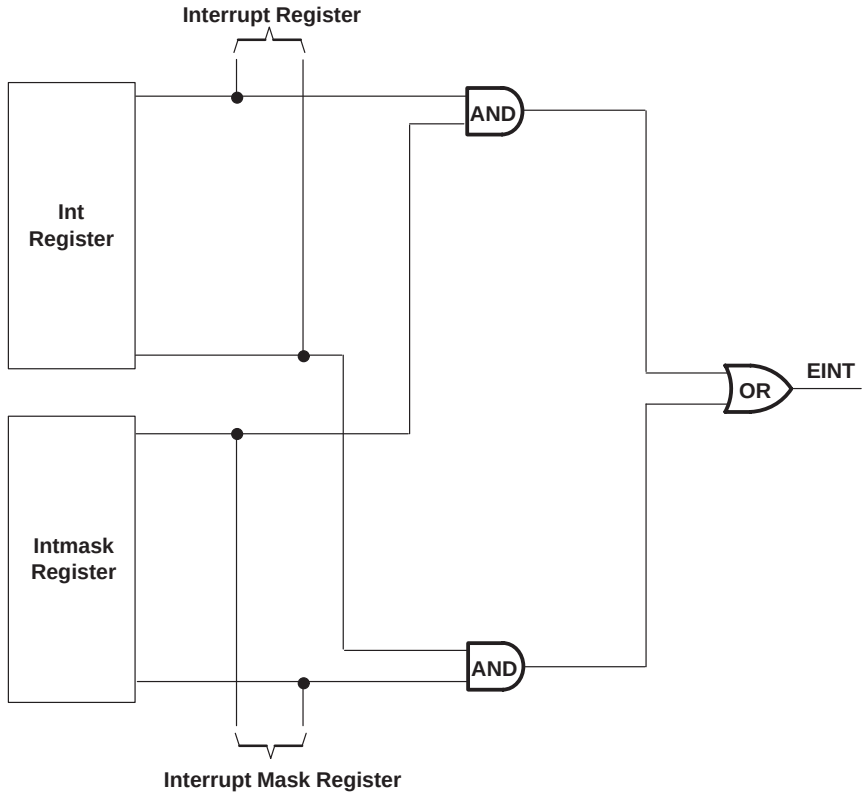


Figure 17. Interrupt Masking Logic

test interrupts (INT)

Test interrupts are generated by asserting the INT bit in the int register. The INT bit in the intmask register must also be set to a 1 state for the interrupt to take effect. The INT bit is used to give the programmer an easy way to test interrupt detection. This bit is the only bit in the int register that is writeable.

PRINCIPLES OF OPERATION

ADD interrupts (NEW, NEWM, CHNG, CHNGM, SECVIO, SECVIOM, and FULL)

The ADD interrupts generate collateral information in the following registers:

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Newnode23–Newnode16	Newnode31–Newnode24	Newnode39–Newnode32	Newnode47–Newnode40	0x30
Newport		Newnode7–Newnode0	Newnode15–Newnode8	0x34

Add interrupts are sourced by the ADD logic operation only based on information from the wire. ADD indicates a new address by a NEW interrupt, an address is changing ports by a CHNG interrupt, and a security violation by a SECVIO interrupt.

The FULL interrupt indicates that ADD needed to start AGE (to free up table space) if NAUTO is 0, or the host needs to delete some addresses if NAUTO = 1.

The add interrupts are indicated in the interrupt register and the information for the particular interrupt is placed in the newnode and newport registers. Since there is only one set of registers that is shared for these interrupts and to ensure that the information placed in these registers is not corrupted during reads, ADD locks the newnode and newport registers until the most-significant byte of newport is read.

Locking these registers means that ADD does not have a place to put information for new events. If additional events are missed, they are indicated in the int register as missed interrupts (NEWM, CHNGM, and SECVIOM).

On a NEW interrupt, the newport register contains information about the node's VLAN and the port for the new address. On a CHNG interrupt, this register identifies the new port address. On a SECVIO interrupt, the address does not move to the port, but the newport register indicates to which port it tried to move.

aging interrupts (AGE, AGEM)

When an address is aged out by the internal AGE logic, it generates an interrupt and reports about that address in the following registers:

BYTE 3	BYTE 2	BYTE 1	BYTE 0	DIO ADDRESS
Agednode23–Agednode16	Agednode31–Agednode24	Agednode39–Agednode32	Agednode47–Agednode40	0x40
AgedVLAN	Agedport	Agednode7–Agednode0	Agednode15–Agednode8	0x44

AGE indicates an interrupt every time that it ages out a node. It places the information on the node (being aged out) on the agednode register, the node's port on agedport, and VLAN in the agedVLAN register. These registers are locked when a new interrupt is given to protect the information contained.

Missed interrupts due to these registers being locked are indicated as an AGEM interrupt. These registers are unlocked when the agedVLAN register is read.

statistic interrupt (STAT)

The statistic interrupt is given when one of the statistic registers (except for numnodes) becomes one-half full; the most-significant bit becomes a 1. This is an indication to the management CPU that the statistic registers should be read to avoid counter overrun. Reading a statistic counter, except for numnodes, clears it.

FIND interrupt (FNDCPLT)

The FIND interrupts are implemented to indicate when the FIND logic operation has completed a FIND command. The software either polls the FIND command bit for a complete indication or it is hardware interrupted by setting the FNDCPLT bit in intmask. This bit is cleared when read.

NBLCK RX FIFO interrupt (RXFIFO)

The NBLCK RX FIFO interrupt is implemented to indicate when the NBLCK FIFO has information in it (not empty). The source address, port, and tag information is placed on this FIFO when the destination address matches an address in the lookup table that has the NBLCK bit set.

IEEE Std 1149.1 test-access port (JTAG)

The test-access port consists of five terminals that are used to interface serially with the TNETX15VE package for boundary-scan testing.

The TNETX15VE is fully JTAG compliant, with the exception of requiring external pullup resistors on the following terminals: TDI, TMS, and $\overline{\text{TRST}}$.

The following instructions, with their 3-bit opcodes, are supported by JTAG.

INSTRUCTION TYPE	NAME	JTAG OPCODE
Mandatory	EXTEST	0000
Mandatory	SAMPLE/PRELOAD	0001
Private	ATPG	0010
Private	Self exercise	0011
Optional	IDCODE	0100
Optional	HIGH Z	0101
Private	PMT	0110
Private	IDDQ	1110
Mandatory	BYPASS	1111

The IDCODE for the TNETX15VE is:

	VARIANT	PART NUMBER	MANUFACTURER	LEAST-SIGNIFICANT BIT
Bit number	31–28	27–12	11–1	0
Binary code	0000b	1011 0001 0110 1011b	000 0001 0111b	1b

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage range, V_{CC} (see Notes 2 and 3) . . . . . –0.5 V to 4 V
Supply voltage range, $V_{CC\ (5V)}$ (see Notes 2 and 3) . . . . . –0.5 V to 5.5 V
Input voltage range, V_I . . . . . –0.5 V to $V_{CC(5\ V)}$ + 0.5 V
Output voltage range, V_O . . . . . –0.5 V to V_{CC}
Thermal impedance, junction-to-ambient package, airflow = 0, $Z_{\theta JA}$ . . . . . 47.5°C/W
Thermal impedance, junction-to-ambient package, airflow = 100 ft/min, $Z_{\theta JA}$ . . . . . 38.2°C/W
Thermal impedance, junction-to-case package, $Z_{\theta JC}$ . . . . . 9.9°C/W
Operating case temperature range, T_C . . . . . 0°C to 95°C
Storage temperature range, T_{stg} . . . . . –65°C to 150°C

† Stresses beyond those listed under “absolute maximum ratings” can cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under “recommended operating conditions” is not implied. Exposure to absolute-maximum-rated conditions for extended periods can affect device reliability.

NOTES: 3. All voltage values are with respect to GND.

4. Turning power supplies on and off (cycling sequence) within a mixed 5-V/3.3-V system is an important consideration. The designer must observe a few rules to avoid damaging the TNETX15VE. Check with the manufacturers of all components used in the 3.3-V to 5-V interface to ensure that no unique device characteristics exist that would lead to rules more restrictive than the TNETX15VE requires.

   The optimum solution to power-supply sequencing in a mixed-voltage system is to ramp up the 3.3-V supply first. A power-on reset component operating from this supply forces all 5-V tolerant outputs into the high-impedance state. Then, the 5-V supply is ramped up. On power down, the 5-V rail deenergizes first, followed by the 3.3-V rail.

   The second-best solution is to ramp both the 3.3-V and 5-V rails at the same time, making sure that no more than 3.6 V exists between these two rails during the ramp up or down. If the 3.3 V is derived from the 5 V, then the 3.3 V rises as the 5 V rises so that the 5-V rail never exceeds the 3.3-V rail by more than 3.6 V. Both the optimum and second-choice algorithms for power up prevent any device damage.

   If it is impractical to implement ramping, follow these rules:

   – When turning on the power supply, all 3.3-V and 5-V supplies should start ramping from 0 V and reach 95 percent of their end-point values within 25 ms. All bus contention between the device and external devices is eliminated by the end of 25 ms.

   – When turning off the power supply, 3.5-V and 5-V supplies should start ramping from steady-state values and reach 5 percent of these values within 25 ms. All bus contention between the device and external devices is eliminated by the end of 25 ms. There is a 250-second lifetime maximum at greater than 3.6 V between the supply rails. Holding this period to 25 ms per power-on/off cycle should not significantly contribute to mean time between failures (MTBF) shifts during product lifetimes.

recommended operating conditions

		MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage	3	3.3	3.6	V
$V_{CC(5V)}$	Supply voltage	4.5	5	5.5	V
V_{IH}	High-level input voltage	2		$V_{CC(5V)}$	V
V_{IL}	Low-level input voltage (see Note 5)	0		0.8	V
I_{OH}	High-level output current			-4	mA
I_{OL}	Low-level output current			4	mA

NOTE 5: The algebraic convention, where the more negative (less positive) limit is designated as a minimum, is used for logic-voltage levels only.

electrical characteristics over recommended operating conditions (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage		V_{CC}–0.6			V
V_{OL}	Low-level output voltage				0.4	V
I_{OZ}	High-impedance-state output current	$V_O = V_{CC}$			20	µA
		$V_O = 0$			–20	
I_{IH}	High-level input current	$V_I = V_{I(MAX)}$			–20	µA
I_{IL}	Low-level input current	V_I = GND			20	µA
I_{CC}	Supply current, 3.3 V				360	mA
$I_{CC(5V)}$	Supply current, 5 V				1	mA

timing requirements (see Note 6 and Figure 18) external SRAM read cycle

NO.			MIN	MAX	UNIT
1	$t_{c(R)}$	Cycle time, read		40	ns
2	$t_{su(ED)}$	Setup time, ED15–ED0 valid before OSCIN↑	11		ns
3	$t_{h(ED)}$	Hold time, ED15–ED0 valid after OSCIN↑	3		ns

operating characteristics over recommended operating conditions (see Note 6 and Figure 18) external SRAM read cycle

NO.	PARAMETER		MIN	MAX	UNIT
4	$t_{d(EA)}$	Delay time, from OSCIN↑ to EA20–EA0 valid		16	ns
5	$t_{d(EOE)1}$	Delay time, from OSCIN↓ to $\overline{EOE}$↓		11	ns
6	$t_{d(EOE)2}$	Delay time, from OSCIN↑ to $\overline{EOE}$↑	3	11	ns
7	$t_{d(EA)}$	Delay time, from EOE↑ to EA20–EA0 transition (change)	0		ns

NOTE 6: $t_{d(EA)}$ and $t_{su(ED)}$ are further characterized over a restricted voltage range on the V_{CC} (3.3 V) rail to allow the use of 15-ns SRAMs. With the values listed, which are valid over a 3.0 V to 3.6 V range (V_{CC}), 12-ns SRAMs are required to meet the system timing. If the V_{CC} (3.3 V) is restricted to the range of 3.4 V to 3.6 V, $t_{d(EA)}$ is reduced to 14 ns, and $t_{su(ED)}$ is reduced to 10 ns. This allows the use of 15-ns SRAMs. Although the operating voltage range of the V_{CC} rail (3.3 V) has a higher minimum, this additional data applies over the operating temperature range of the TNETX15VE.

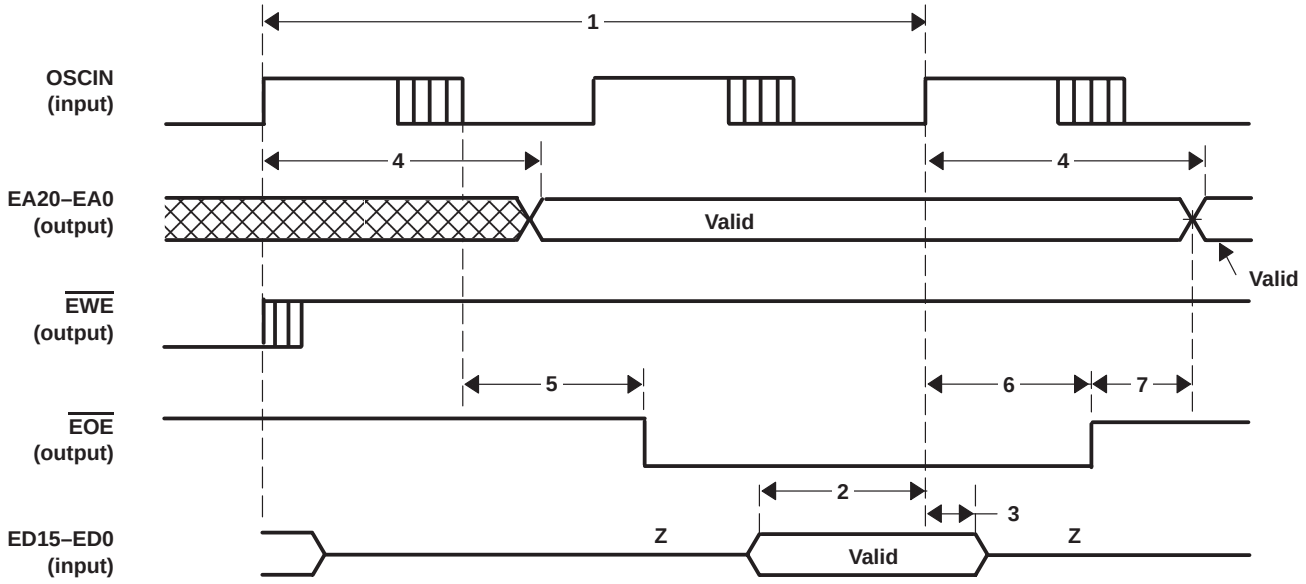


Figure 18. External SRAM Read Cycle

timing requirements (see Figure 19)
external SRAM write cycle

NO.			MIN	MAX	UNIT
1	$t_{c(W)}$	Cycle time, write		40	ns

operating characteristics over recommended operating conditions (see Figure 19)
external SRAM write cycle

NO.	PARAMETER		MIN	MAX	UNIT
2	$t_{d(EA)}$	Delay time, from OSCIN↑ to EA20–EA0 valid		16	ns
3	$t_{d(EWE)1}$	Delay time, from OSCIN↓ to $\overline{EWE}$↓		14	ns
4	$t_{d(ED)1}$	Delay time, from OSCIN↓ to ED15–ED0 valid		14	ns
5	$t_{d(EWE)2}$	Delay time, from OSCIN↑ to $\overline{EWE}$↑	2	12	ns
6	$t_{d(ED)2}$	Delay time, from $\overline{EWE}$↑ to ED15–ED0 invalid	0	10	ns
7	$t_{d(EA)2}$	Delay time, from $\overline{EWE}$↑ to EA20–EA0 transition (change)	0		ns
8	$t_{d(ED)3}$	Delay time, from OSCIN↑ to ED15–ED0 high-impedance Z		16	ns

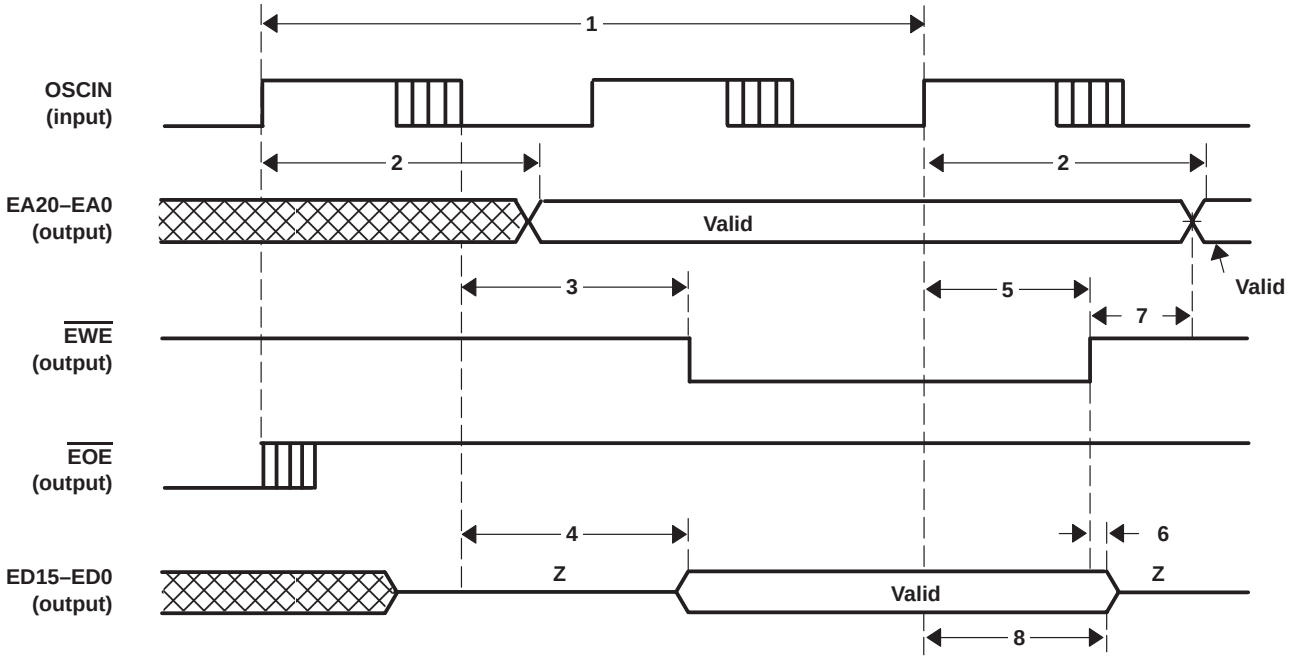


Figure 19. External SRAM Write Cycle

operating characteristics over recommended operating conditions (see Note 7 and Figure 20)
EAM routing code

NO.	PARAMETER		MIN	MAX	UNIT
1	$t_{d(EAM)1}$	Delay time, from EAM15–EAM0 valid to EAM15–EAM0 invalid	100		ns
2	$t_{d(EAM)2}$	Delay time, from OSCIN↑ to EAM15–EAM0 valid		17	ns

NOTE 7: The TNETX3150/TNETX3150A latches the EAM data (at certain edges) on frames that write the first 64 bytes of frame data to the DRAM. The edges where EAM is valid are shown in Figure 20. At other times, the value in EAM is not valid and is ignored by the TNETX3150/TNETX3150A. The TNETX15VE outputs the EAM data on the rising edge of OSCIN before the TNETX3150/TNETX3150A latches the data and holds the data until the end of the transfer burst. This ensures adequate setup and hold times for TNETX3150/TNETX3150As EAM input. The particular frame, where the EAM code is valid, is identified by the start-of-frame (SOF) indicator in the frames flag fields and by the $\overline{DWE}$ strobe being low. The EAM code is not valid on DRAM reads [$\overline{DWE}$ = (1)], on refresh cycles, or on IOB buffer writes and reads.

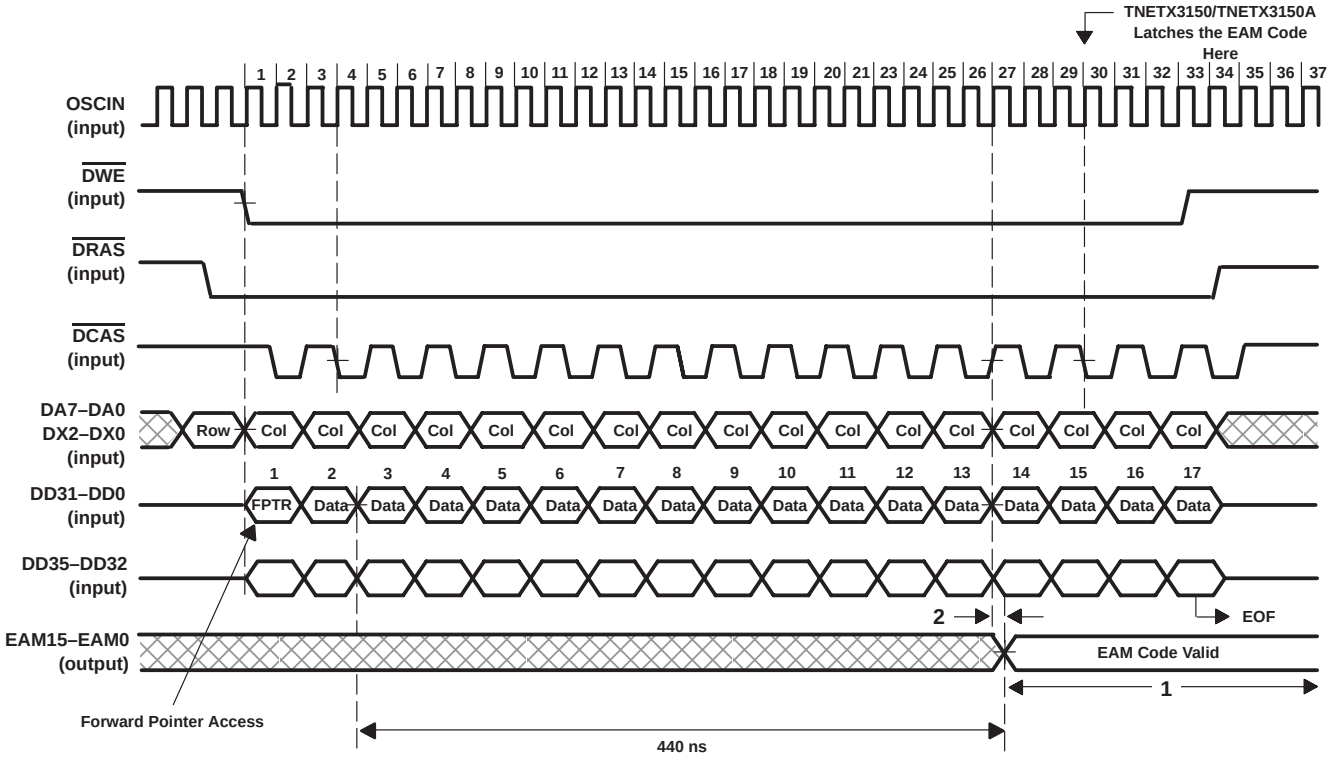


Figure 20. EAM Bus Timing

timing requirements (see Note 8 and Figure 21)
DRAM interface

NO.			MIN	MAX	UNIT
1	$t_{su(DD)}$	Setup time, DD35–DD0 valid before OSCIN↑	12		ns
2	$t_{su(DWE)}$	Setup time, from $\overline{DWE}$↓ to OSCIN↑	4		ns
3	$t_{su(DRAS)}$	Setup time, from $\overline{DRAS}$↓ to OSCIN↑	4		ns
4	$t_{su(DCAS)}$	Setup time, from $\overline{DCAS}$↓ to OSCIN↑	4		ns
5	$t_{h(DD)}$	Hold time, DD35–DD0 valid after OSCIN↑	3		ns
6	$t_{h(DWE)}$	Hold time, $\overline{DWE}$ low after OSCIN↑	3		ns
7	$t_{h(DRAS)}$	Hold time, $\overline{DRAS}$ low after OSCIN↑	3		ns
8	$t_{h(DCAS)}$	Hold time, $\overline{DCAS}$ low after OSCIN↑	3		ns

NOTE 8: The TNETX15VE monitors the DRAM interface to extract frame information. The TNETX15VE extracts information on only the write cycles. The TNETX15VE does not output any signals to the DRAM interface. The only parameters that apply to the TNETX15VE on this interface are delay times, with respect to OSCIN, that translate into the TNETX15VE setup times. The TNETX15VE implements internal delay elements to ensure its setup requirements.

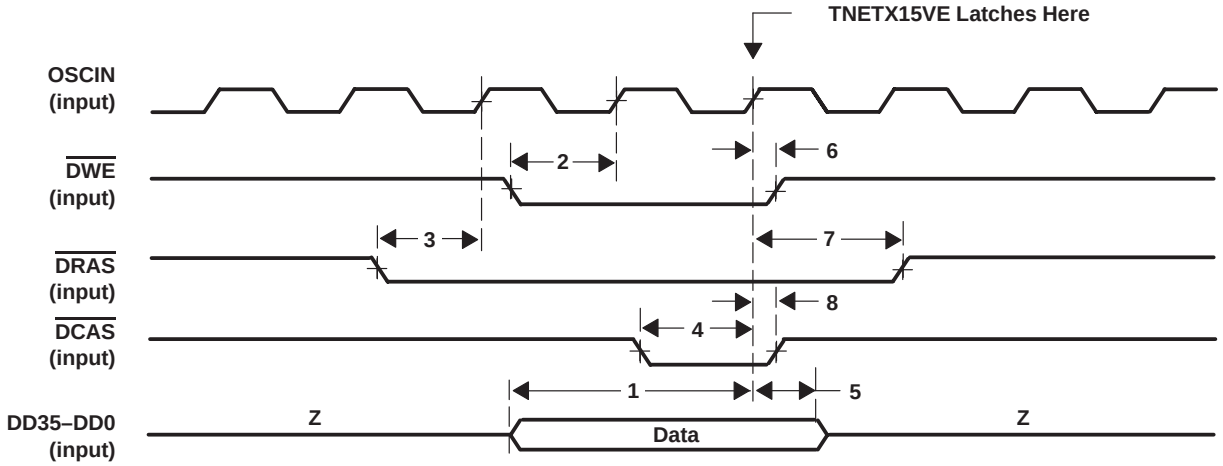


Figure 21. DRAM Interface Timing

timing requirements (see Note 9 and Figure 22)
DIO read cycle

NO.			MIN	MAX	UNIT
1	$t_{su(SRNW)}$	Setup time, SRNW high before $\overline{ESCS}\downarrow$	4		ns
2	$t_{su(SAD)}$	Setup time, SAD1–SAD0 valid before $\overline{ESCS}\downarrow$	4		ns

NOTE 9: The DIO interface is a byte-wide, asynchronous interface that is designed for use with a wide variety of CPUs. The interface can be run at any speed; there is no minimum speed requirement. DIO cycle lengths can vary, depending on the type of register accessed. The TNETX15VE withholds the $\overline{SRDY}$ acknowledge signal on some registers until it has gathered the necessary data.

operating characteristics over recommended operating conditions (see Note 9 and Figure 22)
DIO read cycle

NO.	PARAMETER		MIN	MAX	UNIT
3	$t_{w(SRDYH)}$	Pulse duration, $\overline{SRDY}$ high (see Note 11)		25	ns
4	$t_{d(SRNW)}$	Delay time, from $\overline{SRDY}\downarrow$ to SRNW$\downarrow$	0		ns
5	$t_{d(SAD)}$	Delay time, from $\overline{SRDY}\downarrow$ to SAD1–SAD0 invalid	0		ns
6	$t_{d(SDATA)1}$	Delay time, from SDATA7–SDATA0 valid to $\overline{SRDY}\downarrow$	–7		ns
7	$t_{d(SDATA)2}$	Delay time, from $\overline{ESCS}\uparrow$ to SDATA7–SDATA0 high-impedance Z		17	ns
8	$t_{d(SRDY)1}$	Delay time, from $\overline{ESCS}\downarrow$ to $\overline{SRDY}\downarrow$ (see Note 10)	25+N(20)		ns
9	$t_{d(SRDY)2}$	Delay time, from $\overline{ESCS}\uparrow$ to $\overline{SRDY}\uparrow$ (see Note 11)		25	ns
10	$t_{d(ESCS)}$	Delay time, from $\overline{SRDY}\downarrow$ to $\overline{ESCS}\uparrow$	0		ns

NOTES: 9. The DIO interface is a byte-wide, asynchronous interface that is designed for use with a wide variety of CPUs. The interface can be run at any speed; there is no minimum speed requirement. DIO cycle lengths can vary, depending on the type of register accessed. The TNETX15VE withholds the $\overline{SRDY}$ acknowledge signal on some registers until it has gathered the necessary data.

10. N equals the number of internal cycles required to arbitrate internally to obtain the value being requested. N is equal to a minimum of 1, and can be large during RAM INIT and FINDNEXT operations. Since DIO operations are not complete during these operations (no $\overline{SRDY}$ true), both of these operations can read every SRAM location to finish.

11. These numbers are specified by design but not tested.

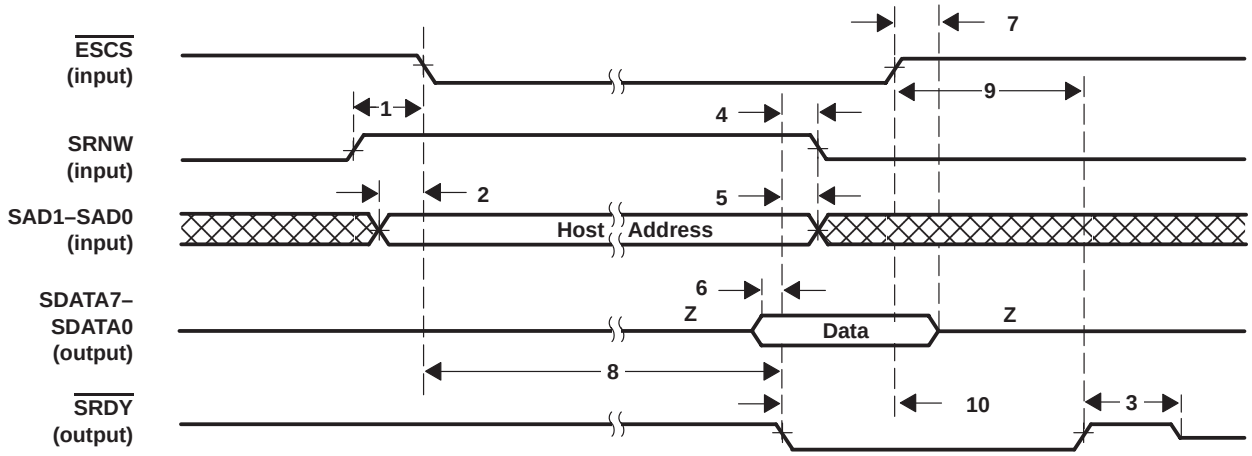


Figure 22. DIO Read Cycle

timing requirements (see Note 9 and Figure 23)
DIO write cycle

NO.			MIN	MAX	UNIT
1	$t_{su(SRNW)}$	Setup time, SRNW low before $\overline{ESCS}\downarrow$	4		ns
2	$t_{su(SAD)}$	Setup time, SAD1–SAD0 valid before $\overline{ESCS}\downarrow$	4		ns
3	$t_{su(SDATA)}$	Setup time, SDATA7–SDATA0 valid before $\overline{ESCS}\downarrow$	4		ns

NOTE 9: The DIO interface is a byte-wide, asynchronous interface that is designed for use with a wide variety of CPUs. The interface can be run at any speed; there is no minimum speed requirement. DIO cycle lengths can vary, depending on the type of register accessed. The TNETX15VE withholds the $\overline{SRDY}$ acknowledge signal on some registers until it has gathered the necessary data.

operating characteristics over recommended operating conditions (see Note 9 and Figure 23)
DIO write cycle

NO.	PARAMETER		MIN	MAX	UNIT
4	$t_{w(SRDVH)}$	Pulse duration, $\overline{SRDY}$ high (see Note 11)		25	ns
5	$t_{d(SRNW)}$	Delay time, from $\overline{SRDY}\downarrow$ to SRNW$\uparrow$	0		ns
6	$t_{d(SAD)}$	Delay time, from $\overline{SRDY}\downarrow$ to SAD1–SAD0 invalid	0		ns
7	$t_{d(SDATA)}$	Delay time, from $\overline{SRDY}\downarrow$ to SDATA7–SDATA0 high-impedance Z	0		ns
8	$t_{d(SRDY)1}$	Delay time, from $\overline{ESCS}\downarrow$ to $\overline{SRDY}\downarrow$ (see Note 10)	25+N(20)		ns
9	$t_{d(SRDY)2}$	Delay time, from $\overline{ESCS}\uparrow$ to $\overline{SRDY}\uparrow$ (see Note 11)		25	ns
10	$t_{d(ESCS)}$	Delay time, from $\overline{SRDY}\downarrow$ to $\overline{ESCS}\uparrow$	0		ns

NOTES: 9. The DIO interface is a byte-wide, asynchronous interface that is designed for use with a wide variety of CPUs. The interface can be run at any speed; there is no minimum speed requirement. DIO cycle lengths can vary, depending on the type of register accessed. The TNETX15VE withholds the $\overline{SRDY}$ acknowledge signal on some registers until it has gathered the necessary data.

10. N equals the number of internal cycles required to arbitrate internally to obtain the value being requested. N is equal to a minimum of 1, and can be quite large during RAM INIT and FINDNEXT operations. Since DIO operations are not complete during these operations (no $\overline{SRDY}$ true), both of these operations can read every SRAM location to finish.

11. These numbers are specified by design but not tested.

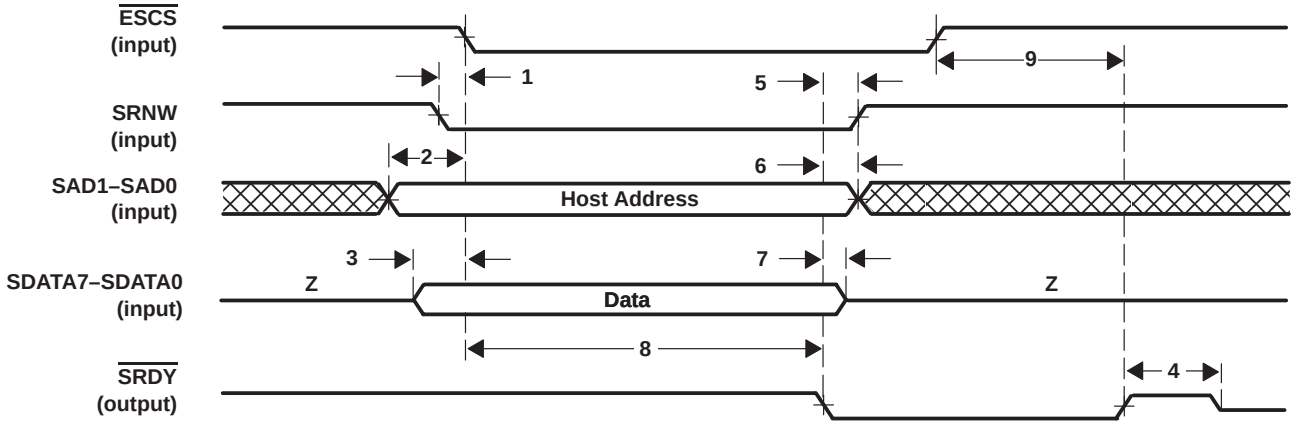


Figure 23. DIO Write Cycle

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

When the host is driving the EEPROM interface through the SIO register (0x0A0), the output terminals are copies of register bits synchronized by OSCIN rising, and the input register bits are copies of the device terminals synchronized by OSCIN rising. This adds a maximum of 20 ns of delay in each direction, which is negligible in terms of the maximum signal change rate allowed for the EEPROMs (approximately 100 kHz), or the MDIO (MII management). The timing for these interfaces is set at the rate that register bits are set in software. For detailed descriptions of the EEPROM requirements, the reader is referred to the data sheet for the part in question. For detailed descriptions of the MII management interface, the reader is referred to the IEEE Std 802.3 specification.

When the TNETX15VE is driving the EEPROM during the automatic register load cycle, the timing on the EEPROM is shown in Figure 24.

operating characteristics over recommended operating conditions (see Note 12 and Figure 24) EEPROM interface timing

NO.	PARAMETER		MIN	MAX	UNIT
	f_{clock}	Clock frequency, ECLK		98	kHz
1	$t_{d(EDIO)1}$	Delay time, from ECLK↑ to EDIO↓	5		µs
2	$t_{d(ECLK)1}$	Delay time, from EDIO↓ to ECLK↓	5		µs
3	$t_{d(EDIO)2}$	Delay time, from ECLK↓ to EDIO invalid	0		µs
4	$t_{d(ECLK)2}$	Delay time, from EDIO valid to ECLK↑	10		ns
5	$t_{d(EDIO)3}$	Delay time, from ECLK↓ to EDIO valid	0		µs
6	$t_{d(EDIO)4}$	Delay time, from ECLK↓ to EDIO invalid	0		µs
7	$t_{d(EDIO)5}$	Delay time, from ECLK↑ to EDIO↑	5		µs

NOTE 12: The following timing refers to the waveforms that the TNETX15VE logic operation outputs while in operation. The EEPROM interface also can be accessed through the SIO register. When accessing the EEPROM through SIO, it is the user's responsibility to ensure proper timing. Proper timing is ensured through software by toggling the appropriate SIO bits at the appropriate time.

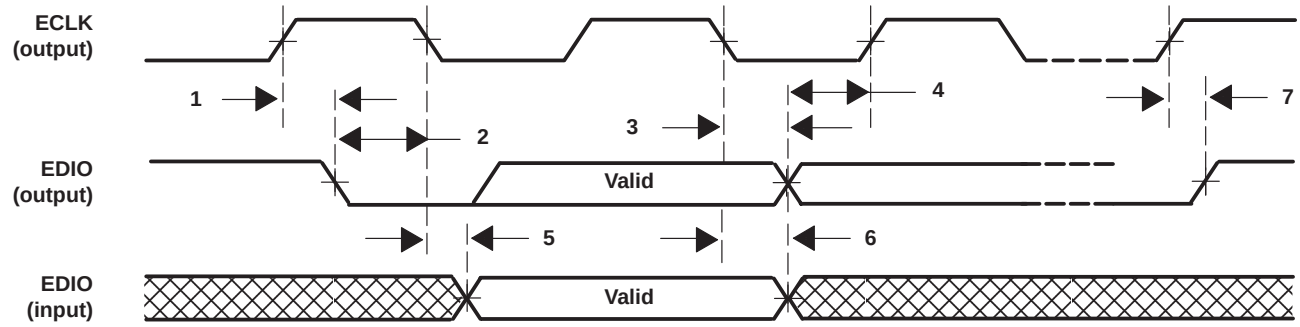


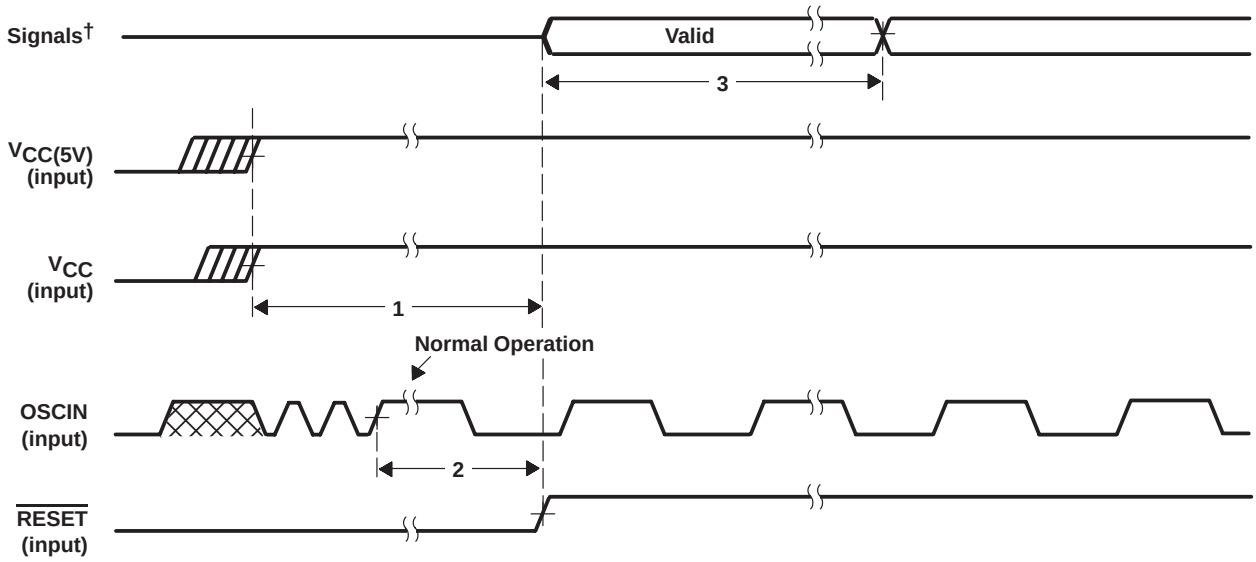
Figure 24. EEPROM Interface Timing

timing requirements
OSCIN clock

		MIN	NOM	MAX	UNIT
$t_{c(OSCIN)}$	Cycle time, $\overline{OSCIN}$	20		20	ns
$t_{w(OSCINH)}$	Pulse duration, OSCIN high	8	10		ns
$t_{w(OSCINL)}$	Pulse duration, OSCIN low	8	10		ns
	Frequency drift, OSCIN clock			±50	ppm

timing requirements (see Figure 25)
power-on reset

NO.			MIN	MAX	UNIT
1	$t_{d(RESET)1}$	Delay time, from $V_{CC}\uparrow$ to $\overline{RESET}\uparrow$	25		ms
2	$t_{d(RESET)2}$	Delay time, from OSCIN↑ to $\overline{RESET}\uparrow$	25		ms
3	$t_{d(AUTO)}$	Delay time, from $\overline{RESET}\uparrow$ to EEPROM autoload finished		50	ms

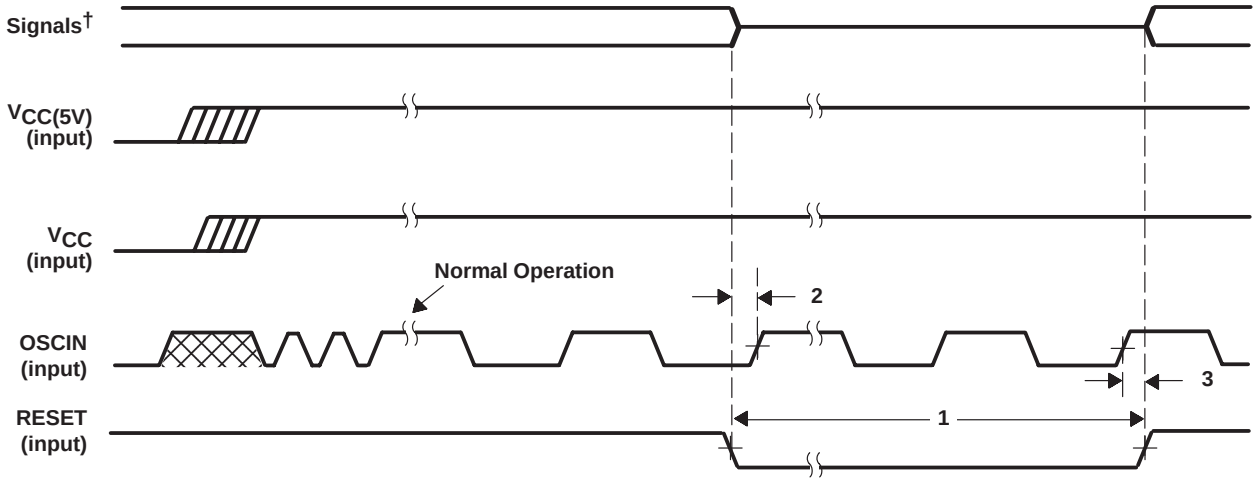


† Outputs are floated during reset. Inputs and I/O's that are not connected to a known logic state at all times, including reset, should have a pullup or pulldown resistor.

Figure 25. Power-On Reset Timing

timing requirements (see Figure 26)
RESET (software) timing

NO.			MIN	MAX	UNIT
1	$t_{w(RESET)}$	Pulse duration, RESET low	3		µs
2	$t_{su(RESET)}$	Setup time, RESET low before OSCIN↑		7	ns
3	$t_{h(RESET)}$	Hold time, RESET low after OSCIN↑		10	ns



† Outputs are floated during reset. Inputs and I/O's that are not connected to a known logic state at all times, including reset, should have a pullup or pulldown resistor.

Figure 26. Reset (Software) Timing

PARAMETER MEASUREMENT INFORMATION

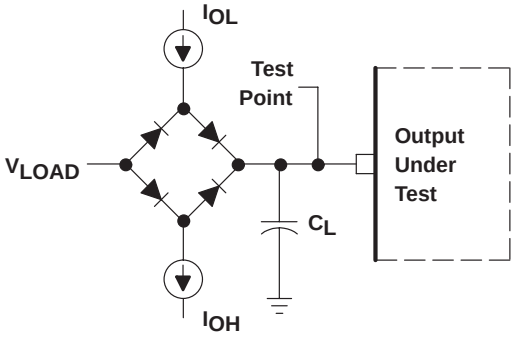
Outputs are driven to a minimum high-logic level of 2.4 V and to a maximum low-logic level of 0.6 V. These levels are compatible with TTL devices.

Output transition times are specified as follows: All transition times are measured at the point where the output signal crosses 1.3 V.

The rise and fall times are not specified but are assumed to be those of standard TTL devices, which are typically 1.5 ns.

test measurement

The test-load circuit shown in Figure 27 represents the programmable load of the tester pin electronics that is used to verify timing parameters of the TNETX15VE output signals.



OUTPUT TEST LOAD

Where: I_{OH} = Refer to I_{OH} in recommended operating conditions.
I_{OL} = Refer to I_{OL} in recommended operating conditions.
V_{LOAD} = 1.5 V, typical
C_L = 25 pF, typical load-circuit capacitance

Figure 27. Test and Load Circuit

APPLICATION INFORMATION

TNETX15VE and TNETX3150/TNETX3150A stand-alone applications

unmanaged switch

The simplest application for the TNETX15VE is shown in Figure 28. This is an unmanaged multiport switch. The TNETX15VE is responsible for matching addresses, learning addresses, and for eliminating (aging out) old addresses. The TNETX15VE also provides options to the manufacturer through its EEPROM. The manufacturer could program this EEPROM through a parallel-port interface to the TNETX15VE. Options that can be set are SRAM size, the aging time, the VLANmask, tagVLAN, and VLANID registers (for this application, they can be all set to the same VLAN and the VLANmask register (for that VLAN) is set to all ports). This is the lowest-cost solution for an unmanaged, VLAN-capable, multinode-per-port switch.

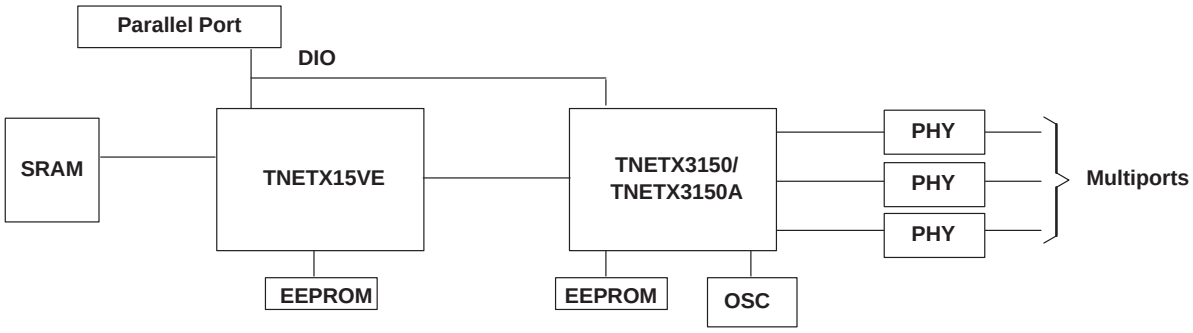


Figure 28. Lowest-Cost Unmanaged Multiport Switch

managed switch

The microprocessor (CPU) interfaces to the TNETX3150/TNETX3150A through a common DIO interface. The microprocessor also has the capability to manage any switch PHY registers through an IEEE Std 802.3u MII interface (SIO register).

The microprocessor's tasks are minimized mainly because the CPU does not have to participate in frame matching. The microprocessor is used to set TNETX15VE and TNETX3150/TNETX3150A modes, secure addresses so that the node does not move ports (useful for routers, attached switches, and servers), support VLANs and spanning-tree options, and respond to control MIB requests.

The TNETX15VE is designed for easy management of the lookup table. Address table lookups, adds, edits, and deletes are performed easily through the TNETX15VE registers. Interrupt support in the TNETX15VE also simplifies the management's tasks. The TNETX15VE gives an interrupt to the CPU when it changes the lookup table. This minimizes code, as the CPU does not have to actively poll a large address table for changes.

APPLICATION INFORMATION

TNETX15VE and TNETX3150/TNETX3150A stand-alone applications (continued)

In-band signaling (see Figure 29) must be done with a MAC device connected to one of the switch ports. There is no access to the data stream via the DIO port on the TNETX3150/TNETX3150A or the TNETX15VE. Both devices affect the ability of a port to receive or forward packets; the packet itself is received and transmitted via a regular switch port.

With access to a port, the CPU can receive frames destined to other nodes. By tagging the CUPLNK bit for that particular address in the address table, the CUPLNK bit copies all frames destined for that address to the ports specified in UPLINKport. By setting UPLINKport to direct these frames to the management CPU, it can receive frames of interest.

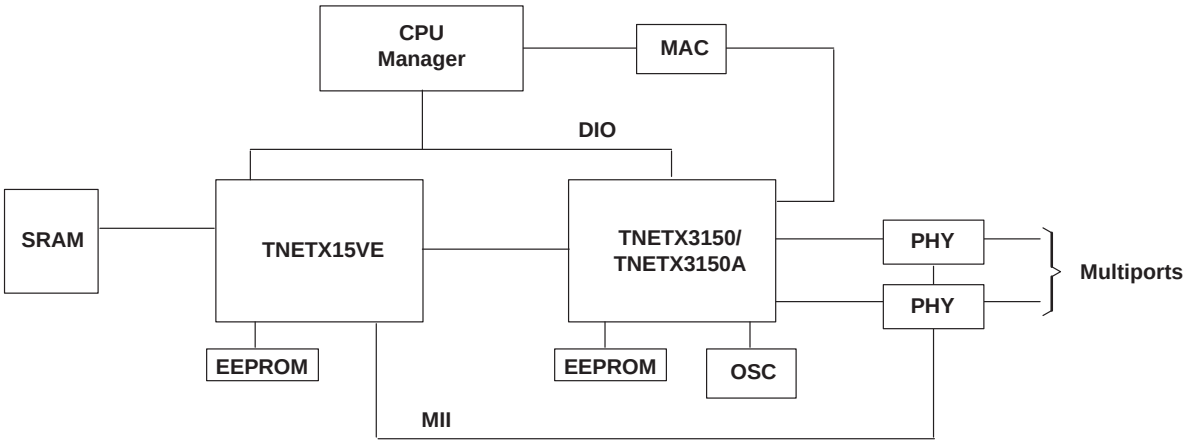


Figure 29. In-Band Managed Multiport Switch

expansion through uplink

The TNETX15VE includes functionality to enable two TNETX3150/TNETX3150As to be cascaded through their port 00. This is accomplished through the TNETX15VE pretag support. The outbound pretag carries port number information, which can be decoded by the inbound TNETX15VE into VLANIDs. The TNETX15VE works in a cascaded system as either a managed or unmanaged device. Figure 30 shows this application. The host processor needs access to the DIO bus of both device sets and needs to coordinate VLAN assignments across both address-lookup tables.

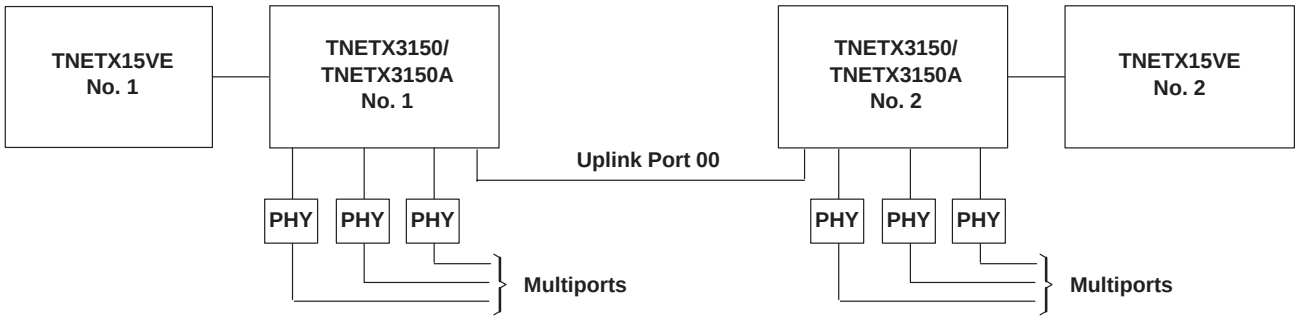


Figure 30. Cascaded Switches Through Uplink With Pretag Support

APPLICATION INFORMATION

third-party interconnect example

The TNETX15VE pretag support enables it to work with proprietary third-party devices connected to the TNETX3150/TNETX3150A through the uplink (port 00). In this mode, the pretag is used to pass along the VLAN information that is required for lookups and an inbound post tag is available for predetermined port delivery. Figure 31 illustrates this application.

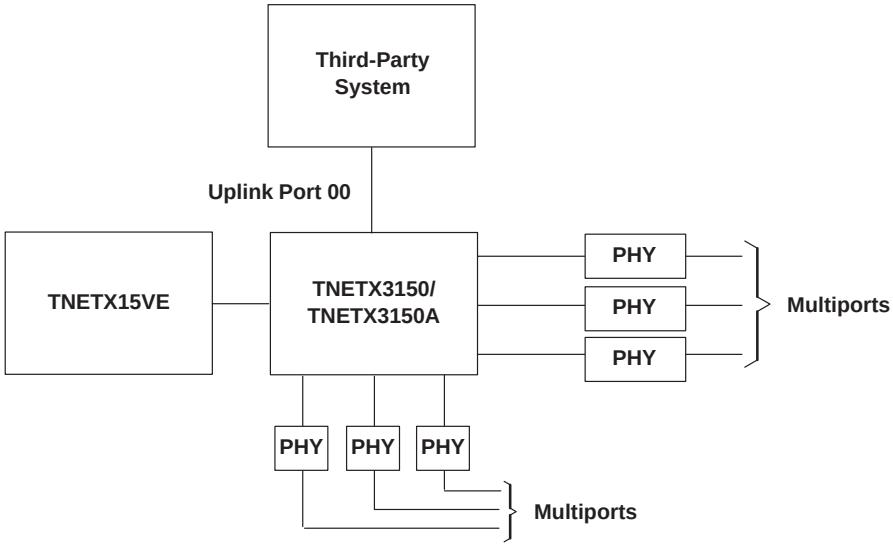


Figure 31. Cascaded With Third-Party Device Through Uplink With Pretag Support

APPLICATION INFORMATION

VLAN support for proprietary systems

In a third-party application the pretag can be used to transfer the VLANID for frames that are sourced by the system to the TNETX3150/TNETX3150A and eventually to the TNETX15VE. Since the pretag is used as an offset in the tagVLAN registers, they must be configured one-to-one so that the tagVLAN offset agrees one-to-one with the VLANID.

The current TNETX15VE implements two bits of VLANID (four possible VLANs), where the pretag field (which is used to index into the tagVLAN array) is four bits. As the designer is deciding on the pretag bit fields, and the entries for the tagVLAN array for translation into VLANids for the inbound packets, there are multiple array entries that can be used in this version of the TNETX15VE. TagVLAN0 to tagVLAN3 is recommended as future versions can be equipped with four bits of VLANid.

VLAN support for proprietary systems (continued)

If the TNETX3150/TNETX3150A and TNETX15VE is the front end for a larger device through port 00 as an uplink, port 00 should be included in all the VLANs in use. This allows stations on the other side of the uplink in each VLAN.

TagVLAN0	=	0x0	VLANID1	=	TNETX3150/TNETX3150A Port 01 VLAN
TagVLAN1	=	0x1	VLANID2	=	TNETX3150/TNETX3150A Port 02 VLAN
TagVLAN2	=	0x2	VLANID3	=	TNETX3150/TNETX3150A Port 03 VLAN
TagVLAN3	=	0x3	VLANID4	=	TNETX3150/TNETX3150A Port 04 VLAN
TagVLAN4	=	Don't care	VLANID5	=	TNETX3150/TNETX3150A Port 05 VLAN
TagVLAN5	=	Don't care	VLANID6	=	TNETX3150/TNETX3150A Port 06 VLAN
TagVLAN6	=	Don't care	VLANID7	=	TNETX3150/TNETX3150A Port 07 VLAN
TagVLAN7	=	Don't care	VLANID8	=	TNETX3150/TNETX3150A Port 08 VLAN
TagVLAN8	=	Don't care	VLANID9	=	TNETX3150/TNETX3150A Port 09 VLAN
TagVLAN9	=	Don't care	VLANID10	=	TNETX3150/TNETX3150A Port 10 VLAN
TagVLAN10	=	Don't care	VLANID11	=	TNETX3150/TNETX3150A Port 11 VLAN
TagVLAN11	=	Don't care	VLANID12	=	TNETX3150/TNETX3150A Port 12 VLAN
TagVLAN12	=	Don't care	VLANID13	=	TNETX3150/TNETX3150A Port 13 VLAN
TagVLAN13	=	Don't care	VLANID14	=	TNETX3150/TNETX3150A Port 14 VLAN
TagVLAN14	=	Don't care			
TagVLAN15	=	Don't care			

VLANmask0 = All ports whose VLANID register is VLAN 0x0 + uplink.
VLANmask1 = All ports whose VLANID register is VLAN 0x1 + uplink.
VLANmask2 = All ports whose VLANID register is VLAN 0x2 + uplink.
VLANmask3 = All ports whose VLANID register is VLAN 0x3 + uplink.

duplicate address support through VLAN

The TNETX15VE VLAN capabilities enable the support of duplicate addresses. Duplicate addresses have the same 48-bit MAC address, but lie in different VLANs. If the address 0x12.34.56.78.9A.BC – VLAN 0x0 is present in the TNETX15VE lookup table, a duplicate (but allowable) address would be 0x12.34.56.78.9A.BC – VLAN 0x01.

The TNETX15VE treats duplicate addresses as independent entities. Each address has its own set of flags, port assignment, and time-stamp information. This means that frames for duplicate addresses are routed differently and are treated differently in the AGE logic operation.

APPLICATION INFORMATION

TNETX15VE VLAN support

The TNETX15VE supports VLAN through its VLANmask, tagVLAN, and VLANID registers. Using these registers, the user can associate each port to a specific VLAN and set the routing options for each VLAN when the address is not found within the lookup table.

The VLANID registers give the VLAN assignment of the ports for the TNETX3150/TNETX3150A that a TNETX15VE is attached to. The tagVLAN registers can be used to either mirror a cascaded TNETX3150/TNETX3150A VLAN port assignment or as a user-defined code that is passed through the uplink (port 00). The VLANmask registers are used to set the routing options for unknown addresses and to mask out multicast-address routing when they are found within the lookup table.

VLAN support for cascaded systems

When two TNETX3150/TNETX3150As are connected in a cascaded system, the VLAN assignment ordinarily is lost when frames traverse through the uplink. To solve this problem, each TNETX3150/TNETX3150A pretags the frame with the source port number. This pretag is shown in the DRAM bus and decoded by the TNETX15VE. Thus, the TNETX15VE knows the source port number for the remote system.

To tie the remote source port number to a VLANID, each TNETX15VE uses the tagVLAN registers. In the TNETX3150/TNETX3150A cascaded systems, the tagVLAN registers enable one TNETX3150/TNETX3150A's TNETX15VE to mirror its TNETX15VE counterpart in the cascaded TNETX3150/TNETX3150A. The tagVLAN registers are set up with the VLANID registers of its partner. Thus, the TNETX15VE No. 1 knows the VLAN assignments for the TNETX15VE No. 2 tables, and vice versa.

The following charts describe how to set up the individual VLAN registers for a cascaded system. TagVLAN0 is a don't care because it maps to the remote TNETX3150/TNETX3150A port 00 that is shared (i.e., in this application there are no frames sourced by the network coming from port 00).

APPLICATION INFORMATION

setup conditions

TNETX3150/TNETX15VE No. 1			TNETX3150/TNETX15VE No. 2		
VLANID1	=	TNETX3150 No. 1 Port 01 VLAN	VLANID1	=	TNETX3150 No. 2 Port 01 VLAN
VLANID2	=	TNETX3150 No. 1 Port 02 VLAN	VLANID2	=	TNETX3150 No. 2 Port 02 VLAN
VLANID3	=	TNETX3150 No. 1 Port 03 VLAN	VLANID3	=	TNETX3150 No. 2 Port 03 VLAN
VLANID4	=	TNETX3150 No. 1 Port 04 VLAN	VLANID4	=	TNETX3150 No. 2 Port 04 VLAN
VLANID5	=	TNETX3150 No. 1 Port 05 VLAN	VLANID5	=	TNETX3150 No. 2 Port 05 VLAN
VLANID6	=	TNETX3150 No. 1 Port 06 VLAN	VLANID6	=	TNETX3150 No. 2 Port 06 VLAN
VLANID7	=	TNETX3150 No. 1 Port 07 VLAN	VLANID7	=	TNETX3150 No. 2 Port 07 VLAN
VLANID8	=	TNETX3150 No. 1 Port 08 VLAN	VLANID8	=	TNETX3150 No. 2 Port 08 VLAN
VLANID9	=	TNETX3150 No. 1 Port 09 VLAN	VLANID9	=	TNETX3150 No. 2 Port 09 VLAN
VLANID10	=	TNETX3150 No. 1 Port 10 VLAN	VLANID10	=	TNETX3150 No. 2 Port 10 VLAN
VLANID11	=	TNETX3150 No. 1 Port 11 VLAN	VLANID11	=	TNETX3150 No. 2 Port 11 VLAN
VLANID12	=	TNETX3150 No. 1 Port 12 VLAN	VLANID12	=	TNETX3150 No. 2 Port 12 VLAN
VLANID13	=	TNETX3150 No. 1 Port 13 VLAN	VLANID13	=	TNETX3150 No. 2 Port 13 VLAN
VLANID14	=	TNETX3150 No. 1 Port 14 VLAN	VLANID14	=	TNETX3150 No. 2 Port 14 VLAN
VLANmask0	=	All ports whose VLANID register is VLAN 0x0. Uplink if at least one tagVLAN register is 0x0.	VLANmask0	=	All ports whose VLANID register is VLAN 0x0. Uplink if at least one tagVLAN register is 0x0.
VLANmask1	=	All ports whose VLANID register is VLAN 0x1. Uplink if at least one tagVLAN register is 0x1.	VLANmask1	=	All ports whose VLANID register is VLAN 0x1. Uplink if at least one tagVLAN register is 0x1.
VLANmask2	=	All ports whose VLANID register is VLAN 0x2. Uplink if at least one tagVLAN register is 0x2.	VLANmask2	=	All ports whose VLANID register is VLAN 0x2. Uplink if at least one tagVLAN register is 0x2.
VLANmask3	=	All ports whose VLANID register is VLAN 0x3. Uplink if at least one tagVLAN register is 0x3.	VLANmask3	=	All ports whose VLANID register is VLAN 0x3. Uplink if at least one tagVLAN register is 0x3.

resulting cross-references

TNETX3150/TNETX15VE No. 1			TNETX3150/TNETX15VE No. 2		
TagVLAN0	=	Don't care	TagVLAN0	=	Don't care
TagVLAN1	=	TNETX3150 No. 2 Port 01 VLAN	TagVLAN1	=	TNETX3150 No. 1 Port 01 VLAN
TagVLAN2	=	TNETX3150 No. 2 Port 02 VLAN	TagVLAN2	=	TNETX3150 No. 1 Port 02 VLAN
TagVLAN3	=	TNETX3150 No. 2 Port 03 VLAN	TagVLAN3	=	TNETX3150 No. 1 Port 03 VLAN
TagVLAN4	=	TNETX3150 No. 2 Port 04 VLAN	TagVLAN4	=	TNETX3150 No. 1 Port 04 VLAN
TagVLAN5	=	TNETX3150 No. 2 Port 05 VLAN	TagVLAN5	=	TNETX3150 No. 1 Port 05 VLAN
TagVLAN6	=	TNETX3150 No. 2 Port 06 VLAN	TagVLAN6	=	TNETX3150 No. 1 Port 06 VLAN
TagVLAN7	=	TNETX3150 No. 2 Port 07 VLAN	TagVLAN7	=	TNETX3150 No. 1 Port 07 VLAN
TagVLAN8	=	TNETX3150 No. 2 Port 08 VLAN	TagVLAN8	=	TNETX3150 No. 1 Port 08 VLAN
TagVLAN9	=	TNETX3150 No. 2 Port 09 VLAN	TagVLAN9	=	TNETX3150 No. 1 Port 09 VLAN
TagVLAN10	=	TNETX3150 No. 2 Port 10 VLAN	TagVLAN10	=	TNETX3150 No. 1 Port 10 VLAN
TagVLAN11	=	TNETX3150 No. 2 Port 11 VLAN	TagVLAN11	=	TNETX3150 No. 1 Port 11 VLAN
TagVLAN12	=	TNETX3150 No. 2 Port 12 VLAN	TagVLAN12	=	TNETX3150 No. 1 Port 12 VLAN
TagVLAN13	=	TNETX3150 No. 2 Port 13 VLAN	TagVLAN13	=	TNETX3150 No. 1 Port 13 VLAN
TagVLAN14	=	TNETX3150 No. 2 Port 14 VLAN	TagVLAN14	=	TNETX3150 No. 1 Port 14 VLAN
TagVLAN15	=	TNETX3150 No. 2 Port 15 VLAN	TagVLAN15	=	TNETX3150 No. 1 Port 15 VLAN

APPLICATION INFORMATION

spanning-tree support

The TNETX15VE is designed to simplify the spanning-tree protocol. The TNETX15VE uses registers that affect routing and learning; these are useful for the spanning-tree states.

learning state

A port in the learning state does not forward frames that are received on this port to the rest of the network. It does not transmit frames that are ordinarily routed to this port. Learning is enabled for this port and the addresses on the port are added to the lookup table.

To place a port in a learning state, its corresponding bit in TXblock and RXblock are set to 1. Its corresponding bit in NLRNport is cleared, which enables learning.

blocking/listening/disabled state

A port in this state does not transmit frames that are sourced by this port to the rest of the network. It does not forward frames that are ordinarily routed to this port. Learning is disabled for this port.

To place a port in a blocking/listening/disabled state, its corresponding bit in TXblock, RXblock, and NLRNport is set to 1.

forwarding state

A port in the forwarding state does not block traffic through the port. Frames that are sourced by this port are routed to the rest of the network. Frames that are routed to this port are transmitted. Learning is enabled for this port and the addresses on the port are added to the lookup table.

To place a port in a learning state, its corresponding bit in TXblock, RXblock, and NLRNports is cleared.

setting up the TNETX15VE to accept BPDUs (determining source port for the BPDUs)

The NBLCK bit in the TNETX15VE is implemented to allow for easy BPDU reception and for determining the source port that originated the frame. To set up the system (to accept BPDUs), the programmer must:

- Program the BPDU addresses into the TNETX15VE with the NBLCK bit set to 1 and the destination address pointing to the port that contains the MAC for the management CPU.
- All BPDU frames are forwarded to the inband CPU. Since the NBLCK bit is set to 1, the TNETX15VE places the source address of this frame in the NBLCK RX FIFO. The NBLCK bit allows the reception of BPDUs when a port is RXblocked.
- On reception of BPDU frames, the source port is read for this frame from the NBLCK RX FIFO. The FIND logic operation is used for this task.

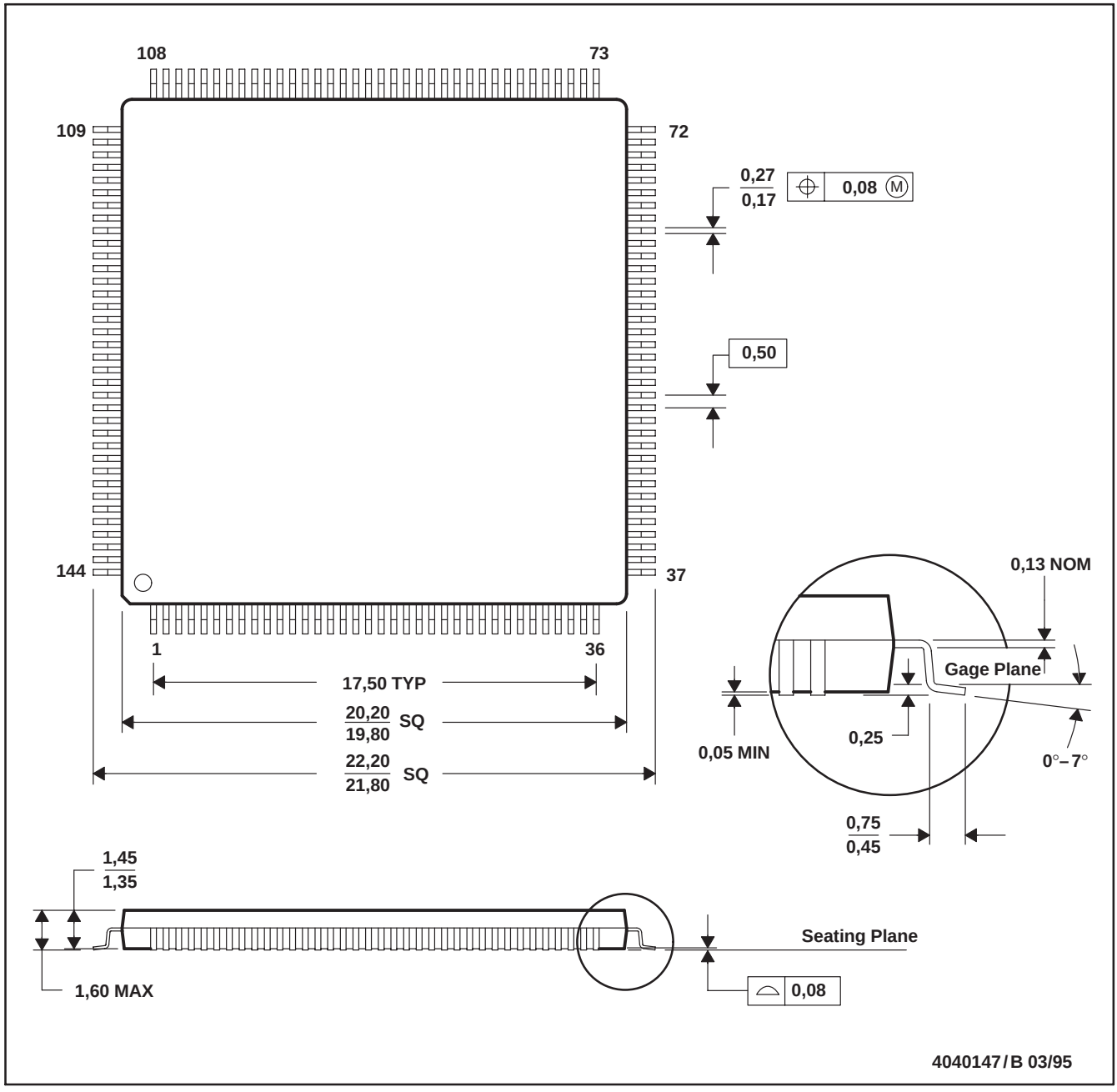
transmitting BPDU frames

The CPUTX register is implemented in the TNETX15VE to allow for easy transmission of BPDU frames. The CPUTX register overrides the LKUP logic operation and the TXblock register. In this way, transmissions to ports that have been TXblocked are performed. To transmit a BPDU frame to a specific port, the programmer must:

- Program the CPUTX register to point to the port where the BPDU frame comes from (usually the inband CPU is connected to this port).
- Program the CPUTX register to point to the port where the BPDU frame is to be transmitted.
- Transmits the frame. The frame enters the TNETX3150/TNETX3150A, goes out on the DRAM bus, and the TNETX15VE matches the source port. When the source port is matched, the TNETX15VE directs the frame to the destination port in CPUTX. It then disables the CPUTX register.
- Repeat the procedure for additional frames.

MECHANICAL DATA

PGE (S-PQFP-G144) **PLASTIC QUAD FLATPACK**



NOTES: A. All linear dimensions are in millimeters.
B. This drawing is subject to change without notice.
C. Falls within JEDEC MO-136

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.