Magic Packet Technology Application in Hardware and Software

AMD

Application Note

ABSTRACT

This application note addresses the use of the Magic Packet technology in conjunction with green PC hardware and system-level software. Its objective is to assist individuals in using this new technology in their own environments.

INTRODUCTION

Over the last few years, two technologies have evolved that seem to be competing rather than complementing each other at the PC desktop. These technologies are network technology and green PC technology. The connectivity rate in large companies using network technology has been increasing exponentially. This trend has resulted in an on-going need for network management and for network managers to have access to PCs on their networks. At the same time, the concept of green PCs has gained momentum because the Environmental Protection Agency (EPA) began encouraging the use of low-power computers to conserve energy.

Network administrators and Information Systems (IS) personnel in large companies are required to perform tasks, such as backups or installation of software upgrades, at remote sites at night when system downtime does not impact the users. However, these tasks performed after hours require the users to leave their machines on overnight, resulting in twice as much electricity consumed as daytime usage (16 hrs. vs. 8 hrs.). Green PCs, however, are not network friendly. That is, if the machine is asleep (in a low-powered state), it cannot be addressed from the network, thus hindering the network administrators from doing their nightly tasks.

Working together, AMD and Hewlett Packard (HP) came up with a solution to this problem. Since power is generally on inside a green machine, the network side of the machine could be left in a state, whereby it would continue to scan every packet coming in from the network, this time looking for the special data sequence that would serve to *wake up* the sleeping machine. The value that separates one machine from another on an Ethernet network is its unique IEEE address. This unique address repeated 16 times in a row anywhere within a valid network frame's data field was chosen to serve as the wake-up call. This frame has now been called a *Magic Packet* frame.

In the interest of making Magic Packet technology an industry standard, AMD is licensing this technology to network vendors at minimal cost, royalty free, so that all vendors can enjoy its benefits. Although Magic Packet technology is not limited to any particular type of Ethernet connection, the only connection that makes sense from a low-power standpoint is a 10BASE-T connection.

The green computer hardware implementation and the software interface now stand in opposition to the final solution. This application note is meant to pull the solution together in a concise format providing a way for PC vendors and embedded system users to implement AMD's Magic Packet technology. Divided into two sections, this application note addresses (1) the hardware interface for three different hardware models and (2) how the DOS-level Magic Packet software driver interfaces both with the operating system and the hardware to bring a complete solution to the marketplace.

Currently, AMD has designed *Magic Packet* technology into two of its network controllers, the PCnet™-ISA II (79C961A) and the PCnet™-PCI II (79C970A). Both of these controllers are available and in production today. For further information, refer to the following AMD data sheets: *Am79C961A Data Sheet (PID 19364)* and *Am79C970A Data Sheet (PID 19436).*

HARDWARE

In the competitive PC environment, *green* has become the byword for the next new standard system feature. The original targets by the Department of Energy (DOE) in its green recommendations were 30 watts (W) for the system box and 30 W for the monitor in a low power state. Since these values were relatively easy to obtain in current technology, the market shifted to asking the question, "how deep a shade of green is your system?" There now stands at least three different solutions to obtaining green status in the market. The following solutions are ordered from most power used to least power used:

Publication# **20381** Rev: **A** Amendment/**0**
Issue Date: **March 1996**

1. Simply lower the clock speed on the motherboard to a slow speed, usually 8 MHz. Shut down all disks and stop all synchs at the video controller.
2. Stop the processor clock, putting the DRAM in slow refresh, in addition to solution 1.
3. Suspend to disk all DRAM and control registers, and then power off the entire system, leaving a small auxiliary power supply to *some* circuits to allow them to awaken the sleeping machine.

In examining each option below, an increasing level of complexity will be found in the hardware solutions to implement the Magic Packet technology. In time, some system vendors may use each of these levels in the orderly shutdown of the box to meet the needs of the green PC market.

Hardware Model 1

Hardware Model 1 requires no hardware, software, or even Magic Packet technology for implementation, because nothing in the system box, except the disks, is shut down. All network activity is fully alive and the speed of the CPU can be varied with the interrupt rate being handled. The CPU is not stopped and the network controller is not required to do anything differently.

Hardware Model 2

Hardware Model 2 requires some level of interface. Since clocks are stopped in this implementation, something must be used to return the system to normal operating mode. Without the network card installed in the system box, either a keystroke or mouse movement could repower the system to full power. In reality, either of these actions could cause an interrupt on the backplane. This model contains a circuit added by chipset vendors to monitor specifically designated interrupts for activity and then return the clock to the CPU, signaling this event through a technique known as System Management Interrupt (SMI).

Since AMD controllers already use an interrupt line to signal that the driver's service is needed for network-related activities, this same interrupt line can be used to wake the system as well. This waking interrupt can be added to the designated interrupt list in the main system board setup menu. However, with the PCI clock stopped, neither Direct Memory Access (DMA) activity nor Magic Packet interrupt generation by the controller on the PCI bus can take place.

Since normal operation of AMD's network controllers requires DMA activity, they must be placed in Magic Packet Mode. Currently, PCnet-ISA II and PCnet-PCI II support this. In this mode, the AMD controller will not initiate any DMA activity. All incoming data is scanned by the address recognition logic until a Magic Packet is received. When this occurs, MPINT is set in CSR 5 (bit 4) and will be reported through the interrupt pin if MPINTE is set in CSR 5 (bit 3). The use of these two bits will be discussed later in the *Software* section.

The level of compliance required (for both green and network*)* can be done on any current card or motherboard implementation on the open market, using either PCnet-ISA II or PCnet-PCI II as the network controller solution, without need for any hardware addition or modification. However, there is one issue that must be considered. That is, if the PCnet-PCI II part is used and the PCI clock is stopped when in standby mode, a PCI interrupt cannot be generated and an LED pin must be used to perform the interrupt operation.

Hardware Model 3

To make Hardware Model 3 work, several issues need to be solved in the hardware implementation. Current technology for the complete system power down and wake up calls for a modem ring detect input to wake up the sleeping unit. When the modem is called the first time, the ring detector pulses a logic input to the system box. Because the serial port does not have the Data Set Ready (DSR) true, the modem does not answer. However, the system starts its power up. The calling party times out and calls again, and by this time, the system is powered up and ready. This is the model to be emulated in the network adapter card using Magic Packet technology.

Since the network card is plugged into the backplane, which will be powered down, the power for the network card must come from an alternate source. This power is usually supplied by an auxiliary power supply connection, and the normal +5 volt (V) connection to the ISA or PCI bus connector is an open circuit. This auxiliary supply would have to be able to supply an additional 100 milliamperes (mA). A zero ohm (Ω) jumper could be used here to make the card a manufacturing configurable unit for use with or without auxiliary power supplies.

If the PCnet controller was certified for +5/-10% VCC tolerance, then a low resistance P channel Field Effect Transistor (FET) could be utilized to provide a direct no jumper support of the Printed Circuit Board (PCB) between backplane voltage and auxiliary voltage as the system was configured. The gate of the FET would be driven with the auxiliary voltage connector and would isolate the power planes from the backplane if the auxiliary power supply were present. However, this would add to the cost and complexity of the PCB design.

For Federal Communications Commission (FCC) purposes, a set of chokes or preferably a balun should be added with bypass capacitors to prevent false ground loops from radiating noise on the auxiliary power connector.

The next issue that must be addressed is that of reset. Since the power on the backplane will be cycling, a

reset on the first power up and then on the power down will occur, and yet a third reset on the subsequent power up. Since a reset to an AMD controller would take the Magic Packet detection logic off line, at least the second reset has to be blocked, or the Media Access Controller (MAC) will be taken off line and the Magic Packet frame will not be seen.

In order for the network card to be used together with the modem ring detection logic, there must be an external signal that comes from the controller on the card that tells the modem logic to wake up the system. AMD designers foresaw this need and provided that the detection of a Magic Packet frame be indicated on any of the LED pins. This is enabled by setting MPSE (bit 9) in any of the respective LED control registers. However, some of the LED registers do have power-up default values that may be the wrong polarity for the modem logic. Only LED3 has a default value to transmit activity that on power up or reset can be guaranteed to provide a signal to the modem logic with the correct polarity. Therefore, the LED3 pin has to be the external connection for Magic Packet indication. Figures 1 and 2 show the schematic and general timing diagram for a correct LED3 implementation using either PCnet-ISA II or PCnet-PCI II.

Magic Packet mode is enabled by performing three steps. First, the PCnet-ISA II or PCnet-PCI II controller must be put into suspend mode, allowing any current network activity to finish. Next MPMODE (CSR 5, bit1) must be set to ONE if it has not been set already. Finally, either SLEEPL must be asserted (hardware control) or MPEN (CSR 5. bit 2) must be set to ONE (software control). Of the two ways, the software way is preferable because the only way on an adapter card to detect power going down is to monitor the backplane VCC or RESET to set the SLEEP pin. This could cause Magic Packet mode to happen in the middle of a DMA cycle resulting in a DMA problem (as power is going down) that may result in loss of data. The only way to coordinate going into Magic Packet mode with the operating system is by using the software-enabled Magic Packet mode. How the software works will be discussed in the next section.

PAL Equation Discussion

Since AMD's current controllers do not have all the necessary interface circuitry built into them, an external PALCE16V8 is used in this application to provide the necessary interface to the system bus RESET and to provide the proper polarity for the LED3 pin to interface to the modem ring detect. In the case of the system vendor used for demonstration purposes, modem ring detect was positive true, while the LED3 pin is negative true on power up default.

Appendix A contains the PALASM equations for the PALCE16V8 used. This PAL's function is mainly to condition the RESET pin of the PCnet controller and to interface the LED3 pin to the system vendor's power control circuitry. Most of the PAL is used to build an asynchronous state machine used to decide when to block the second reset (the one generated when power is going down). Depending on whether the system simply reloads the memory image and takes off, or the system is restarted from ground up, will determine whether subsequent repower-up RESETs need to be blocked also.

In system operation, the first power supply to reach its final operating voltage level is the auxiliary power supply. This usually happens when the machine is simply plugged into the wall. The auxiliary power supply powers the PALCE16V8 and the PCnet controller. It will be some time later when the power switch is turned on for the main system power, and the interface RESET becomes active for the first time while main power is rising to its final voltage level.

From initial auxiliary power supply turn on to the end of the system RESET, the reset pin of the PCnet controller must be held active to ensure the PCnet controller state. Therefore, the equations for S_R_RESET determine the initial time when auxiliary power has come on for the first time and the main power supply has not come on yet. This is accomplished by using an exclusive OR in the first two terms of the equation using PULLUP and NC2, which generates a 2 Tpd pulse, setting S_R_RESET true. This is then reinforced by the third term in a positive feedback loop. S_R_RESET will then be held true until the first system reset (RESET_IN) becomes true. Once S_R_RESET goes false, it will never come true again until the auxiliary power supply goes off and comes back on again.

The equation RESET_TO_CHIP is used to condition the reset pin on the PCnet controller. The first term S_R_RESET will provide reset until the main power supply comes on. Then RESET_IN will reset S_R_RESET, but because FIRST_TIME is false, it will also provide reset to the PCnet controller. The third term in this equation is only used if the system being powered back up reestablishes the network driver. If the system only reloads the DRAM image and does not execute `config.sys` and `autoexec.bat`, then this line should be commented out in the PALASM compile.

The equation FIRST_TIME is used to sense when the PCnet controller has established its default EEPROM values, by using the EEPROM clock to clock a flip-flop whose D input is the false of S_R_RESET. The AMPALCE16V8 has an internal power-on reset circuit for all flip-flops; therefore, it can be guaranteed that this flip-flop will always come up in the reset condition. This flip-flop is then used to qualify what is done with RESET_IN and the fact that VCC at the backplane interface has gone away (VCC_GONE on pin 3 is simply

tied to one of the backplane VCC pins and provides a sense condition for the backplane power).

The equation SEC_RESET_OK is used to provide a state of indication between the power down and the re-power up due to the presence of a Magic Packet frame. The first and the third terms are used in the event that someone uses the power switch on the front panel, rather than at the arrival of a Magic Packet frame. The second term is used if a Magic Packet arrives and is indicated by LED3. LED3 will go away on a Magic Packet arrival condition after power comes up and the RESET_IN is allowed to go the RESET_TO_CHIP through the third term of RESET_TO_CHIP.

The equation LED3_OUT is used to both invert the polarity of the LED output driver as well as AND the indication of a Magic Packet frame's presence with the state of backplane VCC. The polarity is dictated by the system vendor and the spare output may also be used to generate the negative true output for those systems that would prefer that condition.

Appendix A gives the equations for an implementation using the PCnet-ISA II controller. The PCnet-PCI II controller is similar but has some differences. Because of the architecture of the PCnet-PCI II part, the LED pins are multiplexed with the EEPROM pins. The equations applicable to PCnet-PCI II are given in Appendix B. The timing for the PCI version is shown in Figure 3.

PCnet Issues

After these PAL discussions, the issue of having the PCnet controller's I/O pins driving the backplane bus while power is down on the motherboard must be addressed. The PCnet-ISA II controller only has one actively driven high non-tristated output on the DREQ pin selected. Since the current for this pin is limited, the only issue will be the small amount of extra current drawn on the auxiliary power supply.

The PCnet-PCI II does not have the same issues as the PCnet-ISA. When the MAC is put into Magic Packet mode, all bus interface output pins are placed into High Z. The expansion ROM pins are not tristated so an expansion ROM must also be powered off the auxiliary power supply. However, the PCnet-PCI II does have one other consideration. That is, this controller has two sets of power pins, one for the core and the other for the I/O buffers. This was done to support both 5-V PCI bus interface and 3.3-V PCI interface by connecting the Vio pins of the PCI bus interface to the Vbuffer pins on the PCnet-PCI II controller. These power pins were designed to never have to go below 3.0 V in a working system. However, when a motherboard powers down, these pins will go to 0 V. This can cause internal breakdown on the chip, so the voltage (Vbuffer) pins must be connected to the auxiliary power supply. This implies that this card design will not support a 3.3-V PCI bus.

SOFTWARE

The software issues of interfacing the network drivers and Magic Packet code with the operating system will be discussed in this section. The only demonstrable code working today is in the DOS and Windows 3.1/ Windows for Workgroups 3.11 environment. OS2 and other environments, probably with similar implementation schemes, will not be discussed here.

The power management standard in a DOS system was established by Microsoft and Intel by way of the Advanced Power Management (APM) Specification. In the APM specification, some system interrupts were established to inform the necessary drivers needing the information that the system is preparing for power down or going to resume from a power down state. Unfortunately, by the time the APM specification was created, all the interrupts DOS was using were already taken. Therefore, interrupt 2F was chosen to be the APM interrupt. Interrupt 2F had been the catchall for miscellaneous interrupts, so that any driver associating with this interrupt has to check to see if this interrupt is even available for its use. Therefore, it is this interrupt that will be used in the DOS-based Terminate Stay Resident (TSR) driver. Appendix C shows the DOS TSR developed to do the interface between DOS and the PCnet controller and the network driver.

In the way the operating system and the network drivers work, the APM is unknown to the network stack. This may change in the future, but for now the real interface will be between the operating system and the hardware itself. Once loaded, the network driver operates strictly off interrupts, so that if there are no interrupts it simply is asleep and waiting. The hardware is set up by the driver to generate interrupts only for the receive activity. The transmit activity is initiated by a system call to the disk redirector, which eventually filters down to a call to the transmit routine. Therefore, if the system goes to sleep, there will be no redirection effort, because no programs will be running. If the PCnet hardware is put in the suspend mode before being put into Magic Packet mode, then any receive activity will be used for Magic Packet detection only.

Because of this independence between the Magic Packet TSR driver and the network driver, it was decided to treat them as separate entities. The advantage is that both programs can be independently revised as needed. In addition, the TSR is written so that it can be loaded in any order with respect to the network driver, and both can coexist. If the TSR is loaded without the network driver, then the Magic Packet functionality will not work, because the MAC is never initialized and enabled, and there is no damage done to the operating system or the operation of the base system.

(See the *Flowcharts* section for a step-by-step illustration of the Magic Packet TSR driver software.)

If the Magic Packet TSR did want to initialize the PCnet part because a 2F interrupt was received, the stop bit could be checked in the TSR. If the controller was found stopped, the following could be a minimum procedure to place the MAC on line:

1. Write the IEEE address of the node directly to the PADR.
2. Write the MODE register (CSR 15) with DRX and DTX set to 1.
3. Set CSR 4 and CSR 5 as they should be for Magic Packet mode.
4. Set START bit in CSR 0 to 1.

The TSR program listing in Appendix C is basically divided into two parts. The first part of the code is located in the last part of the listing. This is the initialization part of the code. After the code runs for the first time, this code drops off and the second part of the code, the TSR itself, stays behind to do the interface job.

The initialization is responsible for two tasks. First, the controller must be searched out on the backplane. Since the PCnet-ISA II and the PCnet-PCI II are on two different busses, the method of employing the search is different. The ISA controller is found by looking for the characteristic '57' (ASCII *w*) at location XXF in the register space. This search is started at the base address of 200h and incremented by 20h until found. If there is no ISA controller found, then the search is made on the PCI bus.

The search on the PCI bus is accomplished by using a call to the PCI BIOS and asking the PCI BIOS to find the vendor ID of 1022h. If no controller is found, the software driver returns to DOS outputting a message to that fact and does not leave the TSR stub. If a controller is found, the controller is examined to ensure that it supports Magic Packet mode. If the controller found is not Magic Packet capable, then the software driver returns to DOS outputting that fact and does not leave the TSR stub. If the controller found is Magic Packet capable, then the address of the controller is stored for later use by the TSR itself. This value is saved within a data area of the TSR itself. Before hooking the *2F* vector, the TSR programs Magic Packet mode in CSR 5 and sets DPOLL in CSR 4 (bit 12) to disable polling. After saving the address and the initialization routine, the TSR then prepares for the hooking of the 2F vector. This is done by first getting the current CS:IP of the first routine in the stack in the vector table and saving it in a data structure of the TSR. Then the address of the TSR (CS:IP) is put in place of that vector entry in the table and a return is done to DOS, but this time the software driver leaves the TSR part of the code active and waiting.

The TSR is a simple procedure in that it is now hooked at the top of the 2F chain. Once a 2F interrupt is requested, the software driver will be called and first interrogates the call by looking at the AX register for a value of 530Bh, which is for an APM function. If it is not 530Bh, then the software driver simply restores the flag register and calls the routine below. If it is a 530Bh, then the software driver looks at BL for a value of 2 or 3. If BL is a 2, then the software driver is requested to prepare for a *suspend* (not the AMD controller suspend), and if it is a 3, the software driver is being requested to do a *power up resume*. If it is any other value, the software driver simply restores the flag register and calls the routine below.

For a *suspend*, the routine does two things. It uses the stored controller address that was found in the initialization of the TSR to prepare the LED3 control register to output the indication of a Magic Packet frame found on this pin. This is done by setting MPSE (bit 9) in the BCR7 register to support Hardware Model 3. If the interrupt method is used, then this step could be skipped. A consideration might also be to disable all other LED pin drivers by setting bit LEDDIS (bit 13) to 0 (false) to conserve power on the auxiliary power supply. Then CSR 5 is programmed to enable Magic Packet mode. It is during the write to CSR 5 that the Magic Packet interrupt method can be controlled by setting the appropriate enable bit. This corresponds to Hardware Model 2. After that, the AX, DX, and flag registers are fixed and control is passed to the next TSR in the 2F chain.

If a BL of 3 is found, the software driver will do a *power up resume*. Here the program simply does a write of the CSR 5 register to clear the Magic Packet enable and suspend bits, and then fixes the AX, DX, and flag registers and passes control to the next routine. If the interrupt method were employed, then the interrupt bit would have to be reset as necessary.

CONCLUSIONS

The Magic Packet technology enables better working networks within the corporate environment and can easily be added to a *green* enabled computer at little or no cost above the standard system cost, as indicated by this application note. This represents a strong value proposition for the Magic Packet technology to become the standard in the market place, allowing early implementors to be on the leading edge of a new standard with a small investment.

FIGURES

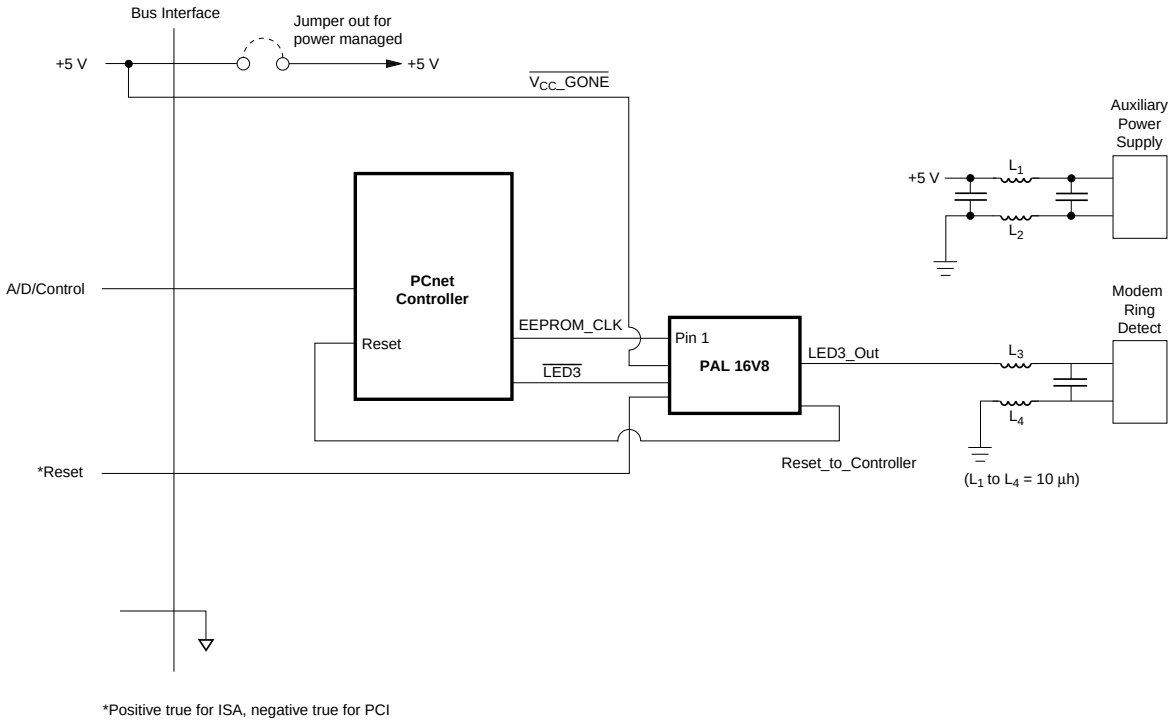


Figure 1. PAL and PCnet Controller Schematic

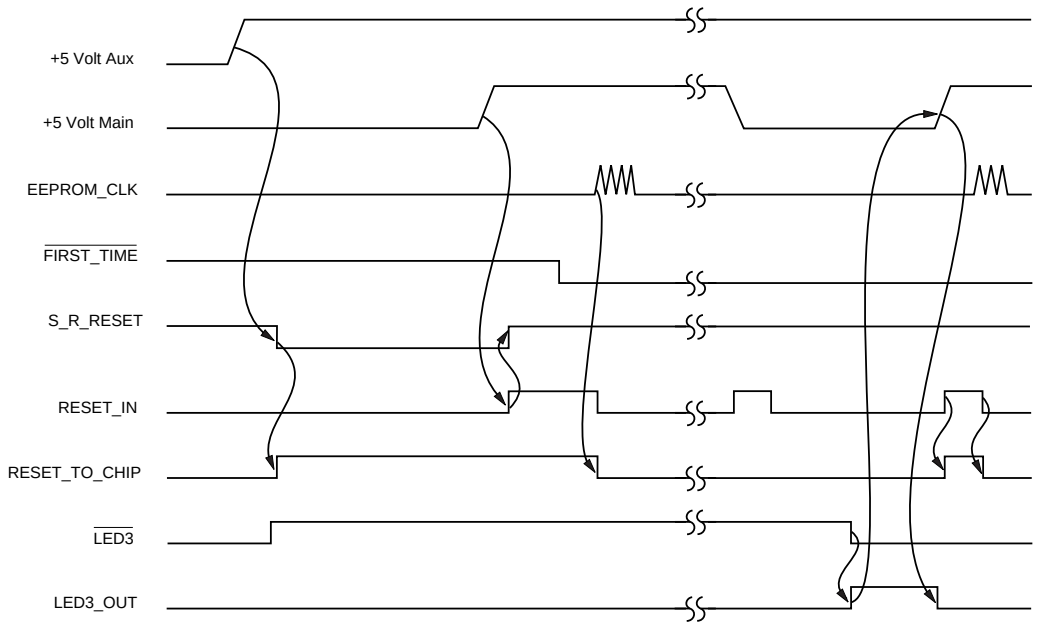


Figure 2. ISA Version Timing Diagram

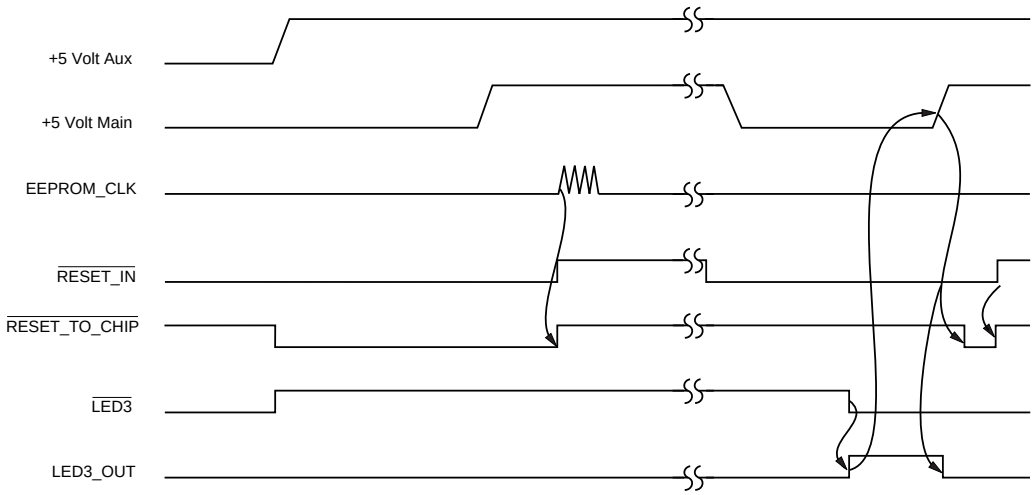
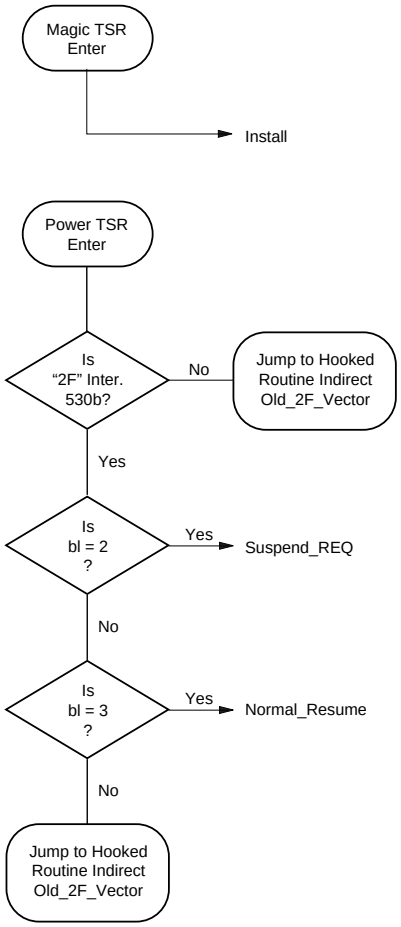
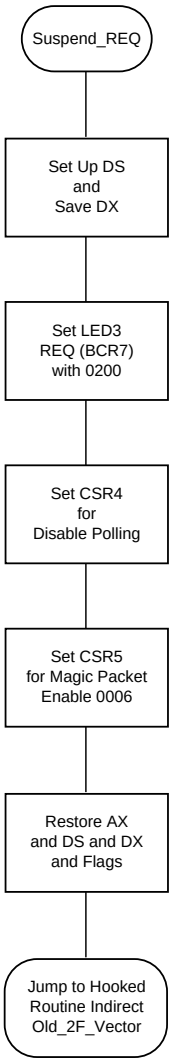
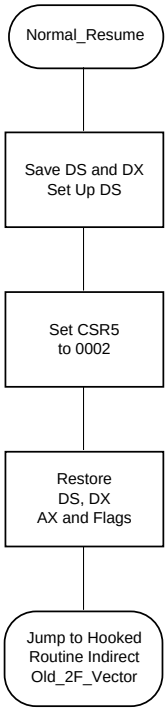


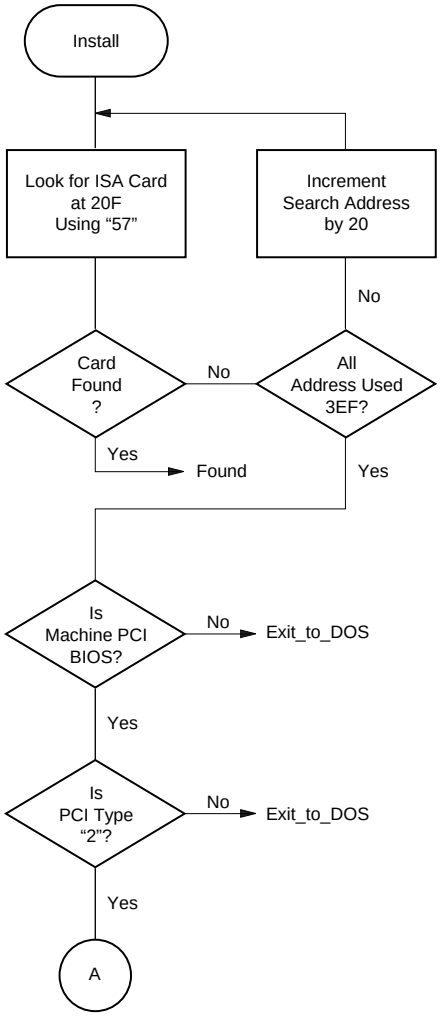
Figure 3. PCI Timing Diagram

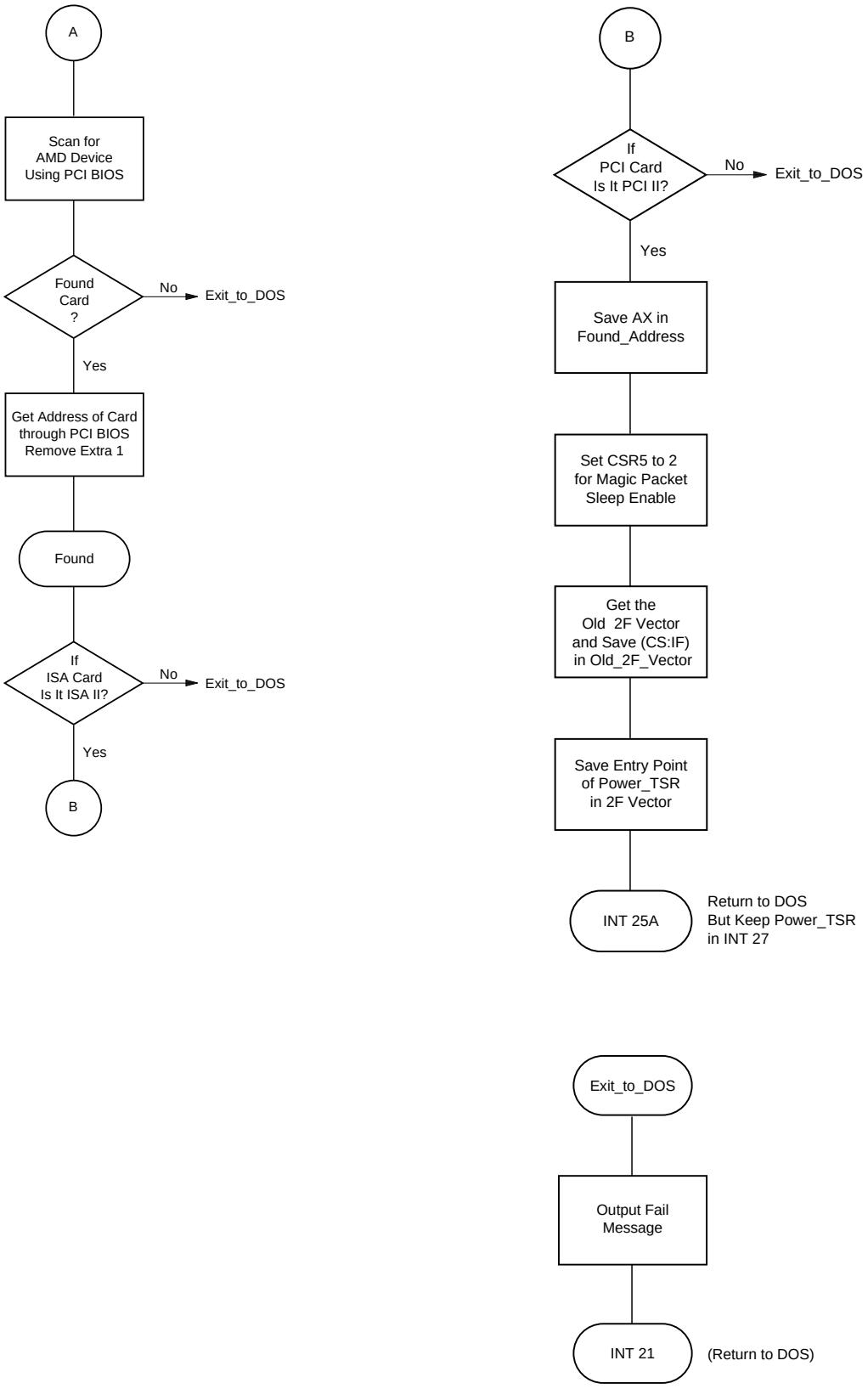
FLOWCHARTS


Magic TSR Enter
Install
Power TSR Enter
Is "2F" Inter. 530b?
No
Jump to Hooked Routine Indirect Old_2F_Vector
Yes
Is bl = 2 ?
Yes
Suspend_REQ
No
Is bl = 3 ?
Yes
Normal_Resume
No
Jump to Hooked Routine Indirect Old_2F_Vector


Suspend_REQ
Set Up DS and Save DX
Set LED3 REQ (BCR7) with 0200
Set CSR4 for Disable Polling
Set CSR5 for Magic Packet Enable 0006
Restore AX and DS and DX and Flags
Jump to Hooked Routine Indirect Old_2F_Vector


Normal_Resume
Save DS and DX
Set Up DS
Set CSR5
to 0002
Restore
DS, DX
AX and Flags
Jump to Hooked
Routine Indirect
Old_2F_Vector


Install
Look for ISA Card
at 20F
Using “57”
Increment
Search Address
by 20
No
Card
Found
?
No
All
Address Used
3EF?
Yes
Found
Yes
Is
Machine PCI
BIOS?
No
Exit_to_DOS
Yes
Is
PCI Type
“2”?
No
Exit_to_DOS
Yes
A


A
Scan for
AMD Device
Using PCI BIOS
Found
Card
?
No
Exit_to_DOS
Yes
Get Address of Card
through PCI BIOS
Remove Extra 1
Found
If
ISA Card
Is It ISA II?
No
Exit_to_DOS
Yes
B
B
If
PCI Card
Is It PCI II?
No
Exit_to_DOS
Yes
Save AX in
Found_Address
Set CSR5 to 2
for Magic Packet
Sleep Enable
Get the
Old 2F Vector
and Save (CS:IF)
in Old_2F_Vector
Save Entry Point
of Power_TSR
in 2F Vector
INT 25A
Return to DOS
But Keep Power_TSR
in INT 27
Exit_to_DOS
Output Fail
Message
INT 21
(Return to DOS)

PAL Equations

```
;PALASM Design Description
;---------Declaration Segment-----------
TITLE    Magic Packet Converter
PATTERN mag_pkt.pds
REVISION A
AUTHOR   David Stoenner
COMPANY AMD
DATE     02/02/95

CHIP _mag_pkt PALCE16V8
---------------PIN Declarations---------
PIN 1     EEPROM_CLK
PIN 2     RESET_IN
PIN 3     /VCC_GONE
PIN 4     /LED3
PIN 5     /PULLUP
PIN 10    GND
PIN 11     /OE
PIN 12    RESET_TO_CHIP
PIN 13    /FIRST_TIME
PIN 14    SPARE
PIN 15    LED3_OUT
PIN 16    SEC_RESET_OK
PIN 17    /S_R_RESET
PIN 18    /NC1
PIN 19    /NC2
PIN 20     VCC

;-------- Boolean Equation Segment -----
-
EQUATIONS

LED3_OUT = LED3 * VCC_GONE

NC1 = PULLUP

NC2 = NC1



S_R_RESET =     PULLUP * /NC2 * /RESET_IN
                     + /PULLUP * NC2 * /
RESET_IN
                  + S_R_RESET * /RESET_IN

SEC_RESET_OK =     FIRST_TIME * VCC_GONE
                   + SEC_RESET_OK * LED3
                        + SEC_RESET_OK *
RESET_IN

FIRST_TIME := /S_R_RESET

RESET_TO_CHIP = S_R_RESET
      + /FIRST_TIME * RESET_IN
         + FIRST_TIME * SEC_RESET_OK *
RESET_IN
```

PALASM Design Description

```
;PALASM Design Description

;---------- Declaration Segment --------
-
TITLE    Magic Packet Convertor
PATTERN  MAG_PCI.PDS
REVISION A
AUTHOR   David Stoenner
COMPANY  AMD
DATE     08/28/95

CHIP  _mag_pkt  PALCE16V8

;------------- PIN Declarations --------
-
PIN  1      EEPROM_CLK
PIN  2      /RESET_IN
PIN  3      /VCC_GONE
PIN  4      /LED3
PIN  10     GND
PIN  11     /OE
PIN  12     /RESET_TO_CHIP
PIN  13     /FIRST_TIME
PIN  15     LED3_OUT
PIN  19     /SLEEP
PIN  20     VCC

;-------- Boolean Equation Segment -----
-
EQUATIONS

LED3_OUT =   LED3 * VCC_GONE * FIRST_TIME
           + LED3_OUT * VCC_GONE

FIRST_TIME := VCC

RESET_TO_CHIP =   /FIRST_TIME * RESET_IN
                + FIRST_TIME * LED3_OUT
* VCC_GONE

SLEEP = VCC_GONE

;--------- Simulation Segment ----------
-
```

```
;---------------------------------------------------------------------
; This program is used with the PCnet family of parts to hook into the power interrupt
;so magic packet can be utilized.  For now we can use PCNTNW or the PCNTND (NDIS 2.0)
;drivers to initialize and run the PCnet_ISA II or the PCnet_PCI II controllers.  This
;TSR could be made to initialize the  controller and deal with the receive interrupt
;but that would require a foot print in memory that would contain at least a 2K buffer
;hence making the TSR 2K larger than needed to demonstrate the magic packet capability.
;This TSR works only if POWER.EXE is activated.
;---------------------------------------------------------------------



    .286
    .model tiny


    .stack 256


;---------------  REV HISTORY  ---------------


;3.0 Reworked adding PCI scan support
;2.0 Reworked the 2f Interrupt section by moving some of the initialization code from
;the original startup to the 2f Interrupt itself.  This was for the setting of csr4
;for the polling disabled and the setting of LED3 for the Magic Packet Interrupt.
;This now made the driver work with both Novell ODI and Windows for Workgroups NDIS
;2.0 drivers both in DOS and Windows environment.  I removed the requirement to have
;the MAC running before loading the 2f interrupt vector so, therefore, made the loading
;of this TSR independent of load order.
;
;1.0 Initial working model for PCnet_ISA II only.
;
;--------- END OF REV HISTORY  -----------
;
; PCI BIOS equates
;
PCI_FUNCTION_ID_1equ0b0h;PCI BIOS spec version 1
PCI_FUNCTION_ID_2equ0b1h;PCI BIOS spec version 2
PCI_BIOS_PRESENT equ 01h;PCI BIOS present
FIND_PCI_DEVICE  equ 02h;PCI device search
```

```
READ_CONFIG_BYTE equ 08h;PCI configuration space byte read
READ_CONFIG_WORD equ 09h;PCI configuration space word read
READ_CONFIG_DWORDequ 0ah;PCI configuration space dword read

;-----------------------------------------------------------------------
;
; AMD PCI devices equates
;
AMD_IDequ1022h;vendor ID
PCI_PCNETequ2000h;golden gate PCnet
MAX_PCI_DEV_NUMequ0ffh;maximum PCI device number

;-----------------------------------------------------------------------
; This data segment will be appended by the loader after the .code section.
; However we will need some storage for the stacks and variables we need to retain
;after we have returned to DOS.  Therefore, we will put some data storage in the code
;section. The following will be a list of the variables. Since we need some stack for
;operations we will set the ss=cs=ds and use a sp=00ffh
;
; Old_2F_Vector dd
;
; Found_address dw
;
; Our_code_segment dw
;
; Incomming_Stack_Segment dw
;
; Incomming_Stack_Pointer dw
;
;
    .data

message1 db "PCnet_ISA II Controller was not found.",0dh,0ah
l_message1 equ $ - message1

message2 db "PCnet Controller was not found and TSR was not installed.",0dh,0ah,
             "Please check setup.",0dh,0ah
l_message2 equ $ - message2

message3 db "PCnet controller that was found is not capable of Magic Packet.",
```

```
            0dh,0ah
l_message3 equ $ - message3

message4 db "PCnet Controller is now ready for Magic Packet.",0dh,0ah
l_message4 equ $ - message4

message5 db "This machine does not have a PCI BIOS.",0dh,0ah
l_message5 equ $ - message5

message6 db "This machine has a PCI BIOS type 1.",0dh,0ah
l_message6 equ $ - message6

message7 db "This machine has a PCI BIOS type 2.",0dh,0ah
l_message7 equ $ - message7

message8 db "PCnet_ISA II Controller was found.",0dh,0ah
l_message8 equ $ - message8

message9 db "PCnet_PCI II Controller was found.",0dh,0ah
l_message9 equ $ - message9

message10 db "PCnet_PCI II Controller was not found.",0dh,0ah
l_message10 equ $ - message10

message11 db "PCI BIOS read failure.",0dh,0ah
l_message11 equ $ - message11


PCI_BIOS        db      0               ; PCI BIOS version number
PCI_DEVICE_NUM  db      MAX_PCI_DEV_NUM ; maximum number of pci devices
PCI_DEVICE_FND  db      0               ; used to decide whether ISA or PCI
;------------------------------------------------------------------------------

        .code

main    proc    far

        .startup
```

```
;we jump over the code we want and only install the encloser code and leave

        jmp Install

; The data section needed in the code space goes in here.

Old_2F_Vector dd 0h

Found_address dw 0h

Our_code_segment dw 0h

Incoming_DX dw 0h

Incoming_DS dw 0h

Incomming_Stack_Segment dw 0h

Incomming_Stack_Pointer dw 0h


main    endp

;--------------------------------------------------------------------------
;   Here is the main Power Interrupt routine
;--------------------------------------------------------------------------

Power    proc

        pushf
        cmp ax, 0530bh
        jne NOT_MINE
        cmp bl, 2
        je  SUSPEND_REQ
        cmp bl, 3
        je  NORMAL_RESUME

; We get here because we got a 2f vector but not a shut down or power up so we
; will pass it on to the other linked routines of 2F.
```

```
NOT_MINE:

        popf
        jmp dword ptr cs:Old_2F_Vector

; We got here because the computer is going to go to sleep.  We need to set
; the PCnet controller into sleep mode.  This is done by putting a 6 in the
; CSR5 register.

SUSPEND_REQ:

        mov Incoming_DX,dx      ; Save off DS, DX and DS
        mov ax,ds
        mov Incoming_DS,ax

        mov ax, cs              ; fix up the data segment pointer
        mov ds, ax

; first we will program the LED3 register for magic packet indication and
; then we will set the part for magic packet detection mode.

        mov dx, Found_address   ; point the RAP to the register wanted
        add dx, 012h
        mov ax, 0007h
        out dx, ax

; DB9 of iscar7 controls the use of magic packet detection for LED3

        add dx, 04h
        mov ax, 0200h
        out dx, ax


        sub dx, 04h             ; realign DX to the RAP
        mov ax, 5
        out dx, ax

        sub dx, 2               ; point the DX to RDP
        mov ax, 026h            ; put into magic packet suspend mode and
```

```
                                ; broadcast accept
        out dx, ax

        mov ax,Incoming_DS      ; Restore DS, AX and DX
        mov ds,ax
        mov dx,Incoming_DX

        mov ax,0530bh           ; repair the ax register

        popf
        jmp dword ptr cs: Old_2F_Vector

; We got here because the computer is going to wake up.  We need to clear the
; Magic Packet look for bit.  We will set CSR5 back to 2

NORMAL_RESUME:

        mov Incoming_DX,dx      ; Save off DS, DX and DS
        mov ax,ds
        mov Incoming_DS,ax

        mov ax, cs              ; fix up the data segment pointer
        mov ds, ax

        mov dx, Found_address   ; point the RAP to the register wanted
        add dx, 012h
        mov ax, 5
        out dx, ax

        sub dx, 2               ; point the dx to RDP
        mov ax, 02h             ; take out of magic packet mode
        out dx, ax

        mov ax,Incoming_DS      ; Restore DS, AX and DX
        mov ds,ax
        mov dx,Incoming_DX

        mov ax, 0530bh          ; repair the ax register
```

```
        popf
        jmp dword ptr cs: Old_2F_Vector

Power   endp

end_of_program:


Install:

        push    cs                      ; just in case
        pop     ds                      ; ds is now set to the cs for com file

; We will now save the current cs for later use

        mov ax,cs
        mov Our_Code_Segment, ax

; Now we will look for a PCnet ISA.  The first location can be 0x200 base

        mov dx, 020fh

LOOK_AGAIN:

        in  al, dx
        cmp al, 057h
        jz  ISA_FOUND

        add dx, 020h
        cmp dx, 040fh

        jz  CHECK_FOR_PCI

        jmp LOOK_AGAIN

; We did not find an ISA device so now we will look for a PCI device.  This
; will involve first seeing if we have a PCI machine.  Then if a PCI BIOS is
; present we will then scan the PCI backplane for an AMD device and save the
; address at configuration space + 10h as the card address.
```

```
;
; now we will put out a message about finding no ISA adapter.

CHECK_FOR_PCI:

        mov bx, 0001h                   ; output message no isa ii card
        lea dx, message1
        mov cx, l_message1
        mov ah, 40h
        int 21h

;--------------------------------
; test PCI BIOS spec version 2 interface
;--------------------------------
        xor     bx,bx                ;clear BX
        xor     cx,cx                ;clear CX
        xor     dx,dx                ;clear DX
movah,PCI_FUNCTION_ID_2;assume PCI BIOS spec version 2
moval,PCI_BIOS_PRESENT ;request for PCI BIOS support
int1ah  ;PCI BIOS interface
;--------------------------------
; check return value from PCI BIOS spec version 2
;--------------------------------
jccheck_PCI_BIOS_ver1;jump, if carry set
;
        cmp     dx,"CP"                 ;check for PCI signature
jnecheck_PCI_BIOS_ver1;jump, if PCI BIOS version is not 2
;
orah,ah;check present status
jnzno_PCI_BIOS;jump, if no PCI BIOS present
;
movPCI_BIOS,2;set PCI BIOS version = 2
movPCI_DEVICE_NUM,cl;save CL = # of last PCI bus in system

        mov bx, 0001h                   ; output message about BIOS type
        lea dx, message7
        mov cx, l_message7
        mov ah, 40h
        int 21h
```

```
        jmp     SCAN_FOR_CARD           ;jump, get hardware mechanism

;--------------------------------
; If not version 2 then test PCI BIOS spec version 1 interface
;--------------------------------

check_PCI_BIOS_ver1:

xor     cx,cx                   ;clear CX
xordx,dx;clear DX
movah,PCI_FUNCTION_ID_1;assume PCI BIOS spec version 2
moval,PCI_BIOS_PRESENT;request for PCI BIOS support
int1ah;PCI BIOS interface

;--------------------------------
; check return value from PCI BIOS spec version 1
;--------------------------------

        jc      no_PCI_BIOS             ;jump, if carry set
;
cmpdx,"CP";check for PCI signature
jneno_PCI_BIOS;jump, if no PCI BIOS present
cmpcx," I";check for PCI signature
jneno_PCI_BIOS;jump, if no PCI BIOS present
;
movPCI_BIOS,1;set PCI BIOS version = 1
movPCI_DEVICE_NUM,MAX_PCI_DEV_NUM;maximun PCI device number
;assume mechanism 2 for PCI BIOS ver1

        mov bx, 0001h                   ; output message about BIOS type
        lea dx, message6
        mov cx, l_message6
        mov ah, 40h
        int 21h


        jmp     SCAN_FOR_CARD           ;jump, get hardware mechanism
```

```
;--------------------------------
; no PCI BIOS exists
;--------------------------------

no_PCI_BIOS:

        mov bx, 0001h                   ; output message no PCI BIOS
        lea dx, message5
        mov cx, l_message5
        mov ah, 40h
        int 21h

        jmp     NOT_HERE                ;exit

;--------------------------------
;  search AMD PCI PCNet devices
;--------------------------------

SCAN_FOR_CARD:

xorsi,si;SI = 0 (index initialize)
;
movdx,AMD_ID;DX = AMD vender ID
;
movcx,PCI_PCNET;assume CX = PCI PCNet device ID

movah,PCI_FUNCTION_ID_1;assume BH = PCI BIOS spec version 1
cmpPCI_BIOS,1;check PCI BIOS version = 1
jePCI_BIOS_version_set;jump, if PCI BIOS version determined
movah,PCI_FUNCTION_ID_2;BH = PCI BIOS spec version 2

PCI_BIOS_version_set:

;--------------------------------
; find AMD PCI device through BIOS API
;--------------------------------

        mov     al,FIND_PCI_DEVICE          ;request for AMD PCI device
        int     1ah;                         PCI BIOS interface
```

```
; This function will return in the BX register the bus number in BH and the
; device number in BL.
; test for error carry = 1

        jnc         find_PCI_device_success         ;find PCI device success

search_fail:

        mov bx, 0001h                               ;output message no PCI II card
        lea dx, message10
        mov cx, l_message10
        mov ah, 40h
        int 21h

        jmp NOT_HERE

                                                    ;

find_PCI_device_success:
       or           ah,ah                           check search successful
       jnz          search_fail                     jump, if search failed


; we have now found a PCnet_PCI device.  Now we will get its address at
; configuration space + 10h.  BX is being maintained here from the previous
; call with the bus number and the device number

       mov          ah,PCI_FUNCTION_ID_1;    assume AH = PCI BIOS spec version 1
       cmp          PCI_BIOS,1               ;check PCI BIOS version = 1
       je           PCI_BIOS_ver_set         ;jump, if PCI BIOS version determined
       mov          ah,PCI_FUNCTION_ID_2     ;AH = PCI BIOS spec version 2
PCI_BIOS_ver_set:
;
        mov     di,010h                 ;DI = PCI config space data bytes

;--------------------------------
; read PCI configuration space
;--------------------------------
```

```
        mov     al,READ_CONFIG_DWORD    ; request for AMD PCI device
        int     1ah                     ; PCI BIOS interface
;
        jc      READ_FAIL               ; exit, if error
;
        or      ah,ah                   ; check return code
        jnz     READ_FAIL               ; jump, if error happened

; CX now contains the address of the card but the address has its lower bit
; set in PCI config register to indicate it is I/O not memory.  Therefore we
; will have to mask off that bit.
                                        ;
        mov     ax,cx                   ; save config byte
        and     ax,0fffeh               ; mask off the lower bit to 0
        mov Found_address, ax           ; Save off the Found_address

        mov ax,1                        ; 1 = PCI Device found
        mov PCI_DEVICE_FND, al

        mov bx, 0001h                   ; output message PCI card found
        lea dx, message9
        mov cx, l_message9
        mov ah, 40h
        int 21h

        jmp FOUND

READ_FAIL:

        mov bx, 0001h                   ; output message read address fail
        lea dx, message11
        mov cx, l_message11
        mov ah, 40h
        int 21h

        jmp NOT_HERE
```

```
ISA_FOUND:

        sub dx, 0fh
        mov ax, dx
        mov Found_address, ax           ; Save off the Found_address

        mov ax,0                        ; 0 = ISA Device found
        mov PCI_DEVICE_FND, al

        mov bx, 0001h                   ; output message ISA card found
        lea dx, message8
        mov cx, l_message8
        mov ah, 40h
        int 21h

; We get here if we found the card at an address.  Now DX contains the address
; of the found card but with an offset of 0xF so we need to adjust.

FOUND:



; Now that we have found the controller we must check to see if the PCnet
; controller can handle the Magic Packet.  Only PCnet ISA II and PCnet_PCI II
; can handle a magic packet.  PCnet ISA II can be identified with a 01h
; in the address space + 9h register.  While a PCnet_PCI II controller can
; be found by a 1X in the rev register of the PCI configuration register.

        mov ah, 0
        mov al,PCI_DEVICE_FND
        jnz USE_PCI_CHECK

; This is the ISA check

        mov dx, Found_address               ; point the RAP to the register wanted
        add dx, 09h
        in  al, dx
        cmp al, 01h
        jz  INTERRUPT_LINK
```

```
; This is the PCI check

USE_PCI_CHECK:

        mov dx, Found_address               ; point the RAP to the register wanted
        add dx, 09h
        in  al, dx
        cmp al, 11h
        jz  INTERRUPT_LINK

        mov bx, 0001h
        lea dx, message3
        mov cx, l_message3
        mov ah, 40h
        int 21h
        jmp RETURN_TO_DOS


INTERRUPT_LINK:

        mov dx, Found_address
        add dx, 012h


; Now we will program the csr5 register turn on magic packet mode.

        mov ax, 05h         ; point the RDP to CSR 5
        out dx, ax

        sub dx, 02h
        mov ax, 02h
        out dx, ax

; Now we will program the csr4 register to disable polling just in case a
; motherboard has a timer on the BREQ/DREQ activity of the controller.

        add dx,02h              ; Realign DX to the RAP

        mov ax, 04h             ; point the RDP to CSR 4
```

```
        out dx, ax

        sub dx, 02h
        in  ax, dx
        or  ax, 01000h          ; set the disable polling bit
        out dx, ax



        add dx, 02h         ; Now reset the address pointer back to 0
        mov ax, 00h
        out dx, ax

; We now need to hook the 2F interrupt (Power Management).  Since this
; interrupt is cascaded with other programs we need to get the old 2F
; interrupt vector and save it in Old_2F_Vector and then put our pointer
; in 2F.

        xor ax, ax
        mov es, ax

        pushf
        cli

        mov ax, es:[2fh * 4 ]        ; Get the old vector
        mov word ptr Old_2F_Vector, ax
        mov ax, es:[2fh * 4 + 2]
        mov word ptr Old_2F_Vector + 2 , ax

        lea ax, Power
        mov es:[02fh * 4], ax
        mov ax, cs
        mov es:[02fh * 4 + 2], ax

        popf                        ; This re-enables the interrupts

; Now we display the message that we are magic packet ready

        mov bx, 0001h
        lea dx, message4
```

```
        mov cx, l_message4
        mov ah, 40h
        int 21h

; now we return to DOS but we leave the TSR for POWER

        mov al, 0
        lea dx, end_of_program
        mov ah, 31h
        int 27h


NOT_HERE:

; Now we display the message that the PCnet was not found and TSR was not
; installed.

        mov bx, 0001h
        lea dx, message2
        mov cx, l_message2
        mov ah, 40h
        int 21h

RETURN_TO_DOS:

        mov ah,4ch
        int 21h

    end
       .
```

Trademarks

AMD, the AMD logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc.

Am186, Am386, Am486, Am29000, *b*IMR, eIMR, eIMR+, GigaPHY, HIMIB, ILACC, IMR, IMR+, IMR2, ISA-HUB, MACE, Magic Packet, PCnet, PCnet-*FAST*, PCnet-*FAST*+, PCnet-Mobile, QFEX, QFEXr, QuASI, QuEST, QuIET, TAXIchip, TPEX, and TPEX Plus are trademarks of Advanced Micro Devices, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.