AT/LANTIC™ Software Developer's Guide

National Semiconductor
Application Note 887
David Milne
May 1993

INTRODUCTION

This document is designed to aid the development of software for the AT/LANTIC device. It is recommended that the AT/LANTIC data sheet be read before and then in conjunction with this description.

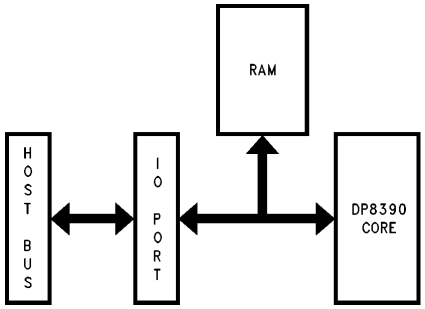
Table of Contents

1.0 INTRODUCTION TO THE AT/LANTIC

The AT Local Area Network Twisted Pair Interface Controller provides a simple method of interfacing any ISA (Industry Standard Architecture) bus based systems to an Ethernet Network. This device can emulate one of the most popular Ethernet Adapter architectures—Novell's NE2000 adapter.

The configuration information describing the devices architecture, address, interrupt etc. can either be loaded from switches or from an EEPROM. Use of the EEPROM method allows the device to be configured solely by software thus providing a more user friendly Ethernet adapter.

A brief introduction to the two Adapter architectures is given below.

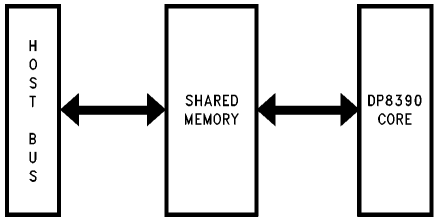


TL/F/11820–1

FIGURE 1. NE2000 Emulation Mode

The NE2000 mode utilizes a Data Port register through which all transfers to/from the buffer RAM take place. Any transfer requires the buffer RAM address, transfer size and direction to be programmed into registers before the transfer can be initiated.

This mode of operation is often referred to as I/O Mode.



TL/F/11820–2

FIGURE 2. Shared Memory Mode

In Shared Memory mode the buffer RAM is mapped into system memory. This allows any data in the buffer RAM to be directly transferred across the ISA bus.

The AT/LANTIC requires a 20H byte space in the PC's I/O Port map, this area contains all of the AT/LANTIC's registers. The contents of this register block depend on the mode the AT/LANTIC is operating in. One common factor is the NIC core register section which contains 16 registers. The location of the register block is given by "I/O address" (also referred to as I/O base). *Figures 3* and *4* show the contents of the register block for either mode.

The AT/LANTIC can access a RAM area of up to 64 kbytes, however the standard is to only use 16 kbytes of this area.

AT/LANTIC™ is a trademark of National Semiconductor Corporation.

 TL/F/11820 RRD-B30M75/Printed in U. S. A.

The method of accessing this RAM area is described in Section 4.0 of this document. The AT/LANTIC's memory map depends on the mode of the device. *Figures 5* and *6* show the full 64 kbyte memory map for each mode (each mode with the chip in compatible mode utilizing only 16 kbytes for the RAM Buffer).

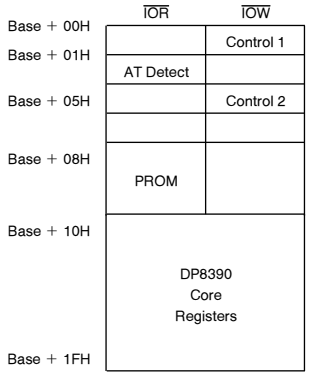


FIGURE 3. Shared Memory Mode I/O Map

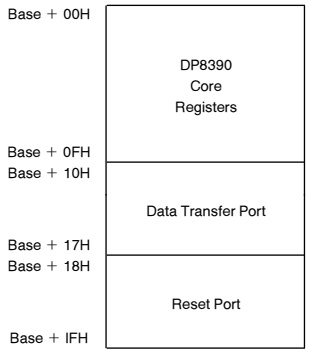


FIGURE 4. I/O Port Mode Register I/O Map

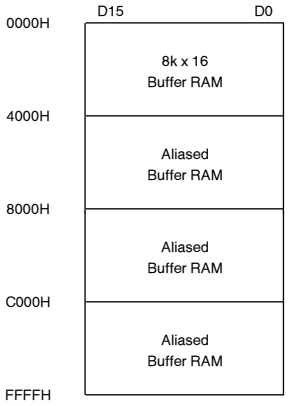


FIGURE 5. Shared Memory Mode Memory Map

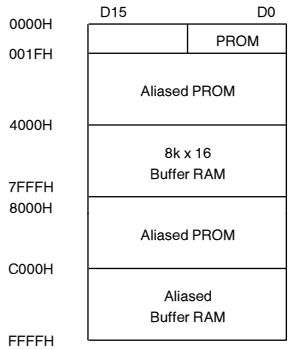


FIGURE 6. I/O Port Mode Memory Map

2.0 CONFIGURING THE AT/LANTIC

The AT/LANTIC controller is designed such that it is fully software configurable. The configuration information is held in three registers A, B and C as described in the data sheet under Section 5.1. As an added safety feature configuration registers A and B are hidden, so that they cannot be accidentally overwriften. Register C is only accessed during a RESET and can not be directly accessed by software.

Register A controls the I/O address, interrupt and mode of the AT/LANTIC device. Register B controls the cable type selection and certain flags altering some interface timings to the ISA bus (refer to Section 4.1 ''Bus Error Condition'').

2.1 Changing and Saving a Configuration

CHANGING THE CONFIGURATION

Registers A and B can be read directly at I/O address offset 0AH and 0BH respectively. To write to these registers a read access must be immediately followed by a write access. Interrupts should be disabled during the write sequence to ensure it is not corrupted. These registers can only be accessed when the NIC command register is set to page 0, refer to AT/LANTIC data sheet Section 5.3.

As register A can change the address location of the AT/LANTIC registers (and hence of reg. B) then a software update of both registers A and B should first change register B and then register A. This allows the same base I/O address to be used for both register updates.

THE GDLNK BIT

Special care should be taken when configuring the GDLNK bit of register B. If this bit is set to 1 then the link integrity checking (TPI mode) is disabled. If link integrity checking has not been disabled (10BT standard) then this bit reads 1 for good link and 0 for link broken. If the AT/LANTIC is in TPI mode with a good link and reg. B is read then the GDLNK bit is shown to be 1, if this was written directly back to reg. B then link integrity checking is disabled. Thus it is necessary to mask out the GDLNK bit when writing to reg. B unless the disabling of link integrity checking is required.

SAVING A CONFIGURATION

The AT/LANTIC has a feature allowing the required configuration to be saved to an EEPROM such that on power up the configuration registers are automatically loaded with the correct values. There is a special algorithm which writes the configuration to the EEPROM, this is described in the pseudo code below. This algorithm does not change the registers directly, i.e. the new state only appears on the next power up.

```
CONFIGURATION_SAVE ()
{
//Interrupts are disabled to ensure the
// sequence is not corrupted.
      Disable interrupts;

// Set EELOAD bit in register B.
      value = READ (Config_Reg_B);
      value = value & (~GDLNK);
      value = value | EELOAD;
      WRITE (Config_Reg_B, value);

// Output the configuration info.
      READ(Config_Reg_B);
      WRITE(Config_Reg_B, config_for_A);
      WRITE(Config_Reg_B, config_for_B);
      WRITE(Config_Reg_B, config_for_C);

// Wait for EELOAD bit to go low.
      while(value && EELOAD)
      {
            value = READ(Config_Reg_B);
            WAIT();
      }
      Enable Interrupts;
}
```

2.2 Enabling a "New" Adapter

It is possible to place the AT/LANTIC controller in a "disabled" state in which it shall not respond at any I/O base location. This is a particularly useful mode for software configurability as it allows the auto selection of an available configuration before the AT/LANTIC based adapter is enabled. Hence potential conflicts of Interrupt and I/O base address can be avoided.

The method of enabling the AT/LANTIC from this "disabled" state consists of writing a byte four consecutive times to port 278H, during which time interrupts should be disabled to ensure the sequence is not corrupted. The lower 3 bits of this byte inform the AT/LANTIC of which address it should enable to. More information on the mapping of these three bits to I/O base locations can be found in the AT/LANTIC data sheet under Section 5.1.

The port 278H is normally a PC's secondary printer port. Using the "four writes" sequence ensures that an adapter is not accidentally enabled if the port is in use. If the port is active when the adapter is to be enabled then it is possible to corrupt a print sequence, to avoid this the code should check if the port is active and if so wait until the port is free. The printer uses port 278H as a data port and port 279H as a control port, reference should be made to the PC's technical manual for an explanation of these registers and how the port operates.

Once the AT/LANTIC is enabled configuration register A may contain old information in bits 3–7. Register A bit 7 is the MEMIO (architecture) bit, the offset of the configuration registers varies depending on the state of this bit. Thus the software has to detect which architecture mode the AT/LANTIC has appeared in before bits 3–7 of reg. A and all of reg. B can be overwritten with the new configuration information.

DETECTING AT/LANTIC MODE

The AT/LANTIC can appear in either the I/O port mode or the Shared Memory mode. As the I/O address is known then the mode of operation can be found by checking the data at the address of register A, i.e., check offset 0AH for I/O mode and offset 1AH for Shared Memory mode. The recommended method of detecting the architecture of an AT/LANTIC at a known I/O base is given in the pseudo code below.

```
FIND_MODE()
{
// Check if in I/O mode
      config_a = READ(IO_BASE + 0AH);
// Check MEMIO is low i.e. I/O mode.
      if((config_a & 80H) ==0)
      {
// Check IOAD bits match the I/O base addr.
      if((config_a & 7) relates to lO_BASE)
      return(IO_MODE_DETECTED);
      }
// Check if in Shared Memory mode.
      config_a = READ(IO_BASE + lAH);
// Check MEMIO is high i.e. S/M mode.
      if((config_a & 80H) == l)
      {
// Check IOAD bits match the I/O base addr.
      if((config_a & 7) relates to IO_BASE)
         return(SHARED_MEM_DETECTED);
      }
// No AT/LANTIC_ mode detected.
return(NO_MODE_DETECTED);
      }
```

2.3 Programming Configuration Register C

Configuration register C can not be accessed directly by software. Details on what configuration register C controls can be found in the AT/LANTIC data sheet Section 5.1. The upper four 4 bits of register C are fixed depending on the design of the adapter card, the lower four bits vary depending on the boot ROM option selected. It is necessary to know the boot ROM option selected so that register C is correctly updated by the "Configuration__Save" routine (as given under Section 2.1). As register C can not be read some mechanism is required to detect the boot ROM option in use. The recommended method is to place a signature text string in the boot ROM at a fixed location. This string would contain information on the size of ROM in question. The code should then scan the RAM space for this signature string. If found, then the location and size of the boot ROM is known and the value required for register C can be calculated. As register C can not be written to directly, the only method of updating it is to use the "Configuration—Save" routine.

It should be noted that any change to the value of register C is not active until the AT/LANTIC has been reset (normally a power off/on of a PC) and the new contents of the EEPROM have been loaded into the registers.

3.0 INITIALIZING THE AT/LANTIC

The following section deals with all of the functions necessary to initialize and partly test the AT/LANTIC device. It is recommended that the code follows the order of each of the sub-sections.

3.1 Hardware Reset

The first step in the initialization sequence is to provide a reset pulse to the NIC core of the AT/LANTIC device. The method of providing this signal depends on the architecture selected for the AT/LANTIC.

I/O MODE RESET SEQUENCE

In this mode a portion of the I/O address map acts as a reset port, offsets 18H to 1FH, any of these offsets can be used as the reset port. To activate a reset the port should be read from and then written to (with any value). Following this a delay of 1.6 ms is required to make sure the reset has completed.

SHARED MEMORY RESET SEQUENCE

This mode contains two control registers (refer to Section 5.2 of the AT/LANTIC data sheet). The MSB of control register 1 is the RESET flag. To activate the reset this bit should be toggled high then low. Again a delay of 1.6 ms is required to allow the reset to complete.

PLACING THE AT/LANTIC IN STOP MODE

Following the hardware reset it is necessary to place the NIC core of the AT/LANTIC in the STOP mode, which performs a software reset. This is done by setting the STOP and RS2 bits of the NIC command register (refer to the AT/LANTIC data sheet Section 5.3 for more information on the NIC core registers), PS0 and PS1 are cleared to ensure that the NIC core is in ''page 0''. (The NIC core registers are split into 3 pages of 16 registers, the bits PS0 and PS1 in the command register define which page the NIC is currently operating in.) The code should wait until the software reset is indicated as complete (when the RST bit of the Interrupt Status register is set to a 1.

3.2 Get Bus Size/ID Bytes

This section deals with detecting the size of the slot an adapter has been inserted into and checking ID byte values are correct for the mode of operation selected. The two AT/LANTIC modes are quite different in their approach to this problem and shall be discussed separately.

I/O MODE

The I/O mode architecture maps PROM data into RAM space locations 00H to 1FH. The contents of this RAM space are given in *Figure 7.* Offset 1EH contains a word value which depends on the size of slot in which the adapter currently resides. If the value equals 5757H, then the slot is 16-bit, if the value equals 4242H, then the slot is 8-bit. If the value does not equal either of the previous values, then the adapter is not correctly functional in NE2000 emulation mode, i.e., these bytes also act as ID bytes. An example of the DMA read operation required to get the word from offset 1EH is described in Section 4.1.2 of this document. The initial transfer requires the Data Configuration Register (DCR) to be programmed for 8-bit operation, and the PC to perform two byte wide reads of the Data Port. Once the data has been transferred the DCR and PC transfers can be set for the correct bus width.

SHARED MEMORY MODE

The shared memory mode contains a register which holds information on the size of slot occupied by the adapter. The LSB of this register, the AT detect register (refer to AT/LANTIC data sheet Section 5.2) is set to 1 for 16-bit mode and 0 for 8-bit mode.

Correct emulation of the Ethercard Plus16 requires an ID byte. In this mode the registers at offsets 08H to 0FH (see *Figure 8*) contain first the Ethernet address then an ID byte followed by a checksum byte.

For the AT/LANTIC shared memory mode the ID byte at offset 0EH should contain value 05H. The twos complement sum of all of the eight bytes should equal FFH. If this is not the case, then the adapter is not correctly operating in Ethercard Plus16 emulation mode.

00H	ETHERNET ADDR. 0
01H	ETHERNET ADDR. 1
02H	ETHERNET ADDR. 2
03H	ETHERNET ADDR. 3
04H	ETHERNET ADDR. 4
05H	ETHERNET ADDR. 5
06H	(BOARD ID BYTE)
07H	(CHECKSUM)
1EH	42H or 57H
1FH	42H or 57H

FIGURE 7. NE2000 Mode PROM Memory MAP

0BH	Addr. Byte 0
	Addr. Byte 1
	Addr. Byte 2
	Addr. Byte 3
	Addr. Byte 4
	Addr. Byte 5
	Board ID By1e
0FH	Checksum

FIGURE 8. S/M Mode PROM Loaded Registers

3.3 Initializing the Registers

When initializing an adapter for Shared Memory mode it is necessary to set up two control registers before initializing the NIC core registers, the following step is ignored for I/O Port mode.

SHARED MEMORY ADDRESS INITIALIZATION

In shared memory mode the Buffer RAM is mapped into the PC address space. Two control registers define where in the PC memory map this RAM shall appear, refer to Section 5.2 of the AT/LANTIC data sheet. Address bits A13–A18 of the selected memory address should be programmed into bits 0–5 of control register one. Address bits A19–A23 should

be programmed into bits 0–4 of control register two. The MEMW bit of control register 2 defines if the memory is 8 kwords or 8 kbytes in size. This bit should be defined during this initialization. It may also be necessary to initialize the MEME bit of control register 1, this depends on the method of controlling the buffer RAM—refer to Section 4.2.

e.g. D0000H, 16 kbytes memory.
Control reg. 1 is 28H,
Control reg. 2 is 41H.

NIC CORE REGISTER INITIALIZATION

The following register initialization sequence is mandatory for both I/O Port and Shared Memory modes. All of the registers discussed are explained in detail under Section 5.3 of the AT/LANTIC data sheet.

1. Put the AT/LANTIC in STOP mode. Refer to Section 3.1 on putting the AT/LANTIC in STOP mode.
2. Initialize Data Configuration Register (DCR), WTS (transfer width) dependent on the result of the previous section, LS set, ARM is dependent on the method chosen to control the receive buffer ring (refer to Section 6.0). The FIFO threshold is typically set to 8 bytes/4 words i.e., FT1 is set. (When in Shared Memory mode the WTS bit can always be set if 16 kbytes of RAM are being used.)
3. Clear Remote Byte count registers.
4. Initialize Receive Configuration Register (RCR), this register determines which packets are accepted by the AT/LANTIC and buffered into RAM. The options available are save errored packets, runt packets, multicast packets, broadcast packets, physical address match packets and all physical address packets (promiscuous mode). Setting the register to 00H allows only physical address match packets. Further discussion on setting the physical and multicast addresses is given later in this section.
5. Place the NIC in loopback mode 1 or 2 by setting bits LB0 or LB1 in the Transmit Configuration Register. Loopback is described in the data sheet under Section 6.5.
6. Initialize the Receive Buffer Ring,
   Boundary Pointer (BNDRY),
   Page Start (PSTART),
   Page Stop (PSTOP),
   Transmit Page Start (TPSR).

   The values for these registers are discussed at the end of this section.
7. Clear the Interrupt Status Register (ISR), by writing FFH to it.
8. Initialize Interrupt Mask Register (IMR), this register controls which sources of interrupt are allowed or disallowed. It would be normal to set PRXE (packet received), PTXE (packet transmitted), PTXEE (packet transmission error) and OVWE (overflow) in the register.
9. Program the Command Register for page 1 (set bits PS0, RD2 and STP) and initialize the physical, multicast and CURR registers as described later in this section.
10. Put the NIC in to START mode, set the START bit and clear the STOP, PS0 and PS1 bits in the Command Register. The receive is still not active as the NIC core is in loopback mode.
11. Remove the AT/LANTIC from loopback mode by initializing the Transmit Configuration Register (TCR) to its correct value, typically 00H.

The AT/LANTIC is now ready to receive and transmit packets.

SETTING UP THE BUFFER RING

The values to be programmed in to the buffer ring pointers PSTART, PSTOP, CURR, BNDRY and TPSR depend on the mode of operation of the AT/LANTIC. There are several possible modes NE2000 and Ethercard Plus16 emulation modes which use 16k of Buffer RAM and a non-compatible mode which allows up to 64k of Buffer RAM. The size of transfers in question also alter the size of the Buffer Ring.

The buffer RAM is split into ''pages'' each containing 256 bytes. The standard is to have a transmit buffer followed by the receive buffer ring. It is recommended that two transmit buffers be utilized as this improves performance. One packet can be loaded into the RAM while another is being transmitted on the network. Each transmit buffer requires 6 ''pages'' for a full Ethernet packet (some protocols may not require a maximum Ethernet packet). The Transmit Page Start Register (TPSR) should point to the buffer being transmitted for the whole duration of a transmission, i.e., when the alternate buffer is being loaded during a transmission the TPSR should not be altered. The receive buffer ring follows the transmit buffer. The Page Start register (PSTART) is set to the page following the transmit buffer(s). The Page Stop (PSTOP) register is set to the end of the buffer RAM plus one page (the size of buffer RAM is determined by the bus width/memory width.)

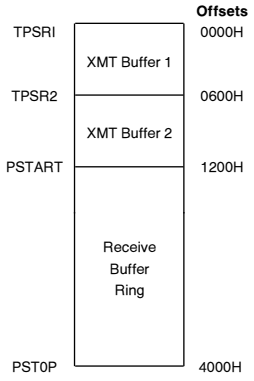


FIGURE 9. Buffer RAM Layout

In NE2000 emulation mode the buffer RAM resides at locations 4000H to 8000H (refer to *Figure 6*). The Ethercard Plus16 buffer RAM resides at 0000H to 4000H (refer to *Figure 5*).These examples are for 16-bit modes.

There are two possible methods of controlling the receive buffer ring a software method and an automated method called ''send packet''. These are discussed in more detail in Section 6.0 of this document.

Initializing CURR and BNDRY for

(i) the software method,
CURR = PSTART + 1
BNDRY = PSTART
Next_PKT = PSTART + 1

(ii) the ''send packet'' method,

CURR = BNDRY = Next__PKT = PSTART

(''Next__PKT'' is a software declared variable further described in Section 6.0).

SETTING THE PHYSICAL ADDRESS REGISTERS

During initialization, the NIC core physical address registers are loaded with the adapters Ethernet address (refer to Section 5.3 of the AT/LANTIC data sheet). Each adapter contains a unique six byte address for identification on a network. The Ethernet address is held in a PROM store in the AT/LANTIC, the method of retrieving the information depends on its present architecture mode. In shared memory mode the Ethernet address is held at register offsets 08H to 0DH, see *Figure 3.* These values can then be written to the NIC physical address registers. In I/O mode the address is held in the first three words of the RAM thus a Remote DMA read is required to retrieve the information, refer to Section 4.1.2.

SETTING THE MULTICAST REGISTERS

To allow a network station to receive packets destination addresses other than the stations physical node address, it is necessary to store a list of these destination addresses. A group of addresses to be received are referred to as multicast addresses. This device can not hold all of the addresses, thus the AT/LANTIC contains 8 multicast address registers (MAR0–7) which decode the addresses to be received. These multicast registers provide filtering of multicast addresses hashed by the CRC logic. All destination addresses are fed through the CRC logic and as the last bit of the destination address enters the CRC, the 6 MSB's of the CRC generator are latched. These six bits are then used to index a unique filter bit (FB0–63) in the multicast address registers. When a software developer wishes to accept a specific multicast address the above sequence should be followed to determine which filter bit in the multicast registers should be set. Several bits can be set to accept several multicast addresses. A pseudo code example of the routine required for this is given below.

```
// Hexadecimal equivalent of the NIC's CRC
// equn.
define CRC_POLYNOMlAL 04ClldB6H

// The multicast address is held in a 6 byte
// array.
unsigned char mult_addr[6];
crc = FFFFFFFFH;

// The following loops create the 32-bit CRC
// value.

// Loop through each byte of the address.
for(i=0; i<6; i++)
{
// Loop through each bit of that byte.
for(bit=0; bit<8; bit++)
  {
  carry=(crc3l )^((mult_addr[i] &
    (l<<bit )) >> bit);
  crc <<= l;   if (carry)
    crc = ((crc^CRC_POLYNOMIAL) | carry);
  }
}

// Extract the 6 MSB's from the CRC value,
// this six bit value is used to index a
// unique filter bit.
index >>= 26;
index &= 3FH;
// Find the multicast register number and
// bit of that register to set.
register_no = crc >> 3;
register_bit = l << (crc & 7);

// Calculate the new register value.
value = READ(MAR[register_no]) |
register_bit;

// Set the Multicast Address Register (MAR)
// value.
WRITE (MAR[register_no], value);
```

3.4 Checking the Cable

There are four possible media types that can be selected with the AT/LANTIC device. Both the TPI(10BT) and TPI(non spec.) modes simply use the GDLNK bit of configuration register B to indicate the cable status. The test for a good ''Thin'' Ethernet cable is more involved and a pseudo code description of what is required is given below. Thick Ethernet cable can be checked by performing level 3 loopback as described in Section 6.5 of the AT/LANTIC data sheet. However this test can only be guaranteed if performed on a non-active network, i.e. no information is being passed on the network during the test.

```
// Assume already initialized.

thin_cable_check()
{
// Set up a cable check packet for
// transmission. The destination addr.
// should equal the source address to avoid
// other stations receiving this pkt. The
// data field is of the developers choice.
      packet = set_up_cable_pkt();
      length = size of (packet);
```

```
// Set the transmit byte count registers.
      WRITE(TBCR0, length_LSB);
      WRITE(TBCR1, length_MSB);

// Clear the Interrupt Status Register.
      WRITE(INTSTATUS, FFH);

// Issue the transmit command.
WRITE(CMND, (RD2 | STA | TXP));

// Poll the interrupt status register until
// the packet is indicated as transmitted
// or there is a timeout.
      while(time_in_loop < 1 second)
      {
       STATUS = READ(INTSTATUS);

       if(STATUS & (ISR_TXE | ISR_PTX)
         break;

       update--time--in--loop();
      Short_Delay();
      }

// If the routine timed out the tx failed.
      if(time_in_loop > 1 second)
            return(NO_CABLE);

// Read the Transmit Status Register.
      TSR_value = READ(TSR);

// Check if there were excessive collisions.
      if(TSR_value & ABT)
            return(UNTERMINATED);

// If there was 1 to 15 collisions the cable
// is good.
      if(TSR_value & COL)
            return(CABLE_OK);

// Check for other transmission failures
// only after collision check because if a
// collision occurred it can set some of
// the following bits in error.
      if(TSR_value & (CDH I CRS I FU));
            return(NO_CABLE);

// If this point is reached the tx passed.
      return(CABLE_OK);
}
```

More detail on the Interrupt Status Register (ISR) and the Transmit Status Register (TSR) bits can be found in the AT/LANTIC data sheet under Section 5.3.

3.5 Checking the Interrupt

When an interrupt is detected an interrupt handler is called. This handler should then investigate the cause of the interrupt and act accordingly. In the case of the AT/LANTIC there is an Interrupt Status Register which provides information on the cause of the interrupt. It is necessary for the handler to be installed before this test is carried out.

The following pseudo code routine checks interrupt operation by assuming that when the interrupt handler is called and the ISR is found to have its PTX bit set then a ''packet__transmitted'' flag is incremented/set.

```
// Assume Initialized and ISR installed.
Interrupt_check()
{
// Clear the 'packet_transmitted' flag.
      packet_transmitted = 0;

// Clear the Interrupt Status Register.
      WRITE(INTSTATUS, FFH);

// Set the transmit byte count zero.
      WRITE(TBCR0, 0);
      WRITE(TBCR1, 0);

// Issue the transmit command.
      WRITE(CMND, (RD2 | STA | TXP));

// This transmission is used to generate an
// interrupt.

// Loop until ISR is called and
// packet_transmitted flag is set
// or there is a time out.
      while(time_in_loop < 1 second)
      {
      if(packet_transmitted)
            return(Interrupt_OK);

      update_time_in_loop();
      }
      return(Interrupt_FAlLED);
}
```

3.6 Checking the Boot ROM

If a boot ROM has been identified as belonging to the adapter under going diagnostics then it is possible to check that its data has not been corrupted. The sum of all of the ROM bytes should equal 0. The size of the ROM (in ½ k segments) is held at byte offset 03H, offsets 00H and 01H should hold values 55H and AAH respectively.

4.0 TRANSFERRING INFORMATION

Before transmission of a packet or processing of a reception, it is necessary to store or retrieve data from the buffer RAM. The two architecture modes available on the AT/LANTIC have different methods of accessing the buffer RAM.

4.1 I/O Mode Transfers

When in I/O mode the NIC core of the AT/LANTIC transfers data to or from the buffer RAM using one of its DMA channels. The software programs the start address of the RAM segment to be transferred, the size of transfer and the direction of the transfer. The DMA controller passes data between the data transfer port and the buffer RAM, byte or word at a time. The system always writes or reads data via this port. The Data Transfer Port is mapped from I/O base offset 10H to 17H (any of these registers act as the Data Transfer Port). It is important that no other value be written to the command register during a DMA.

BUS ERROR CONDITION

On some implementations of the ISA bus it is possible for an error condition to arise during a DMA transfer. This is flagged by a ''BE'' (Bus Error) bit in configuration register B being set (writing a one to this bit resets it). The AT/LANTIC provides two differing methods for correcting this condition, these are implemented by setting either the ''IO16CON'' bit or the ''CHRDY'' bit in configuration register B. For a more detailed discussion on the cause of the error and the remedies available reference should be made to the AT/LANTIC data sheet Section 6.7 (''16-Bit I/O Cycles with CHRDY Fix'').

It is recommended that a check for this error condition be carried out before an adapter is enabled on a network. This can be done by performing a DMA, checking if the ''BE'' bit has been set and if so implementing one of the fixes. It is also possible to perform the check at the end of every DMA (as the error may not happen during all DMA sequences).

4.1.1 DMA WRITE SEQUENCE

The DMA write sequence is typically used to load up a packet to be transmitted in to the transmit buffer, as in the example below.

```
      ...
// Create a transmit packet and hold a
// pointer to its address in the PC RAM.
      packet = set_up_xmt_pkt();

      Disable Interrupts;

// Write out the Buffer RAM address for the
// xmt packet to the Remote DMA addr.
// registers. This is either TPSR1 or TPSR2.
      WRITE(RSAR0, TPSR_LSB);
      WRITE(RSAR1, TPSR_MSB);

// Write out the size of the packet to the
// Remote DMA BYTE count registers.
      length = SIZEOF(packet);
      WRITE(RBCR0, length_LSB);
      WRITE(RBCR1, length_MSB);

// Issue the Remote DMA write command.
      WRITE(CMND, (RD1 | STA));

// Loop until all bytes/words transferred.
      address = packet;
      for(loop through the transmit pkt)
      {
      value = contents of address;
      WRITE(DATA_PORT, value);
      increment address pointer;
      }

// It is necessary to wait until the last
// transfer is flagged as being placed into
// memory. An access to the command
// register before the DMA has completed may
// corrupt the last transfer and lead to
// serious system errors.
      while(time_in_loop < 1 second)
      {
      status = READ(INTSTATUS);

      if(status & ISR_RDC)
            return(TRANSFER_OK);

      update_time_in_loop();

      Short_Delay();
      }

Enable Interrupts;

return(TRANSFER_FAILED);
...
```

4.1.2 DMA READ SEQUENCE

The DMA read sequence is typically used for removing packets from the receive buffer ring. However the example below reads the Ethernet address from Buffer RAM (as would be required during initialization).

```
      ...
// Read the Ethernet address and place it at
// PC RAM space pointed to by 'addr'.

      Disable Interrupts;

// Set the Remote read start address to 0.
      WRITE(RSAR0, 0);
      WRITE(RSAR1, 0);
```

```
// Set the Remote DMA BYTE count to 6.
      WRITE(RBCR0, 6);
      WRITE(RBCR1, 0);

// Issue the Remote DMA read command.
      WRITE(CMND, (RD0 | STA));

// Read 3 words or 6 bytes depending on bus
// width.
      for(loop 3 or 6 times)
      {
      contents of addr = READ(DATA_PORT);
      increment address;
      }

      Enable Interrupts;

      return(TRANSFER_COMPLETED);
      ...
```

The above examples often refer to the Command and Interrupt Status registers and there associated bits. More details on these registers can be found in the AT/LANTIC data sheet Section 5.3.

The size of access to the data port is dependent on the width of the bus detected during the initialization sequence. However, it should be noted that in 16-bit mode the length of transfer is programmed in bytes even if the transfers to/from the data port are to be word wide.

There are some assembly language commands that greatly simplify the transfer loop to the data port.

e.g., for the Intel 286 Processor.

```
set cx to byte count;
set es:di to pc RAM destination;
set dx to DATA_PORT;
set direction flag;
rep   insw/outsw
```

4.2 Shared Memory Transfers

When in Shared Memory mode the buffer RAM is mapped into a portion of the PC RAM space. Access to the buffer uses the same method as access to any portion of the PC address space. The location at which the buffer RAM appears is controlled by two Shared Memory Control Registers. These registers also set the size and width of transfer as well as enabling or disabling the RAM buffer. The buffer RAM can be set to 16 kbytes or 8 kbytes in size (as determined by the adapter hardware) and the CPU transfer width can be 8-bit or 16-bit wide (as determined in Section 3.2). These settings are controlled by two register bits, the MEMW bit of control register 2 controls the width of memory i.e., 8 kbytes/16 kbytes and the 8-bit/16-bit of control reg. 2 controls whether the transfer is 8-bit or 16-bit wide. If the transfers are to be 16-bit wide then the 8-bit/16-bit should be set only for the duration of the transfer. It is also possible to disable the buffer RAM when there is no transfer in progress by using the MEME bit of control register 1. This allows more than one adapter to utilize the same RAM location in the PC memory map. It should be noted that if multiple adapters are to use the same RAM location then all the adapters must disable the RAM when transfers are not in progress. If only one adapter is allowed to use the RAM location then the memory can be enabled at initialization.

If the software being developed has to be fully compatible with the Ethercard Plus16 architecture then the control registers cannot be assumed to be readable. Thus when toggling the MEME and 8-bit/16-bit during a transfer it is necessary to either re-calculate the value of the address bits in the register or recall some store of the initialized value.

EXAMPLE TRANSFER

This example is for 16 bit wide transfers with memory only enabled for the duration of the transfer.

```
// 'Cntrl1' and 'Cntrl2' are stored at
// initialization.

// Enable the buffer memory.
      WRITE(CNTRL1, (cntrl1 | MEME));
// Enable 16-bit wide transfers.
      WRITE(CNTRL2, (cntrl2 | 8/16);

// Memory transfer.
      MEMCOPY(PCaddr, BufferAddr, size);

// Disable the buffer memory.
      WRITE(CNTRL1, cntrl1);
// Disable 16-bit wide transfers.
      WRITE(CNTRL2, cntrl2);
```

There are some assembly language commands that provide a simple and efficient means of doing the ''memcopy()'' function in the above example.

e.g., for the Intel 286 Processor.

```
set cx to byte count;
set es:di to pc RAM destination;
set ds:si to pc RAM source;
set direction flag;
rep    movsw
```

4.3 The Boot ROM

The AT/LANTIC provides support for both standard boot ROMs and FLASH PROMS. Configuration register C controls the location at which the ROM is enabled, access to the ROM is then the same as access to any area of PC RAM.

The use of a FLASH prom allows software to directly update a boot ROM with the latest driver software for an adapter, thus eliminating the need to replace a ROM for each new release of software. It is necessary to set the BPWR bit of configuration register B when a FLASH prom is to be updated. Write cycles to the ROM area are only allowed when this bit is set. After this bit is set the software should follow the algorithm to program the FLASH prom (as given in the PROM's data sheet).

5.0 TRANSMISSION SEQUENCE

When a transmission is required, it is necessary to select a free transmit buffer to place the packet information into (assuming there is more than one transmit buffer in the buffer RAM). The packet should then be transferred, as described in Section 4.0, to this location. A check is then required to see if a packet is presently being sent out onto the network. If a transmission is in progress then the address and size of

this new packet should be held in a ''store'' until the transmission completes. If a transmission is in progress then this packet can be sent directly out on to the network. This requires the Transmit Page Start Register and the Transmit Byte Count Registers to be programmed with the address and length of the packet respectively. Following this the transmit command can be issued to the Command Register. It is important to remember that the TPSR cannot be altered during a transmission.

ISSUING A TRANSMIT

```
...
// No transmission in progress. The new
// packet is held at transmit buffer
// address XMT_BUFFERx
      WRITE(TPSR XMT_BUFFERx_MSB);
// Set the byte size of the packet, held in
// 'length'.
      WRITE(RBCR0, length_LSB);
      WRITE(RBCR1, length_MSB);

// Issue the transmit command.
      WRITE(CMND, (RD2 | STA | TXP));
```

5.1 Transmission Status Checking

Once a transmission has completed the AT/LANTIC will return an interrupt. It is the task of the Interrupt Service Routine (ISR) to determine whether the transmission was successful or not and deal with it accordingly. If the transmission completed successfully, a check should be made to see if more transmissions are required, the details of which may be held in the ''store'' described in the above paragraph.

When called the ISR should check the Interrupt Status Register. If the PTX bit is set then this indicates that the packet was transferred without error and the transmission can be flagged as successful. If the TXE bit is set, then an error has occurred and more information is required. This extra information is provided by the Transmit Status Register (TSR) from which the relevant error flags and statistics counters can be updated before the transmission can be flagged as completed with error. If collisions occurred during transmission then the COL bit of the TSR is set and the number of collisions that occurred is held in the Number of Collisions Register (NCR). The software can use this register to keep statistics on the amount of collisions occurring on a network.

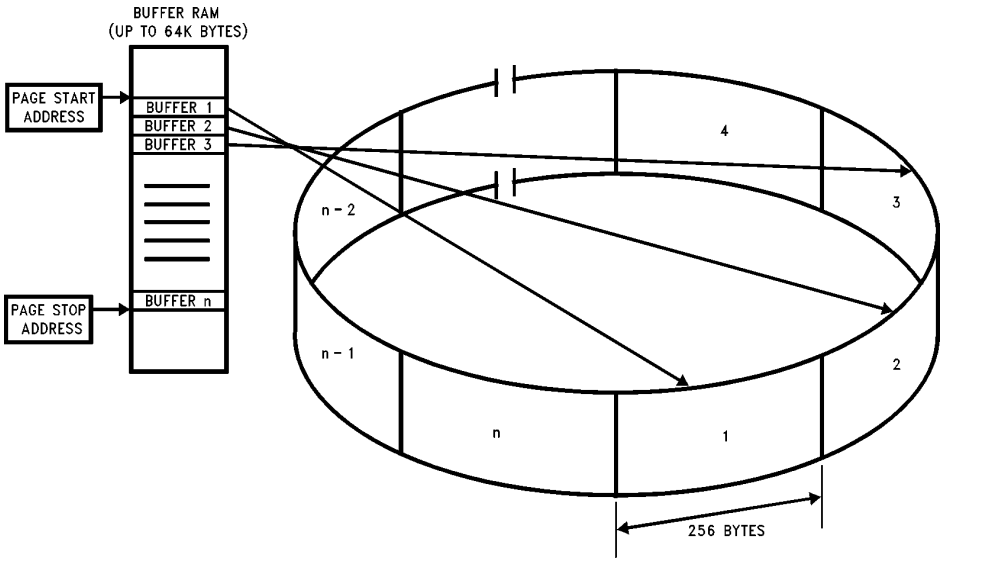
6.0 RECEPTION SEQUENCE

6.1 The Buffer Ring

As packets are received they are placed in to the buffer ring and as they are processed they are removed from the ring. At initialization an area of memory is allocated to act as the receive buffer ring and the AT/LANTIC's buffer management scheme controls the operation of the memory.

Four pointers are used to control the ring; the page start (PSTART) and page stop (PSTOP) pointers to determine the size of the buffer ring, the current page (CURR) pointer to determine where the next packet will be written to from the network and the boundary (BNDRY) pointer to show where the next packet to be read and processed lies. As packets are received, the boundary pointer follows the current page pointer around the ring. During operation the page start and page stop registers remain unchanged.

The receive buffer ring is divided into 256 byte buffers (called ''pages'') and these are linked together as required by the received packets. Since all AT/LANTIC registers are byte wide the ring pointers refer to the page (256 byte) boundaries, see *Figure 10*.



TL/F/11820–3

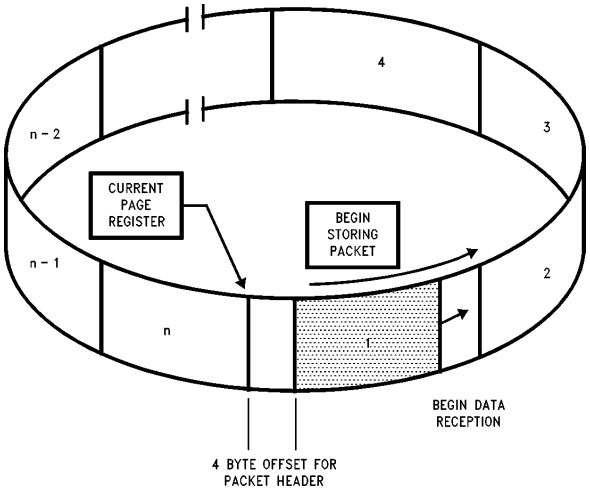
FIGURE 10. The Receive Buffer Ring

On a valid reception the packet is placed in the ring at the page pointed to by CURR plus a 4 byte offset (see *Figure 11*) . The packet is transferred to the ring through the AT/LANTIC which links the page buffers as necessary until the complete packet is received. The first and last buffers (PSTART and PSTOP) are linked just as the first and second buffers would be. At the end of a reception, the status from the Receive Status Register (RSR), a pointer to the next packet and the byte count of the current packet are written into the 4 byte offset (see *Figure 12*).

6.2 Removing a Packet

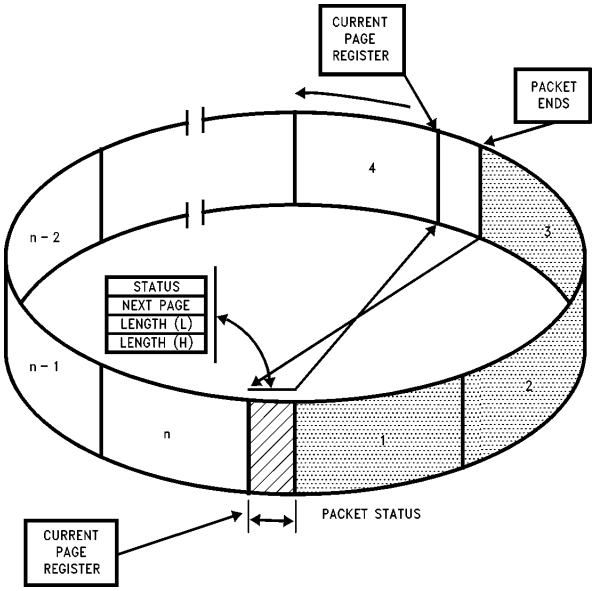
Once packets are in the receive buffer ring they must be processed. The AT/LANTIC supports two differing adapter architectures which have there own method of accessing the buffer memory, these are discussed in Section 4.0 of this document. As packets are removed from the buffer ring, the boundary (BNDRY) pointer must be updated. The BNDRY follows CURR around the ring (see *Figure 13*).

If the current local DMA address (the place where a newly received packet is being stored) ever reaches BNDRY then the ring is full and any more receptions cannot be achieved until some processing has been done on the ring. This condition is known as overflow. More details on this condition and how to overcome it are given in Section 6.3.



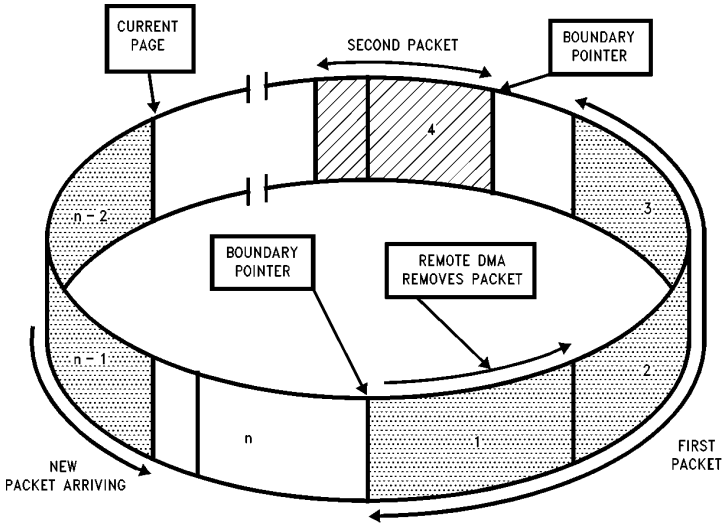
TL/F/11820–4

FIGURE 11. Receiving a Packet



TL/F/11820–5

FIGURE 12. Receive Packet Header



TL/F/11820–6

FIGURE 13. Removing a Packet

When the CURR and BNDRY pointers are equal, the buffer ring can either be full or empty. To ensure that the software never misinterprets this condition the BNDRY pointer can be kept one less than the CURR pointer when the ring is empty, and only equal to CURR when the ring is full, as shown below.

1. Use a variable (Next__pkt) to indicate where the next packet will be removed from the buffer ring.
2. At initialization set (see Section 3.3):

```
BNDRY = PSTART
CURR = PSTART + 1
Next_pkt = PSTART + 1
```

3. After each packet is removed from the ring, use the next packet pointer in the header information (the second byte of the header), HNXTPKT and set:

```
Next_pkt = HNXTPKT
BNDRY = HNXTPKT
if (BNDRY < PSTART)
BNDRY = PSTOP – 1
```

THE SEND PACKET COMMAND

When in I/O mode the Remote DMA channel can be automatically initialized to transfer a single packet from the receive buffer ring. The transfer is initiated by issuing a ''send packet'' command, setting bits RD1, RD0 and STA in the command register. The DMA will be initialized to the value of the BNDRY pointer and the Remote Byte Count registers will be initialized to the values found in the buffer header of each received packet. After the data has been transferred, the BNDRY pointer is advanced and the Remote DMA is then prepared to receive the next packet.

This method does not require the manual updating of registers as discussed in Steps 1–3, however it does limit the software to accessing a packet just once.

The receive sequence is initiated by an interrupt generated by the AT/LANTIC when data is ready to be removed from the ring or an overflow has occurred. The Interrupt Service Routine should then interrogate the Interrupt Status Register which flags a PRX bit when a packet is ready to be removed or OVW if the buffer ring is full. When this PRX bit is set then a receive subroutine following the above sequence should be called, if the OVW bit is set the overflow routine discussed in Section 6.3 should be called.

The receive buffer ''ring'' is a linear section of RAM forced into a ring by linking the end (PSTOP) and beginning (PSTART) of the buffer. Some of the received packets shall overlap or ''wraparound'' this link in the buffer, i.e. the start of the packet is held up to PSTOP and the rest is held from PSTART. This condition is automatically dealt with when in I/O mode however Shared Memory mode requires more care. In Shared Memory mode a ''wraparound'' packet has to be transferred in its two sections, i.e., the section up to PSTOP and the section from PSTART.

A pseudo code example of the sequence is given below.

RECEIVE ROUTINE

```
Receive_subroutine()
{
// This routine loops until all the packets
// currently held in the buffer ring are
// removed.
      while(Next_pkt != CURR)
      {
```

```
// Get the 4 byte header from the packet
// pointed to by Next_pkt, the method of
// removing the data depends on the mode of
// the AT/LANTIC.
      status = read_status();
// If in shared memory mode then check if
// the packet wraps around the PSTOP
// pointer.
      if(in_shared_memory_mode)
      {
      if (status.length + Next_pkt > PSTOP)
         {
         transfer_up_to_PSTOP0;
         transfer_from_PSTART();
         }
        else
         transfer_all_at_once();
      }
      else
// I/O mode automatically carries out
// wraparound.
      transfer_all_once();
// Set the Next_pkt pointer to equal the
// next packet pointer in the status bytes.
      Next_pkt = status. next_packet;
// Update the boundry pointer.
      BNDRY = Next_pkt - 1;
      if(BNDRY < PSTART)
         BNDRY = PSTOP - 1;

      WRITE(BNDRY);
      } // end of while loop.
}
```

6.3 Dealing with Overflows

In heavily loaded networks it is possible for the receive buffer ring to be filled with packets that still require processing, i.e., the CURR pointer reaches the BNDRY pointer. If this situation occurs then the AT/LANTIC suspends further receptions and posts an overflow (OVW) interrupt.

In the event of an overflow condition occurring it is necessary to follow the routine given below. If this routine is not adhered to then the AT/LANTIC may act in an unpredictable manner. A flow chart of the routine is given in *Figure 8.*

Note: It is necessary to define a variable in the driver, which will be called ''Resend''.

1. Read and store the value of the TXP bit in the AT/LANTIC Controller's Command Register.
2. Issue the STOP command to the AT/LANTIC Controller. This is accomplished by setting the STP and RD2 bits in the AT/LANTIC Controller's Command Register.
3. Wait for at least 1.6 ms. Since the AT/LANTIC Controller will complete any transmission or reception that is in progress, it is necessary to time out for the maximum possible duration of an Ethernet transmission or reception. By waiting 1.6 ms this is achieved with some guard band added. Previously, it was recommend that the RST bit of the Interrupt Status Register be polled to insure that the pending transmission or reception is completed. This bit is not a reliable indicator and subsequently should be ignored.
4. Clear the AT/LANTIC Controller's Remote Byte Count registers (RBCR0 and RBCR1).
5. Read the stored value of the TXP bit from step 1, above.

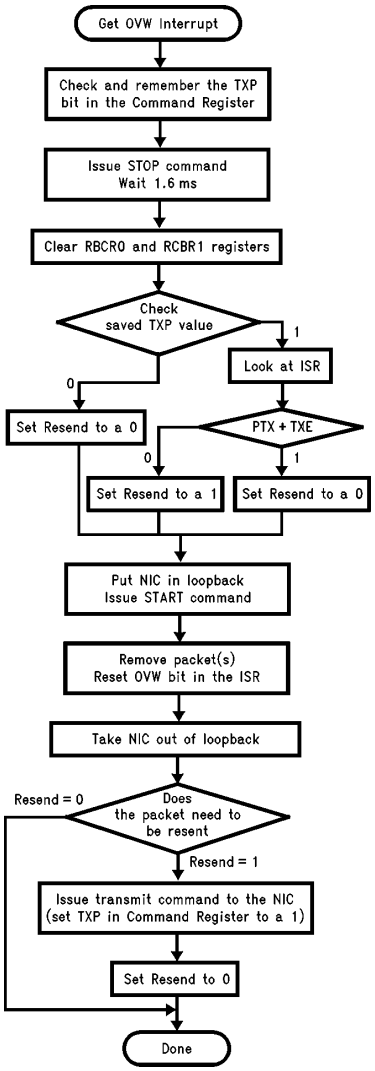
   If this value is a 0, set the ''Resend'' variable to a 0 and jump to step 6.

   If this value is a 1, read the AT/LANTIC Controller's Interrupt Status Register. If either the Packet Transmitted bit (PTX) or Transmit Error bit (TXE) is set to a 1, set the ''Resend'' variable to a 0 and jump to step 6. If neither of these bits is set, place a 1 in the ''Resend'' variable and jump to step 6.

   This step determines if there was a transmission in progress when the stop command was issued in step 2. If there was a transmission in progress, the AT/LANTIC Controller's ISR is read to determine whether or not the packet was transmitted by the AT/LANTIC Controller. If neither the PTX nor TXE bit was set, then the packet will essentially be lost. If a packet was ''lost'' then a transmit command can be reissued to the AT/LANTIC Controller once the overflow routine is completed (as in step 11). Also, it is possible for the AT/LANTIC Controller to deter indefinitely, when it is stopped on a busy network. Step 5 also alleviates this problem. Step 5 is essential and should not be omitted from the overflow routine, in order for the AT/LANTIC Controller to operate correctly.
6. Place the AT/LANTIC Controller in either mode 1 or mode 2 loopback. This can be accomplished by setting bits D2 and D1, of the Transmit Configuration Register, to ''0,1'' or ''1,0'', respectively. Further explanation of loopback can be found in the AT/LANTIC data sheet under Section 6.5.
7. Issue the START command to the AT/LANTIC Controller. This can be accomplished by setting the START and RD2 bits in the Command Register. This is necessary to activate the AT/LANTIC Controller's Remote DMA channel.
8. Remove one or more packets from the receive buffer ring.
9. Reset the overwrite warning (OVW, overflow) bit in the Interrupt Status Register.
10. Take the AT/LANTIC Controller out of loopback. This is done by writing the Transmit Configuration Register with the value it contains during normal operation. (Bits D2 and D1 should both be programmed to 0.)
11. If the ''Resend'' variable is set to a 1, reset the ''Resend'' variable and reissue the transmit command. This is done by the TXP bit in the Command Register. If the ''Resend'' variable is 0, nothing needs to be done.

Note: If Remote DMA is not being used, the AT/LANTIC Controller does not need to be started before packets can be removed from the receive buffer ring. Hence, step 8 could be done before step 7, eliminating or reducing the time spent polling in step 5.

Note: When the AT/LANTIC Controller is in STOP mode, the Missed Packet Tally counter is disabled.



TL/F/11820–7

FIGURE 14. Overflow Routine

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998