Advanced
Micro
Devices

PCnet™ Application Examples

PCnet-ISA to MC68000
PCnet-ISA with a Big Endian Processor

Application Note

PCnet Application Examples

Advanced Micro Devices

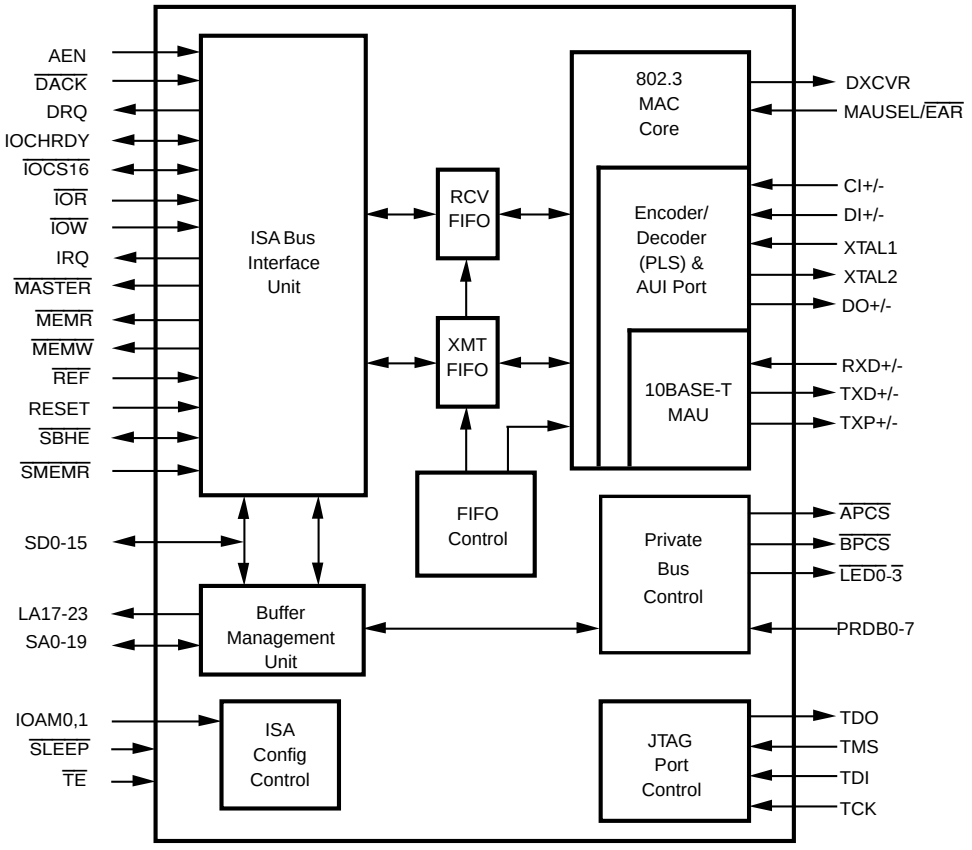
Application Note

Mike Keith & Mike Santoro

I. PCnet-ISA CONTROLLER IN AN EMBEDDED APPLICATION

Interfacing to the Motorola MC68000 Microprocessor

The design of an interface between the 68000 microprocessor and the PCnet-ISA Ethernet controller is presented. Among the various design options available, and particularly whether the design effort is new or is an upgrade of an established LANCE-based design, one of two alternative design solutions may be optimal. A major consideration is whether established system software exists and must be preserved as is or whether the software can be revised and to what degree. New designs will obviously have greater freedom among design options; LANCE-based system upgrades may have very little.



19889A-1

Figure 1. PCnet Ethernet Controller (Block Diagram: Bus Master Mode)

Publication# **19889** Rev. **A** Amendment **/0**
Issue Date: **June 1995**

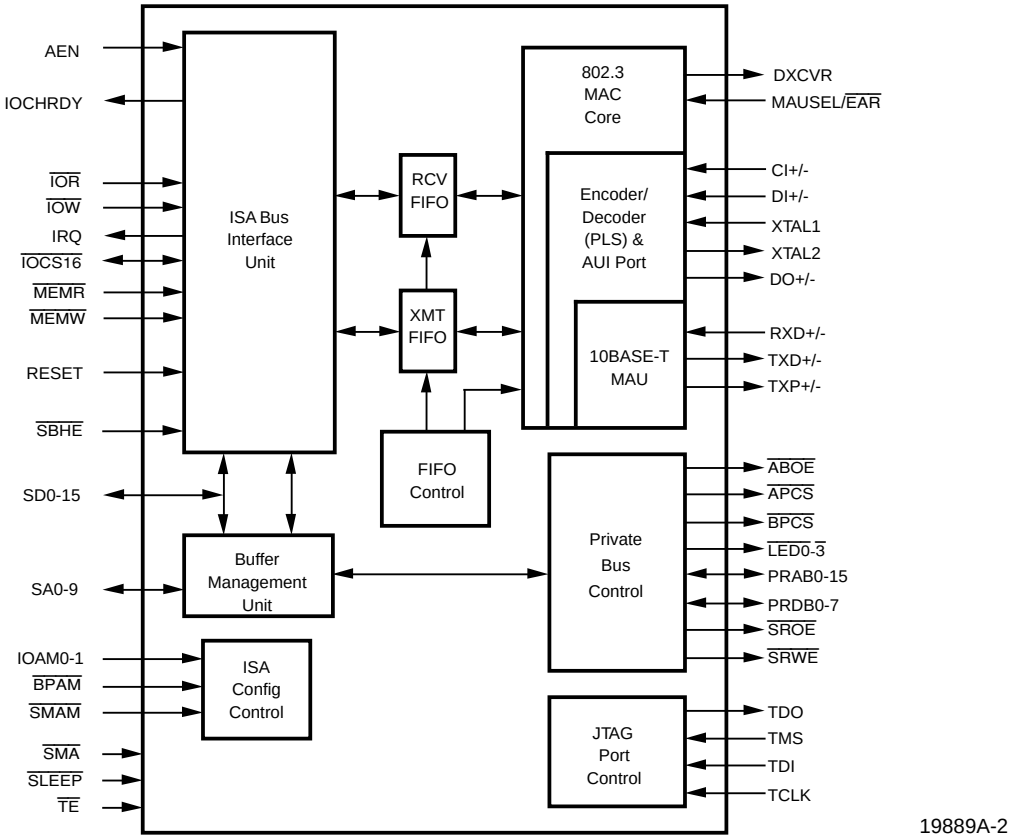


Figure 2. PCnet Ethernet Controller (Block Diagram: Shared Memory Mode)

The PCnet-ISA controller is a highly-integrated single-chip Ethernet controller designed to interface directly with the PC/AT Industry Standard Architecture (ISA) system bus. The PCnet-ISA controller is 100 percent software compatible with the Am7990 LANCE (Local Area Network Controller for Ethernet). The simple and inexpensive hardware interface to the 68000 bus, the overwhelming advantage of preserving the investment in established system software for LANCE-based designs, and the device cost, power, performance, integration, and reliability benefits of the latest- generation-technology make the PCnet-ISA controller an ideal solution for advancing the state of the art in Ethernet connectivity for the 68000.

System Architecture

The hardware architectural design will affect system performance, material cost, real estate, software structure, etc. There are several system architectural options to consider.

Block diagrams for the PCnet-ISA in Bus Master Mode and Shared Memory Mode are shown in Figures 1 and 2.

Bus-Master Architecture vs. Shared-Memory

In the bus-master architecture, the PCnet-ISA controller is allowed to arbitrate for and acquire control of the 68000 system bus. This provides the PCnet-ISA controller with a DMA path into system memory, through which data is exchanged with the 68000. In contrast, the shared-memory architecture requires additional dedicated memory coupled directly to the PCnet-ISA controller for the exchange of information. This does save the 68000 from being preempted during the PCnet-ISA controller data transfers, but carries the extra costs of:

a) the additional memory components,

b) the power they must be supplied,

c) the increase in required real estate, and

d) the reduction in system performance and CPU utilization associated with the need now to make an extra copy of all network data.

A bus-master design is the architecture of choice; this will maintain software-compatibility with the LANCE (the LANCE does not support the shared-memory architecture).

Byte-Swapping

The 68000 is characterized as a big-endian machine; odd-addressed bytes reference data on the low-byte data lines D<0–7>, even-addressed bytes on the high-byte data lines D<8–15>. The PCnet-ISA controller

is the opposite, a little-endian machine, where even-addressed bytes reference data on the low-byte data lines D<0–7> and odd-addressed bytes on the high-byte data lines D<8–15>. (Unlike the LANCE, the PCnet-ISA controller does not contain a "byte-swapping" mechanism which optionally configures it as a big-endian machine.) Any byte-oriented data which both devices share access to, such as network packet data, will be inconsistently byte- addressed between the two devices at the byte level.

As an example, consider packet data being received and accepted by the PCnet-ISA controller from the network and placed into system buffer memory. The first byte is placed into low-byte-memory at receive-buffer base-address-offset 0, the second byte at high-byte-memory offset-address 1. The 68000, however, will find the first byte at low-byte-memory address 1, the second byte at high-byte-memory address 0. The 68000 data is not in sequential order; it is not consistent with the PCnet-ISA controller's representation of the data. This inconsistency occurs on all packet data, both transmit and receive.

Thus, it is seen that individual network data-bytes are addressed in memory by the PCnet-ISA consistent with the construction of the network packet. The 68000 addressing is inconsistent with the network data. To link into a standard 802.3 network, some form of byte-swapping is required.

(As an aside, it is interesting to note the implication of network systems with a 68000 and the PCnet-ISA controller at each and every node. These systems could presumably communicate freely with one another without byte-swapping, even though the network data would not comply with the 802.3 specification. Data would be "reversed" once when being transmitted onto the network, then reversed back again, to normal, at reception—this would be transparent to the 68000 at each end. The difficulty with this scheme lies in the node addressing and automatic recognition by the PCnet-ISA controller. If the address-field bytes are swapped, the node address must be compensated in order to recognize the targeted address; this may be simple to handle in a small dedicated network. The remainder of this discussion assumes full compliance with 802.3 networks.)

Consider now the distinction of "word oriented" (16-bit) data. While individual data bytes are not consistently addressed between the 68000 and the PCnet-ISA controller, data words are. The bytes within the word are not addressed the same, but the word address is common to both devices. Thus, for example, the most significant bit (MSB) of the data at word-address 0 is the same bit for both devices and is transferred on D15. However, this bit is the MSB of byte 0 for the 68000, byte 1 for the PCnet-ISA controller.

As a result, it is advantageous to have the non-network data—the descriptor ring, initialization block, and the PCnet-ISA controller internal register data—organized and accessed strictly as 16-bit words. This maintains consistent addressing between the two devices in this area which greatly reduces the complexity of the overall design, is quite easily implemented, and eliminates the need to byte-swap this data. It does require that all software accesses to this non-network data be coded as word accesses only.

Byte-swapping, therefore, is required only on the network data. There are at least three alternative byte-swapping solutions to consider:

Software Swap—The 68000 can perform a byte-swap function in software. Referring to network data, the first (byte #0, using byte addresses from the PCnet-ISA controller point of reference) and all remaining even-addressed data-bytes are shifted down to the lower half of the data bus; the second (byte #1) and all remaining odd-addressed data-bytes are shifted up to the upper half of the data bus. This may be accomplished through series of move instructions or by an eight-position circular shift of data processed as 16-bit words. (Note that there is a word-swap function in the 68000, but no byte-swap.) This will make the network-data byte-addresses consistent between the 68000 and the PCnet-ISA controller. Data must pass through this "byte-swap filter" on any transfer between the 68000 and any transmit or receive buffer.

The overwhelming drawback of this approach is the significant reduction in system throughput and CPU performance associated with the inefficient and time consuming task of software-swapping each and every data byte which is transferred on the network. Additionally, this option does not preserve LANCE software compatibility. While viable and possibly favorable under particular circumstances, this software-swap is rejected in lieu of more attractive alternatives.

Data-Bus Swap—The PCnet-ISA controller data-bus can be hard-wired in reverse so as to provide the effect of a permanent byte-swap function. Here, SD<0–7> is connected to high-byte memory, SD<8–15> is connected to low-byte memory. This scheme will maintain consistent addressing of all network data passing between the 68000 and the PCnet-ISA controller.

The major negative here is the impact on non-network data such as in accesses to the descriptor rings, the initialization block, or the PCnet-ISA controller internal registers (as discussed above). Now this data must be byte-swapped in order to negate the effects of the hard-wired swap. Additionally, there is the confusion factor associated with keeping track of bit positions and data flows now that the data-bus is used in a non-standard mode. And, as before, LANCE-compatibility is no longer preserved.

The attraction to this option lies in the simplicity of the design. No special hardware is required to handle any byte-swapping.

The byte-swapping on the non-network data may occur via:

(i) a software filter, as discussed earlier—the system performance degradation associated with the software-swap of non-network data is small since these data transfers typically occur relatively infrequently and usually aren't time critical. And the software function is fairly simple to implement.

Or,

(ii) simply by redefining the bits in the non-network data registers and tables—this is especially applicable to new software development efforts. If the definition of the bits in these registers/tables are changed to reflect the effect of the data-bus swap, the software can then be designed to access this data properly and all is well and transparent to the overall system; the byte-swapping is accomplished in the system documentation. Note that the danger of confusing bit positions or functions here is real and significant and must be cautiously considered.

This Data-Bus Swap design alternative is very attractive, particularly to original design efforts. It is the basis for the first design example, detailed below.

Hardware-Buffer Swap—This alternative achieves full architectural and software compatibility with the LANCE. It consists of discrete hardware buffers to byte-swap network data while maintaining a standard and direct hardware and software interface to non-network data. The swapping occurs on the PCnet-ISA controller data-bus. This isolates the swapping function from the remainder of the system.

The obvious advantage here is backward compatibility with established LANCE system software. A LANCE-to-PCnet-ISA controller system upgrade can be implemented entirely through hardware design. There is no need to revise software which may be mature and field-reliable.

The only significant caveat here lies in the technique used to determine when to execute a swap. (Recall that non-network data is not to be swapped.) There are no bus-cycle distinctions between the PCnet-ISA controller accesses to network and non-network data. So how do you know when to make a swap? One possibility is to base the decision on the address used in the transfer: If the system memory map is partitioned into "swap" and "no-swap" areas, the swap decision can be made on a simple decode of the target address. This will obviously place restrictions on the locations available for network buffers (i.e., swap-memory only) and thus may impact LANCE backward-compatibility. This impact is expected to be minimal and easy to work around. Alternatively, a different or more complex hardware-swap decision-mechanism may be able to match the existing software footprint and achieve the desired 100 percent compatibility.

The costs of the Hardware-Buffer Swap design are minimal and are detailed in the second design example, following.

Design Examples

The 68000-to-PCnet-ISA controller Data-Bus Swap and Hardware-Buffer Swap design examples are illustrated in Figures 3 through 11.

The design options presented do not address the PCnet-ISA controller interfaces to the network, the address PROM, or the boot PROM. These are independent interfaces which are not affected by the PCnet-ISA controller system bus interface design. The one exception is the boot PROM, a PC-specific function which will not typically apply to a 68000-based system.

Design Notes

The remainder of this document refers to Figures 3 through 11:

1. Description of diagrams:

   Figure 3: Block Diagram

   Figure 4: Data-Bus Swap Schematic, Sheet 1

   Figure 5: Data-Bus Swap Schematic, Sheet 2

   Figure 6: Data-Bus Swap MACH 110 Design File

   Figure 7: Hardware-Buffer Swap Schematic, Sheet 1

   Figure 8: Hardware-Buffer Swap Schematic, Sheet 2

   Figure 9: Hardware-Buffer Swap MACH 110 Design File

   Figure 10: Address Decode 16V8 Design File

   Figure 11: System Memory Map Example

2. The following new signals are defined:

PCNCS_	PCnet-ISA controller chip select
MEMCS_	system memory chip select
HMCS_	high-byte memory chip select
LMCS_	low-byte memory chip select
MOE_	memory output enable
MWE_	memory write enable
IORS_	I/O read strobe
IOWS_	I/O write strobe
S/R_	send/receive, transceiver direction control
SWAP_	swap-transceiver enable
NOSWAP_	noswap-transceiver enable

3. These designs meet worst-case timing with the parts indicated and with the 68000 running at its full speed of 16.67 MHz.
4. The system memory ICs and the address decoding scheme are an example of minimal system resources only and are not necessarily recommendations for a specific implementation.
5. All 68000 accesses to the PCnet-ISA controller and to non-network data should be coded as 16-bit accesses; the hardware is configured for operation as such.
6. "PU" pull-up and "PD" pull-down resistors are loosely recommended as 3.3K ohm resistors. These values must be carefully considered in an actual design which may differ considerably in specific areas. Watch especially signal SA0 which is sure to have additional resources sharing its load.
7. The decision on whether or not to swap the data (via the transceivers) in the Hardware-Buffer Swap design is made via address decoding per the discussion in the text. Refer to Figure 11. The system SRAM is fully shadowed—all of the memory is accessible as either swapped or non-swapped data, as a function of A16. The example depicted is quite flexible and open to alternative schemes which may be tailored to an existing (i.e., LANCE-based system) software footprint.
8. The PCnet-ISA controller bus-cycle control registers, MSRDA (in ISACSR0) and MSWRA (in ISACSR1), should be configured as follows:

   MSRDA—equal to 3 (minimum) for a 150 ns $\overline{\text{MEMR}}$ cycle minimum

   MSWRA—equal to 1 (minimum) for a 50 ns $\overline{\text{MEMW}}$ cycle minimum
9. The MACH110 design is partitioned into six major segments: bus master exchange, wait-state control, system memory control, hardware swap-buffer control, misc control, and misc.
10. The MACH "bus master exchange" circuit:
    a. PCnet-ISA controller output-signal $\overline{\text{MASTER}}$ is not used. $\overline{\text{DACK}}$ is its functional equivalent and is the indicator of the PCnet-ISA controller bus-mastership when it is LOW. Eliminating the use of $\overline{\text{MASTER}}$ frees one MACH pin.
    b. The two flip-flops used to generate $\overline{\text{ML1}}$ and $\overline{\text{ML2}}$ act as a synchronizing circuit to capture the asynchronous ($\overline{\text{DRQ}}$ + DACK_) term. This is to minimize metastable susceptibility in the generation of $\overline{\text{BGACK}}$.
11. The MACH "wait-state control" circuit:
    a. A minimum of two 68000 wait-states are inserted on 68000-to-PCnet-ISA controller accesses.
    b. DTACK_ is a totem-pole-output wait-state control signal. To provide a wired-OR capability for other system resources to insert 68000 wait-states via DTACK_ control, the open-drain and unused signal ZDTACK_ is available—simply use ZDTACK_ (with a pull-up resistor, of course) instead of DTACK_. As implemented, ZDTACK_ may insert an additional (i.e., three, minimum) wait-state if worst case timing occurs. If this cannot be tolerated, a faster MACH device is needed.
    c. The IOCHRDY output from the PCnet-ISA controller may or may not become active to insert additional 68000 wait-states in 68000-to-PCnet-ISA controller accesses. IOCHRDY can be expected to insert additional wait-states under two conditions: (a) rapid back-to-back PCnet-ISA controller accesses, or (b) accesses to private and slow PCnet-ISA controller resources such as the Address PROM.
12. The MACH "misc control circuit": 68000 interrupts are configured for the auto vector mode via FC and VPA_.
13. The two 74F32 OR-gates are required so as to meet the PCnet-ISA controller address hold time requirements.
14. The PLD designs were compiled on AMD's Palasm 4, Version 1.5.
15. DISCLAIMER! Caution: This design has NOT been prototyped. No design verification of any physical circuitry has occurred at this time.

References

1. Ethernet/IEEE 802.3 Family 1994 World Network Data Book/Handbook, AMD PID# 14287C.
2. Am79C960 PCnet-ISA Controller Technical Manual, May 1992, AMD PID# 16850B.
3. Hilf, Werner and Nausch, Anton, The 68000 Family, Volumes 1 and 2 (Englewood Cliffs: Prentice Hall, 1989). ISBN# 0-13-541525-X.
4. MACH Family Data Book, High Density EE CMOS Programmable Logic, Summer 1992, AMD PID# 14051F.
5. MACH Technical Briefs, 1991, AMD PID# 15972A.

6. M68000 Family Reference, 1990, Motorola Reference # M68000FR/AD.

7. 8-/16-/32-Bit Microprocessor User's Manual, Eighth Edition, Motorola Reference # M68000UM/AD REV 7.

8. M68000 Electrical Specifications, Motorola Semiconductor Technical Data, Motorola Reference # M68000EC/D.

9. Solari, Edward, AT Bus Design (San Diego: Annabooks, 1991). ISBN# 0-929392-08-6.

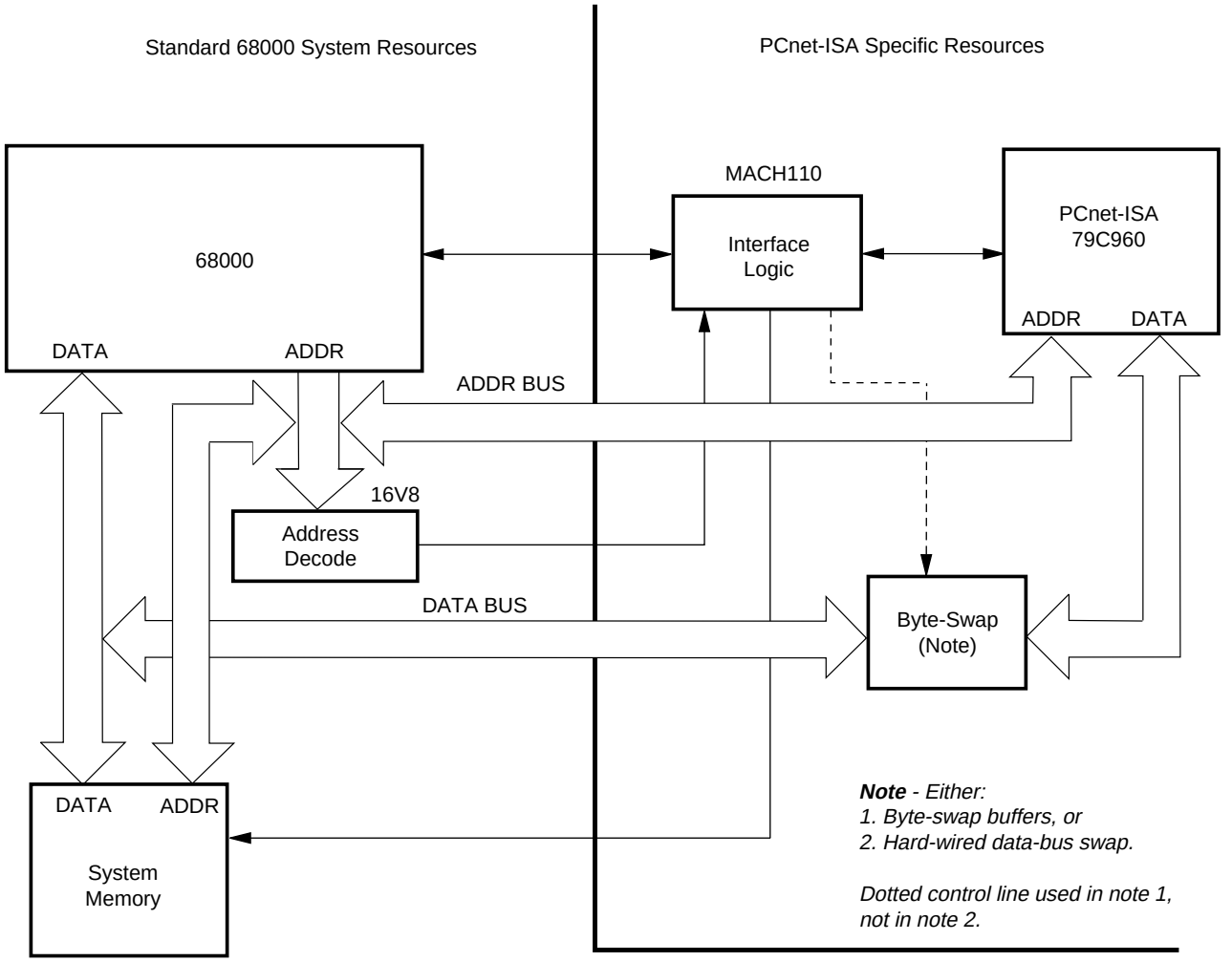
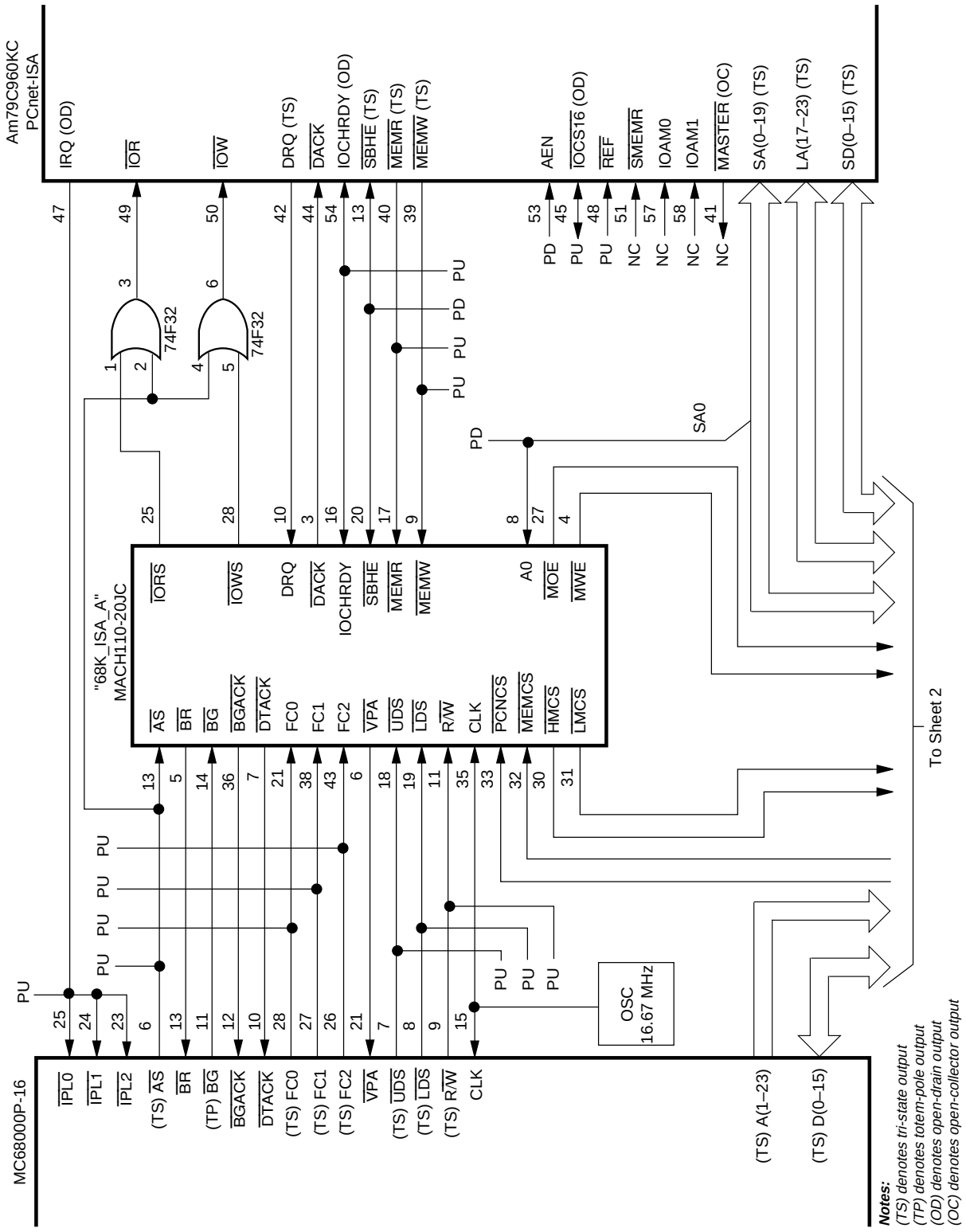
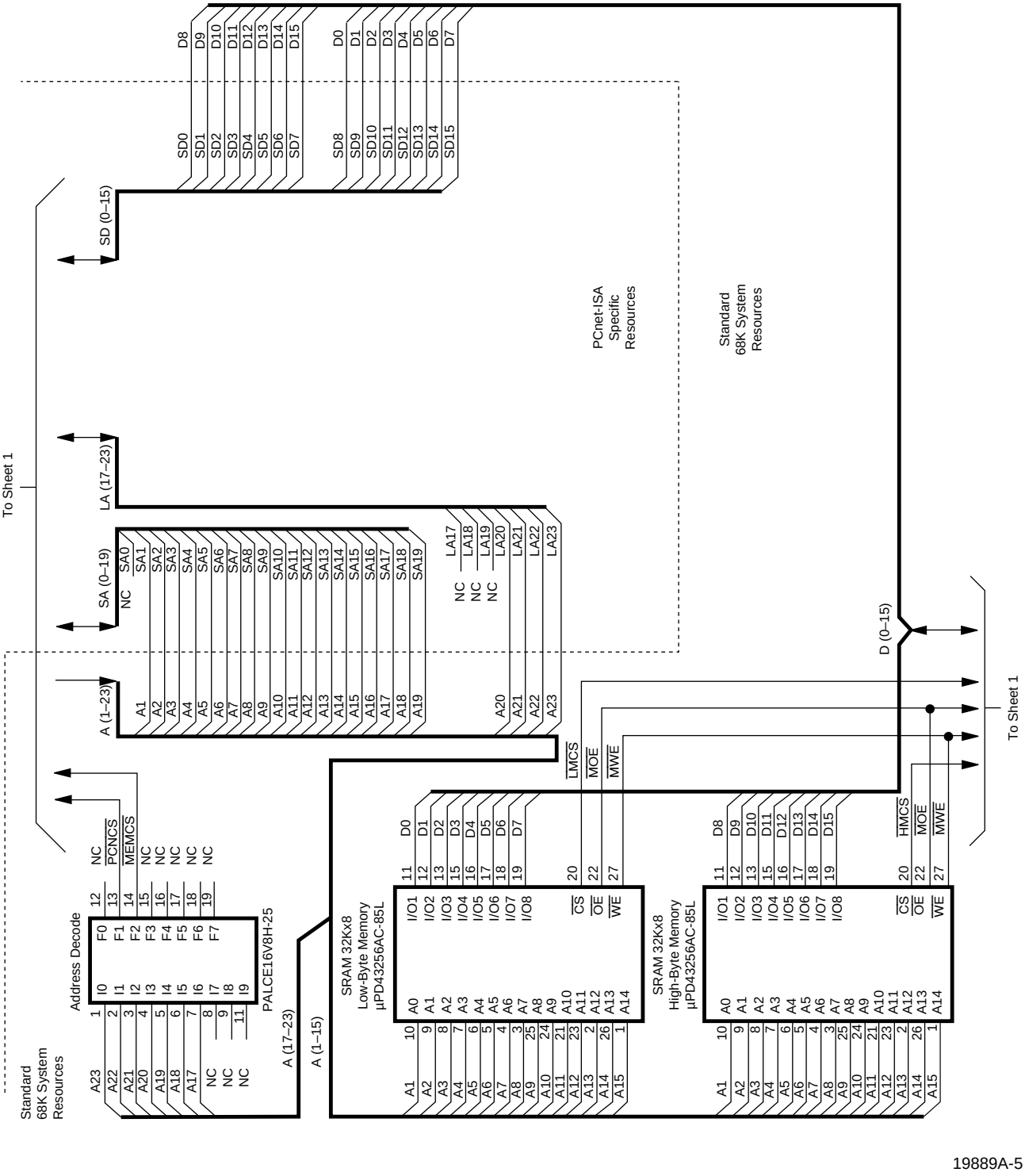


Figure 3. PCnet-ISA-to-68K Interface—Block Diagram



19889A-4

Figure 4. PCnet-ISA-to-68K Interface—Data-Bus Swap (Sheet 1)



19889A-5

Figure 5. PCnet-ISA-to-68K Interface—Data-Bus Swap (Sheet 2)

```
TITLE 68K-to-PCnet-ISA Interface, with Data-Bus Swap
PATTERN
REVISION 0
AUTHOR Mike Keith, Field Applications Engineer
COMPANY Advanced Micro Devices
DATE  2/17/93
CHIP _68K_ISA_A MACH110
;———————————— PIN Declarations ————————————
PIN 2          ZDTACK_      COMBINATORIAL    ; OUTPUT
PIN 8          A0                            ; INPUT
PIN 11         R_W_                          ; INPUT
PIN 13         AS_                           ; INPUT
PIN 33         PCNCS_                        ; INPUT
PIN 17         MEMR_                         ; INPUT
PIN 9          MEMW_                         ; INPUT
PIN 32         MEMCS_                        ; INPUT
PIN 18         UDS_                          ; INPUT
PIN 19         LDS_                          ; INPUT
PIN 20         SBHE_                         ; INPUT
PIN 4          MWE_         COMBINATORIAL    ; OUTPUT
PIN 27         MOE_         COMBINATORIAL    ; OUTPUT
PIN 31         LMCS_        COMBINATORIAL    ; OUTPUT
PIN 30         HMCS_        COMBINATORIAL    ; OUTPUT
PIN 28         IOWS_        COMBINATORIAL    ; OUTPUT
PIN 25         IORS_        COMBINATORIAL    ; OUTPUT
PIN 35         CLK                           ; INPUT
PIN 14         BG_                           ; INPUT
PIN 10         DRQ                           ; INPUT
PIN 16         IOCHRDY                       ; INPUT
PIN 21         FC0                           ; INPUT
PIN 38         FC1                           ; INPUT
PIN 43         FC2                           ; INPUT
PIN 6          VPA_         COMBINATORIAL    ; OUTPUT
PIN 7          DTACK_       REGISTERED       ; OUTPUT
NODE 8         Q0           REGISTERED       ; FEEDBACK
NODE 28        BRS_         REGISTERED       ; FEEDBACK
NODE 30        ML2_         REGISTERED       ; FEEDBACK
NODE 29        ML1_         REGISTERED       ; FEEDBACK
PIN 3          DACK_        COMBINATORIAL    ; OUTPUT
PIN 36         BGACK_       COMBINATORIAL    ; OUTPUT
PIN 5          BR_          COMBINATORIAL    ; OUTPUT
```

Figure 6. Mach 110 Design File—Data-Bus Swap

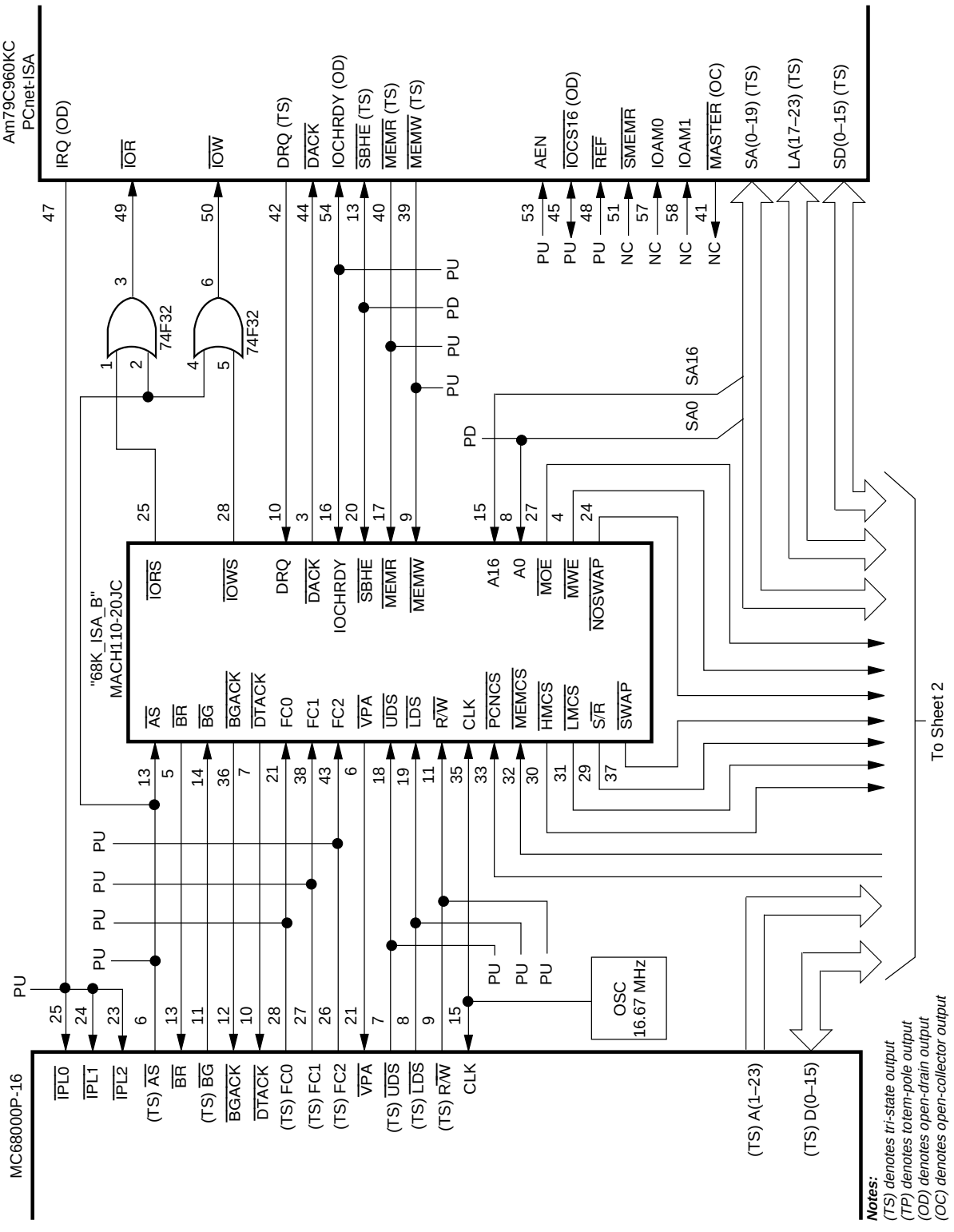
MACH110 Design File—68000-to-PCnet-ISA with Data-Bus Swap

```
NODE 1 GLOBAL
EQUATIONS
;————————————————- Bus master exchange:
DACK_           =  /(AS_ *
                   /(BG_ * BGACK_) )
BGACK_          =  (/DRQ + DACK_) + ML2_
ML1_            := /DRQ + DACK_
ML2_            := ML1_
BRS_            := ML2_
BR_             =  /(BRS_ * DRQ)
;————————————————- Wait state control:
Q0              := AS_ + PCNCS_ + /DTACK_
DTACK_          := Q0 + AS_
Q0.RSTF         =  /IOCHRDY
DTACK_.RSTF     =  /IOCHRDY
ZDTACK_         =  GND                  ;if needed: tri-state form of DTACK_
ZDTACK_.TRST    =  /DTACK_              ;if needed: tri-state form of DTACK_
;————————————————- System memory control:
MOE_            =  MEMCS_ + (MEMR_ * (/R_W_ + /DACK_))
MWE_            =  MEMCS_ + (MEMW_ * R_W_)
HMCS_           =  MEMCS_ + (UDS_ * (SBHE_+ DACK_))
LMCS_           =  MEMCS_ + (LDS_ * (A0 + DACK_))
;————————————————- Misc control:
VPA_            =  /(FC0 * FC1 * FC2 * /AS_)        ; 68K autovector interrupts
IORS_           =  /R_W_ + PCNCS_ + AS_
IOWS_           =  R_W_ + PCNCS_ + AS_
;————————————————- Misc:
ML1_.CLKF       =  CLK
ML2_.CLKF       =  CLK
BRS_.CLKF       =  CLK
Q0.CLKF         =  CLK
DTACK_.CLKF     =  CLK
ML1_.RSTF       =  GND
ML2_.RSTF       =  GND
BRS_.RSTF       =  GND
GLOBAL.SETF     =  GND
```

19889A-6

Figure 6. Data-Bus Swap Mach 110 Design File (Continued)

MACH110 Design File—68000-to-PCnet-ISA with Data-Bus Swap



19889A-7

Figure 7. PCnet-ISA-to-68K Interface—Hardware-Buffer Swap (Sheet 1)

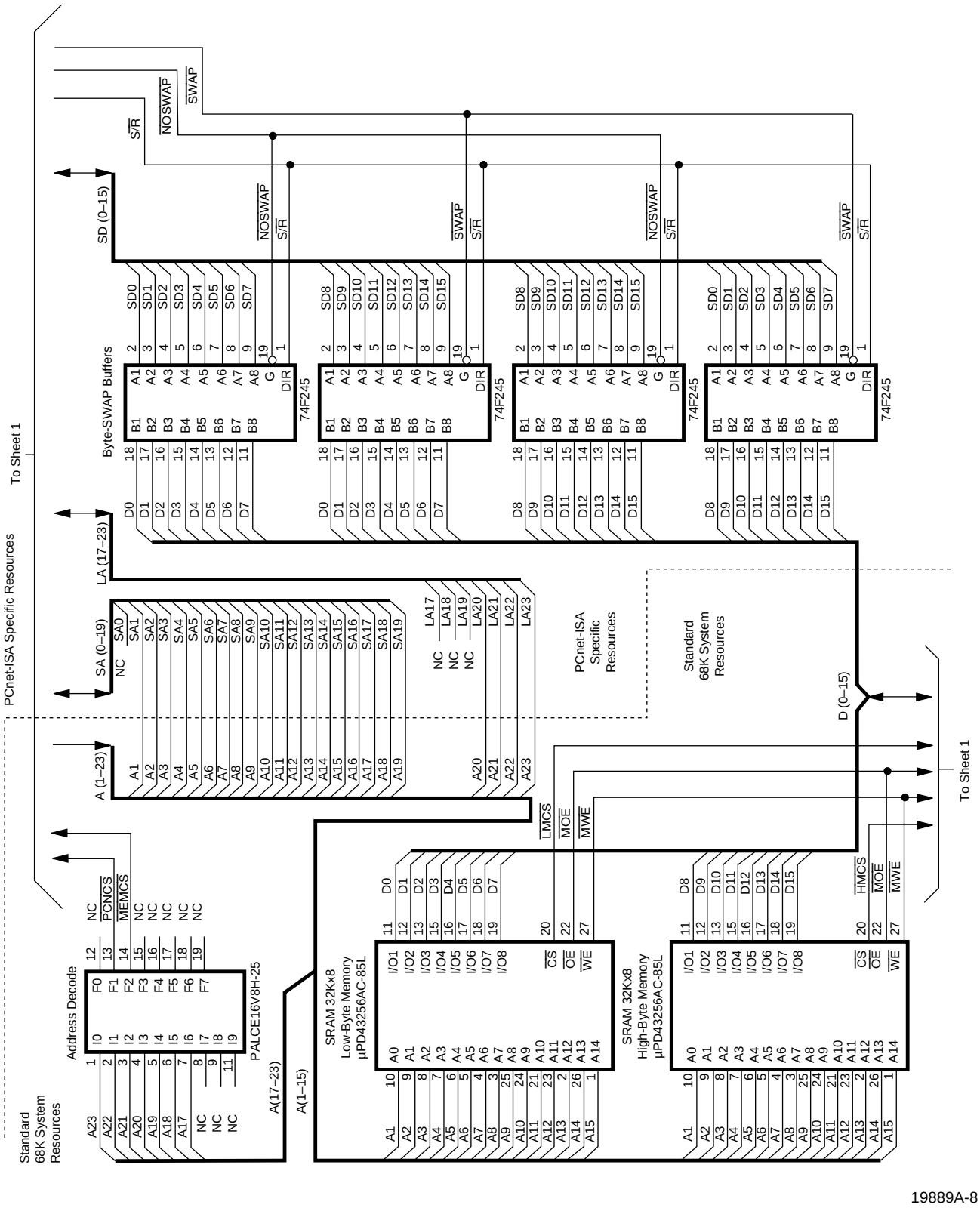


Figure 8. PCnet-ISA-to-68K interface—Hardware-Buffer Swap (Sheet 2)

```
TITLE 68K-to-PCnet-ISA Interface, with Hardware-Buffer Swap
PATTERN
REVISION 0
AUTHOR Mike Keith, Field Applications Engineer
COMPANY Advanced Micro Devices
DATE  2/17/93
CHIP _68K_ISA_B MACH110
;———————————————— PIN Declarations ————————————
PIN 2          ZDTACK_        COMBINATORIAL    ; OUTPUT
PIN 8          A0                              ; INPUT
PIN 15         A16                             ; INPUT
PIN 11         R_W_                            ; INPUT
PIN 13         AS_                             ; INPUT
PIN 33         PCNCS_                          ; INPUT
PIN 17         MEMR_                           ; INPUT
PIN 9          MEMW_                           ; INPUT
PIN 32         MEMCS_                          ; INPUT
PIN 18         UDS_                            ; INPUT
PIN 19         LDS_                            ; INPUT
PIN 20         SBHE_                           ; INPUT
PIN 24         NOSWAP_        COMBINATORIAL    ; OUTPUT
PIN 37         SWAP_          COMBINATORIAL    ; OUTPUT
PIN 29         S_R_           COMBINATORIAL    ; OUTPUT
PIN 4          MWE_           COMBINATORIAL    ; OUTPUT
PIN 27         MOE_           COMBINATORIAL    ; OUTPUT
PIN 31         LMCS_          COMBINATORIAL    ; OUTPUT
PIN 30         HMCS_          COMBINATORIAL    ; OUTPUT
PIN 28         IOWS_          COMBINATORIAL    ; OUTPUT
PIN 25         IORS_          COMBINATORIAL    ; OUTPUT
PIN 35         CLK                             ; INPUT
PIN 14         BG_                             ; INPUT
PIN 10         DRQ                             ; INPUT
PIN 16         IOCHRDY                         ; INPUT
PIN 21         FC0                             ; INPUT
PIN 38         FC1                             ; INPUT
PIN 43         FC2                             ; INPUT
PIN 6          VPA_           COMBINATORIAL    ; OUTPUT
PIN 7          DTACK_         REGISTERED       ; OUTPUT
NODE 8         Q0             REGISTERED       ; FEEDBACK
NODE 28        BRS_           REGISTERED       ; FEEDBACK
NODE 30        ML2_           REGISTERED       ; FEEDBACK
```

Figure 9. MACH110 Design File—68000-to-PCnet-ISA with Hardware-Buffer Swap

```
NODE 29              ML1_              REGISTERED        ; FEEDBACK
PIN 3                DACK_             COMBINATORIAL     ; OUTPUT
PIN 36               BGACK_            COMBINATORIAL     ; OUTPUT
PIN 5                BR_               COMBINATORIAL     ; OUTPUT
NODE 1 GLOBAL
EQUATIONS
;————————————————- Bus master exchange:
DACK_          =  /(AS_ *
                   /(BG_ * BGACK_) )
BGACK_         =  (/DRQ + DACK_) + ML2_
ML1_           := /DRQ + DACK_
ML2_           := ML1_
BRS_           := ML2_
BR_            =  /(BRS_ * DRQ)
;————————————————- Wait state control:
Q0             := AS_ + PCNCS_ + /DTACK_
DTACK_         := Q0 + AS_
Q0.RSTF        =  /IOCHRDY
DTACK_.RSTF    =  /IOCHRDY
ZDTACK_        =  GND                ;if needed: tri-state form of DTACK_
ZDTACK_.TRST   =  /DTACK_             ;if needed: tri-state form of DTACK_
;————————————————- System memory control:
MOE_           =  MEMCS_ + (MEMR_ * (/R_W_ + /DACK_))
MWE_           =  MEMCS_ + (MEMW_ * R_W_)
HMCS_          =  MEMCS_ + (UDS_ * (A0 + A16 + DACK_) * (SBHE_ + /A16 + DACK_))
LMCS_          =  MEMCS_ + (LDS_ * (A0 + /A16 + DACK_) * (SBHE_ + A16 + DACK_))
;————————————————- Hardware swap-buffer control:
NOSWAP_        = ((PCNCS_ +/R_W_ + (UDS_ * LDS_))      ; 68k reads pcnet
                  * (PCNCS_ + R_W_)                    ; 68k writes pcnet
                  * (MEMCS_ + DACK_ + /A16 + MEMR_)    ; pcnet reads noswap mem
                  * (MEMCS_ + DACK_ + /A16 + MWE_))    ; pcnet writes noswap mem
SWAP_          =  /NOSWAP_ +
                  ((MEMCS_ + DACK_ + A16 + MEMR_)      ; pcnet reads swap mem
                  * (MEMCS_ + DACK_ + A16 + MWE_))     ; pcnet writes swap mem
S_R_           =  (/PCNCS_ * R_W_ * (/UDS_ + /LDS_) )  ; 68k reads pcnet
                  + (/MEMCS_ * /DACK_ * MEMR_)   ; pcnet writes mem
;————————————————- Misc control:
VPA_           =  /(FC0 * FC1 * FC2 * /AS_)            ; 68K autovector interrupts
IORS_          =  /R_W_ + PCNCS_ + AS_
IOWS_          =  R_W_ + PCNCS_ + AS_
```

Figure 9. MACH110 Design File—68000-to-PCnet-ISA with Hardware-Buffer Swap (Continued)

```
;————————————————- Misc:
ML1_.CLKF          =  CLK
ML2_.CLKF          =  CLK
BRS_.CLKF          =  CLK
Q0.CLKF            =  CLK
DTACK_.CLKF        =  CLK
ML1_.RSTF          =  GND
ML2_.RSTF          =  GND
BRS_.RSTF          =  GND
GLOBAL.SETF        =  GND
```

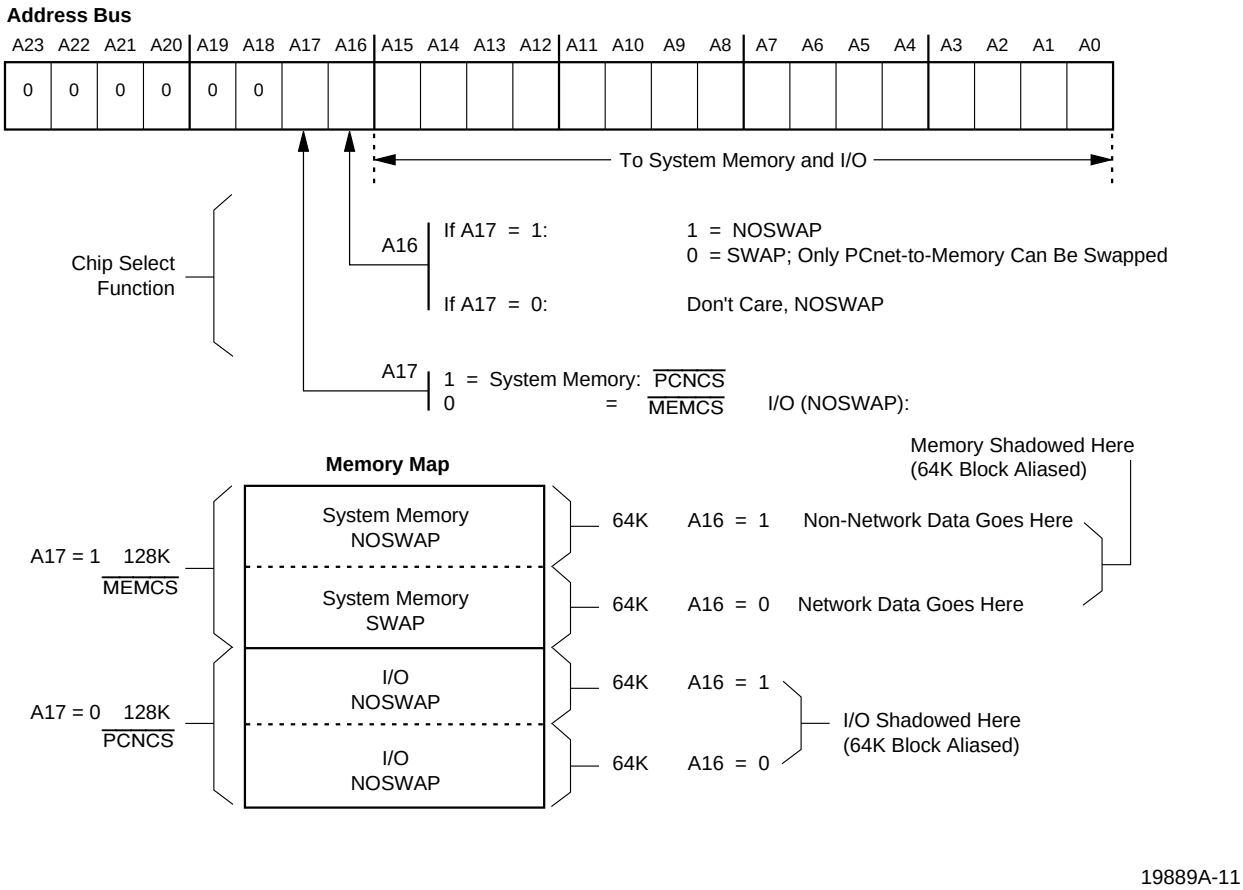
19889A-9

Figure 9. MACH110 Design File—68000-to-PCnet-ISA with Hardware-Buffer Swap (Continued)

```
TITLE   Address Decode—68K-to-PCnet i/f
PATTERN
REVISION 0
AUTHOR Mike Keith, Field Applications Engineer
COMPANY Advanced Micro Devices
DATE  2/17/93
CHIP _decode PALCE16V8
;———————————————————— PIN Declarations ————————————
PIN 1   A23                                ; INPUT
PIN 2   A22                                ; INPUT
PIN 3   A21                                ; INPUT
PIN 4   A20                                ; INPUT
PIN 5   A19                                ; INPUT
PIN 6   A18                                ; INPUT
PIN 7   A17                                ; INPUT
PIN 13  PCNCS_   COMBINATORIAL             ; OUTPUT
PIN 14  MEMCS_   COMBINATORIAL             ; OUTPUT
EQUATIONS
MEMCS_          =  /A17 + A23+A22+A21+A20+A19+A18
PCNCS_          =  A17 + A23+A22+A21+A20+A19+A18
```

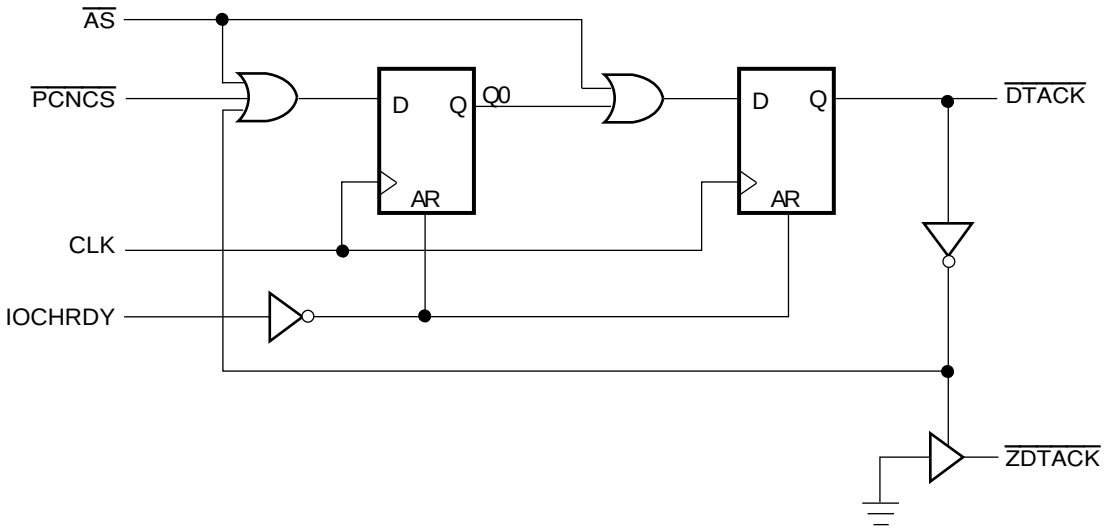
19889A-10

Figure 10. 16V8 Design File—Address Decode



19889A-11

Figure 11. Memory Map Example (Applies to Hardware-Buffer Swap Only)



19889A-12

Figure 12. Wait State Control Unit

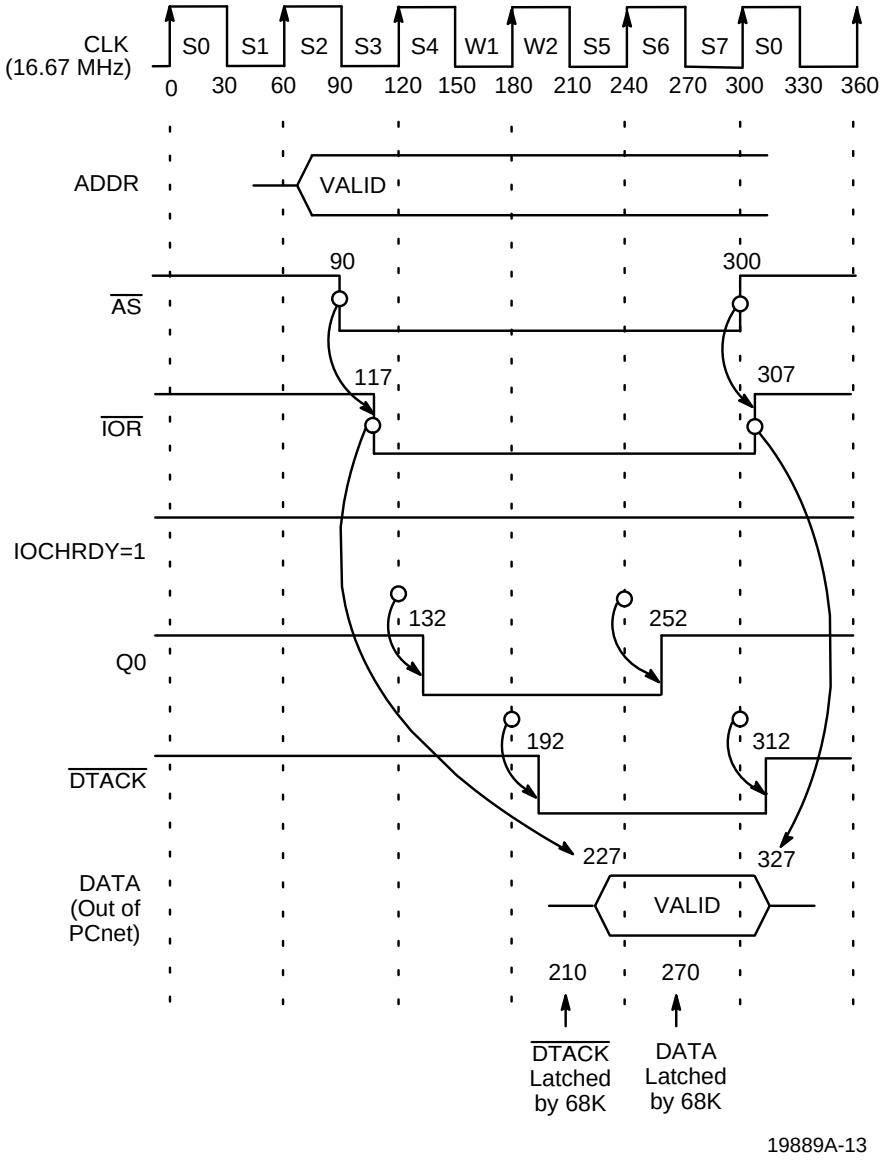


Figure 13a. Wait State Control Timing Diagram (68K Reads PCnet, IOCHRDY=1)

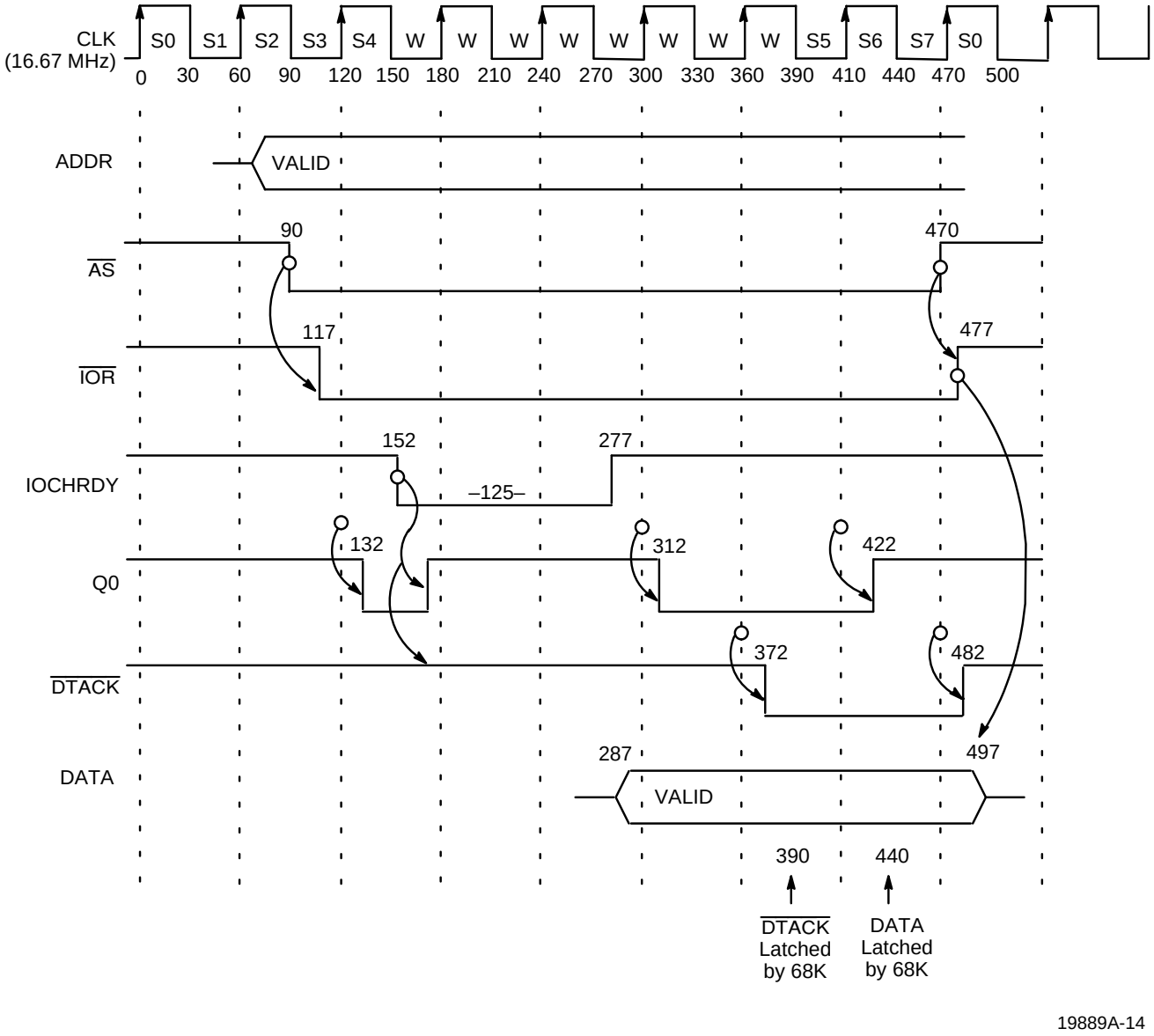


Figure 13b. Wait State Control Timing Diagram (68K Reads PCnet with waits inserted via IOCHRDY)

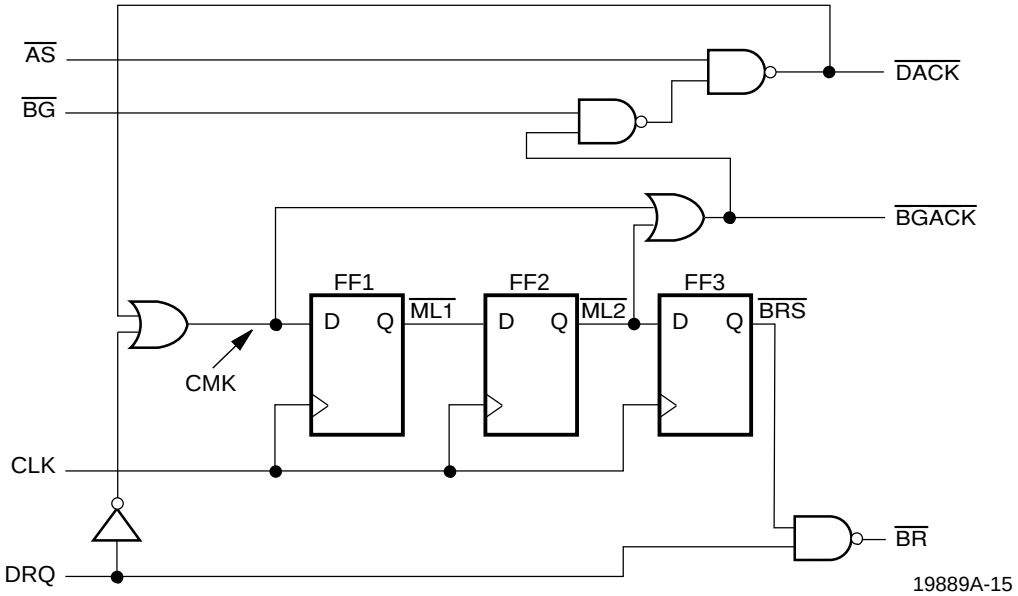


Figure 14. Bus Master Exchange Circuit

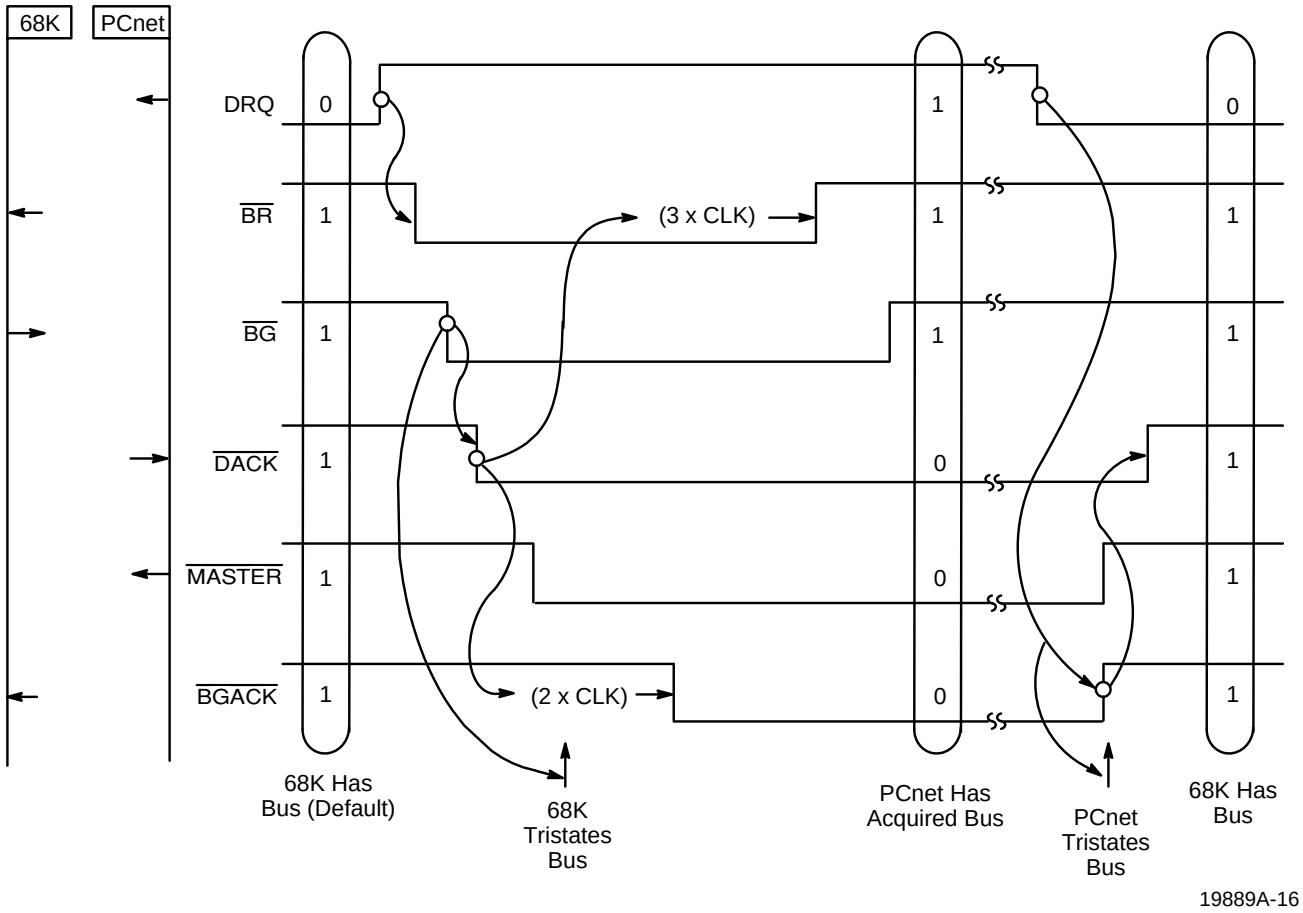


Figure 15. Bus Master Exchange Timing Diagram

II. USING THE PCnet-ISA CONTROLLER WITH A BIG ENDIAN PROCESSOR

The PCnet-ISA controller (Am79C960) is an excellent general purpose Ethernet controller. One of its strongest features is buffer management. Its buffer management is very similar to the buffer management of the Am7990 LANCE (Local Area Network Controller for Ethernet) [and the new Am79C90 C-LANCE (CMOS version of the Am7990 LANCE)], another industry standard Ethernet controller chip from AMD. Unlike the LANCE, the PCnet-ISA controller has separate transmit and receive FIFOs for Ethernet packets. The transmit FIFO is 136 bytes deep, and the receive FIFO is 128 bytes deep. The PCnet-ISA controller also has an onboard 10BASE-T transceiver as well as an onboard Manchester encoder/decoder which can interface to a 10BASE-T, 10BASE2 or 10BASE5 MAUs (Media Attachment Unit). Other features of the PCnet-ISA controller include: External Address Detect Interface, Dynamic Transmit FCS Generation on a packet by packet basis, Power Down Modes and JTAG support.

The bus interface of the PCnet-ISA controller was designed to be compatible to the ISA PC bus. This bus architecture is very similar to the buses of iAPX processors. The PCnet-ISA controller has the capability to tune the read/write timing on its ISA bus interface. This allows the PCnet-ISA controller to interface to many clone PCs as well as provide interface flexibility to a wide range of processors. The PCnet-ISA controller may tune its read/write pulse width between 50 and 750 ns, which gives the PCnet-ISA controller a maximum throughput rate of 150 ns/word (with 50 ns read/write pulse width).

The PCnet-ISA controller is very easy to interface to iAPX processors. The PCnet-ISA controller can also be interfaced to Motorola 68000 class processors. The reader can reference the LANCE (Am7990) Technical Manual (contained in the ***1992 World Network Data Book/Handbook PID# 14287B***) for details of interfacing a 68000 to a LANCE. While the LANCE and the PCnet-ISA controller do not have identical interfaces, they are somewhat similar, and this application note will give the user a feel for how to treat interface signals.

(The user can also refer to the 1994 Ethernet/IEEE 802.3 Family Data Book/Handbook, PID# 14287C for a complete description of the Am79C90 C-LANCE; CMOS, pin-compatible version of the Am7990 LANCE.)

(NOTE: The reader should also refer to the application note titled “Interfacing the Motorola M68000 Microprocessor to the PCnet-ISA Am79C960 Ethernet Controller” by Mike Keith Senior FAE, for additional information on byte swapping techniques to those discussed in this application note.)

There are issues relating to byte swapping high and low order bytes when interfacing the PCnet-ISA controller to a 68000 class of processor, or any processor that is a big endian processor. The rest of this application note will propose several solutions to dealing with byte swapping.

Big Endian Versus Little Endian

The PCnet-ISA controller device is a little Endian device and only supports that byte orientation, unlike the LANCE which supports both little and big endian.

A big endian device associates lower order addresses with higher order bytes (reference Figure 16b). A 68000 has a big endian format. A little endian device associates lower order addresses with lower order bytes (reference Figure 16a). iAPX processors are little endian devices. Note, some devices like the AMD 29000 family can support either byte orientation.

The relation of address bits for byte lane accessing is different for big versus little endian devices.

For big endian devices: (lower addresses are associated with higher order bytes, refer to Figure 16b):

A<4:0> = <00000>, accesses byte 0, bits<15:8>

A<4:0> = <00001>, accesses byte 1, bits<7:0>

For little endian devices (lower addresses are associated with lower order bytes, refer to Figure 16a):

A<4:0> = <00000>, accesses byte 0, bits<7:0>

A<4:0> = <00001>, accesses byte 1, bits<15:8>

Endian Issues for the PCnet-ISA Controller

The PCnet-ISA controller is a little endian device. There is no byte orientation problem when using the PCnet-ISA controller with a little endian processor. To understand the issues regarding mixing a the PCnet-ISA controller with a big endian processor, it is necessary to examine the types of data transfers with which the PCnet-ISA controller can be involved.

The PCnet-ISA controller deals with two classes of transfers (reference Figure 17: Basic System Block Diagram).

1) Slave mode reads and writes. For these types of transfers, the PCnet-ISA controller is a slave to the processor, reacting to the processors control signals. In these types of transfers, the processor is accessing the PCnet-ISA controller internal registers for the purpose of configuring or reading the status of the PCnet-ISA controller.
2) Master mode reads and writes. For these types of accesses, the PCnet-ISA controller is in control of its interface bus and is controlling

reads and writes to system memory. The PCnet-ISA controller is performing DMAs to and from memory during these transfers. There are two types of transfers within the Master mode class:

a) Ring Descriptor Transfers: The PCnet-ISA controller is dealing with descriptor ring data. In this case, the PCnet-ISA controller is moving data between its internal registers and external memory. Descriptor rings are data structures in memory which are associated with the Ethernet transmit and receive packets. They define the locations of transmit/receive buffers and contain status relating to transmit/receive packets. The PCnet-ISA controller and the processor both access the Ring Descriptors.

b) Transmit/Receive FIFO Data Transfers: The PCnet-ISA controller is dealing with Ethernet packet data. The PCnet-ISA controller is either moving data to be transmitted (onto Ethernet) from memory into its transmit FIFO, or moving data received from the Ethernet from its receive FIFO to memory. The PCnet-ISA controller and the processor both access the transmit/receive data buffer memory.

For all types of transfers that involve the PCnet-ISA controller internal registers, the endian-ness of the processor versus the PCnet-ISA controller should not be an issue. (This would include Slave mode read/writes and Master mode descriptor ring transfers.) Effectively, for all these types of transfers, the PCnet-ISA controller does not care about byte ordering as implied by the endian-ness of a processor. The PCnet-ISA controller only cares that the specification of its bit definitions for internal registers be met.

For example, CSR0 (Control Status Register 0 [reference Figure 18a]) defines certain status bits of the PCnet-ISA controller. Notice that the definition of the register is on a per bit basis and this bit definition must be adhered to by the processor interfacing to the PCnet-ISA controller.

Relating to Descriptor Rings, Figure 18b shows the Transmit Descriptor Ring 1 (TMD1). Note that similar to the status registers, the Descriptor Rings have bit level definitions that a processor must adhered to.

For both these examples, the processor must adhere to the definition of bits as required by the PCnet-ISA controller to have the PCnet-ISA controller operate correctly. Therefore, the driver code dealing with the PCnet-ISA controller must store constants, generate bit patterns, and output and input data accordingly.

The one area remaining is Master Mode Transfers that transfer transmit/receive data to and from the PCnet-ISA controller internal FIFOs. Proper byte orientation is imperative for these transfers. There is a relationship between which byte of a word is taken from the transmit FIFO and sent onto the Ethernet first. Assuming the PCnet-ISA controller performs an aligned word read access from memory (with A0 = 0), the data on data bits <7:0> will be loaded into location of the PCnet-ISA controller's transmit FIFO. Bits<15:8> will be loaded into location n+1 of the FIFO. FIFO location n will be transmitted first. There will therefore be a problem when interfacing to a big endian processor because the serial data stream transmitted onto the Ethernet will be out of order. Similarly, receive data will be out of order.

Possible Solutions

The issues of interfacing the PCnet-ISA controller to a big endian processor may be solved with either software or hardware, or a combination of hardware and software. As described above, Master mode transmit and receive transfers care about endian-ness, whereas other transfer types don't care. Three solutions will be proposed. Solution #1 utilizes software only. Solution #2 reverse wires the data buses between the PCnet-ISA controller and the big Endian processor. There is some software help required for this solution. Solution #3, the most hardware intense, adds byte swap logic into the path between the PCnet-ISA controller and the big endian processor.

Solution I. Software Only

This solution proposes that the data in the PCnet-ISA controller transmit and receive buffers (in external memory) is byte swapped by the processor. Code must be added to byte swap the data in these buffers.

For transmit buffers, the transmit data is usually moved from some application area to the PCnet-ISA controller transmit buffer memory space, then transmitted onto the Ethernet by the PCnet-ISA controller. When this data movement is being performed bytes can be swapped as the data is being moved by the processor. This will position the bytes in the correct order for serialization onto the Ethernet. For receive buffers, the received data is usually moved from the PCnet-ISA controller receive buffers in memory to an application area of memory. At the time this data movement occurs bytes can then be swapped by the processor so that data is now byte ordered as the application expects.

Performance Impact of Solution 1:

Additional code to swap bytes need only be added in the driver code for the PCnet-ISA controller. This change should not affect upper layer software which one might want to be portable to different systems. The addition of instruction(s) to perform the byte swap should not add much time to the transmit or receive code. Some processors have single cycle byte swap instructions (for example, 68040). However, each design should consider the performance impact of this additional

code based on the specifics of their design (processor speed and time required to do byte swap). This solution does burden every 16 bit word that is transmitted or received with at least one additional line of code that needs to be executed.

Solution II. Hardware/Software (Reverse Data Bus Wiring and Software Workarounds)

Another approach to the problem is to take care of the Master mode data buffer transfers by cross wiring the busses between the PCnet-ISA controller and the processor/memory (reference Figure 19). The byte swapping required for the PCnet-ISA controller Master mode transmit/receive buffer accesses is now accomplished by a hardwired swapping of high order and low order bytes. The processor will read and write bytes in memory in its natural way, big endian. The PCnet-ISA controller will read and write bytes in memory in a swapped order from the processor which will make the positioning look like little endian for the PCnet-ISA controller.

Now the problem of Slave Mode and Master Mode ring buffer accesses must be addressed. One approach to the problem is to perform a byte swap in software for the PCnet-ISA controller slave and ring buffer accesses (performed by the processor). This method adds some software delay for the PCnet-ISA controller register and ring buffer accesses.

Another approach could be to use software tools to get around the problem. The PCnet-ISA controller does not care about byte ordering for slave or ring buffer accesses. It only requires that its bit definition be adhered to. By cross wiring the buses, we have tied bit 0 of the processor to bit 8 of the PCnet-ISA controller, and software must take this into account. For this issue, one can use the facilitates of a high level language compiler or assembler to relate mnemonics to bit positions and make this transparent to the programmer. Using software tools to make the bit swapping transparent should not increase software overhead when dealing with the PCnet-ISA controller registers or the ring buffers.

Performance Impact of Solution 2:

The performance impact of solution #2 will probably be less than the impact of the Software Only Solution, solution #1. Access to transmit/receive data buffers will be performed more often than the PCnet-ISA controller resister and Ring Buffer accesses. Byte swap code added to deal with transmit/receive data will be executed more often than byte swap code dealing with Ring buffers/the PCnet-ISA controller registers.

Beyond that, the transfer types that must be addressed in software with this solution (ring buffer accesses and internal registers) can be handled by software tools rather than performing byte swapping in software which adds cycles to code.

Solution III. Hardware Only

As stated previously, the only transfers that care about byte orientation are transmit/receive data buffer transfers dealing with the PCnet-ISA controller controller's FIFOs. To address this situation in hardware, it is necessary to add byte swap logic between the PCnet-ISA controller and memory. This byte swap logic is enabled only when the PCnet-ISA controller is performing Master mode accesses to data memory (reference Figure 20). All other types of transfers as previously discussed need not be byte swapped. (Note: Refer to previous section entitled "Interfacing the Motorola M68000 Microprocessor to the PCnet-ISA Am79C960 Ethernet Controller" by Mike Keith.)

Figure 20 shows logically how the byte swapping hardware should work. Referencing to Figure 20; The address decoder determines when the PCnet-ISA controller is accessing transmit/receive buffer memory. The address decode logic generates an ENABLE to the Byte Swap Logic. When ENABLE is asserted, the Byte Swap Logic will swap bytes to/from the PCnet-ISA controller. The Byte Swap Logic must also know the direction of transfer (read or write). Figure 20 also shows logically, that when bytes are being swapped, that Byte Write enables must also be swapped to the memory banks, such that a PCnet-ISA controller byte write modifies the proper memory bank.

There is a system level implication in that the Address Decoder logic must know the address space of the PCnet-ISA controller's transmit/receive buffers since this is the only time the byte swap logic should be enabled. There is some implication in that this address area for transmit/receive buffers must be fixed and remain fixed.

Some hardware implementation details relating to this solution are discussed in the following section entitled "Implementation Discussion for Solution #3".

Performance Impact of Solution 3:

The byte swap hardware adds delay into the path between the PCnet-ISA controller and the big endian processor, and the PCnet-ISA controller and memory, for all types of the PCnet-ISA controller accesses. As long as this addition delay does not add wait states when: 1) the processor is accessing the PCnet-ISA controller registers, or 2) when the PCnet-ISA controller is accessing memory, then this solution has no performance impact.

Conclusions

The PCnet-ISA Ethernet controller is a very flexible and powerful controller for Ethernet. While it has an interface optimized for the ISA bus, it can be easily used with iAPX processors. The PCnet-ISA controller can also be used with other processors such as Motorola 68K class processors.

One consideration for interfacing to various processors is the question whether the processor is big or little endian. The PCnet-ISA controller is a little endian device. When interfacing to a big endian processor, there are several solutions possible in both hardware and software.

The software solutions are fairly reasonable and portable. Many of the software issues can be addressed using facilities available in high level language compilers or assemblers. The software overhead for performing byte swapping where necessary will probably not hinder performance. The system designer should determine performance impact based on processor speed and instruction set.

Solution # 1 requires no additional hardware nor add any additional logic delay in the data paths of the PCnet-ISA controller.

Solution # 2 will probably have less performance impact than Solution # 1. In each case, the impact of ease of understanding and maintenance from the software viewpoint should also be considered.

Solution #3 will add some logic to the design. In some cases, some of the logic required for address decoding may be combined with existing logic. Additional delays may be incurred by adding byte swap logic, depending on how the byte swap logic is implemented. For those designs where the PCnet-ISA controller already has a set of transceivers, additional byte swap transceivers as suggested in Appendix A add no extra delay to the path between the PCnet-ISA controller and memory.

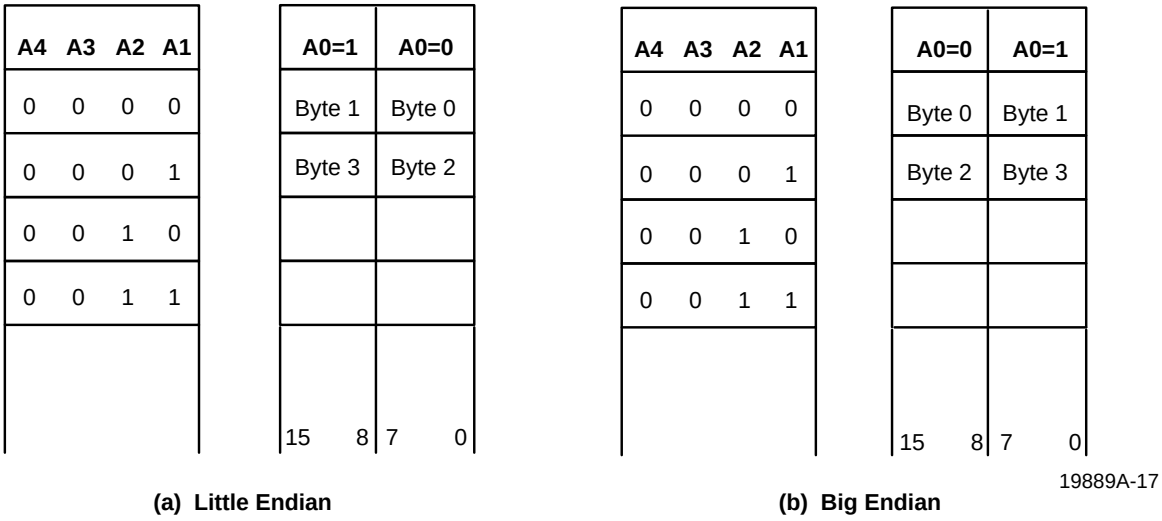


Figure 16. Big Endian/Little Endian Orientation

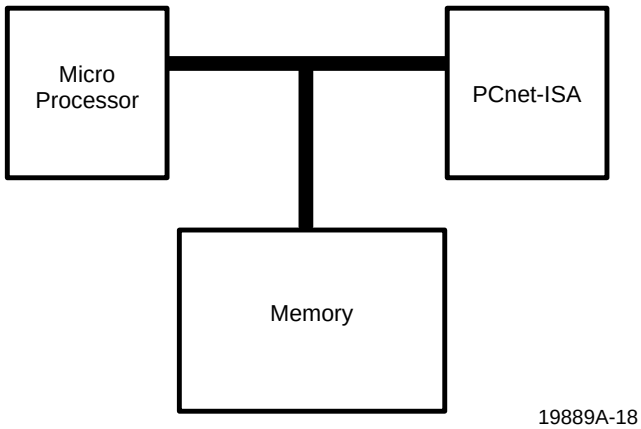


Figure 17. Basic System Block Diagram

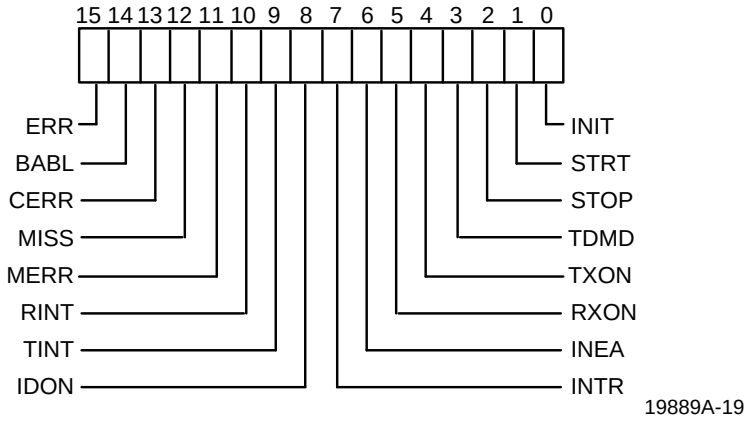


Figure 18a. Example of PCnet-ISA CSR0

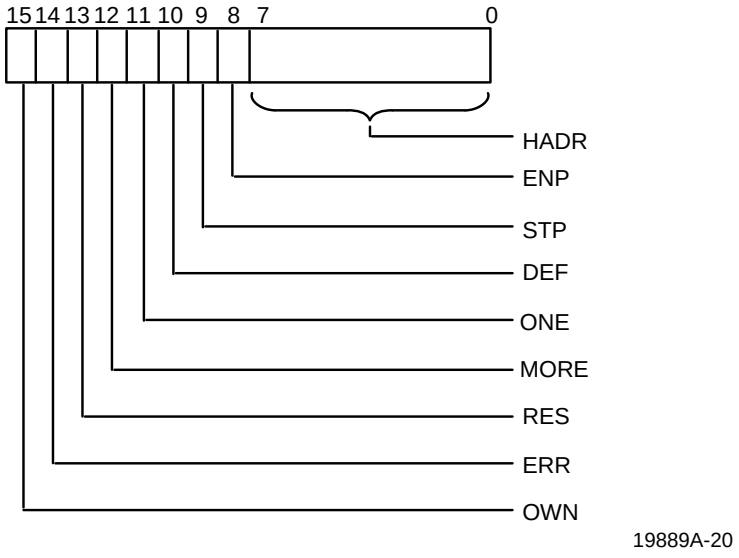


Figure 18b. Example Transmit Descriptor Ring TMD1

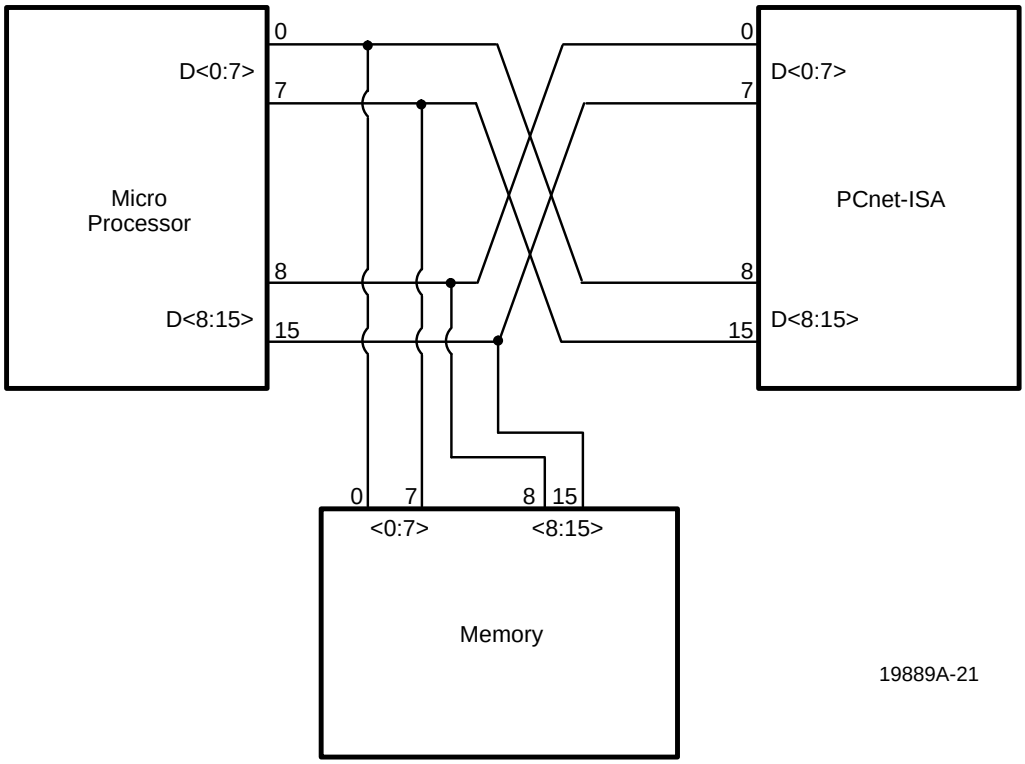
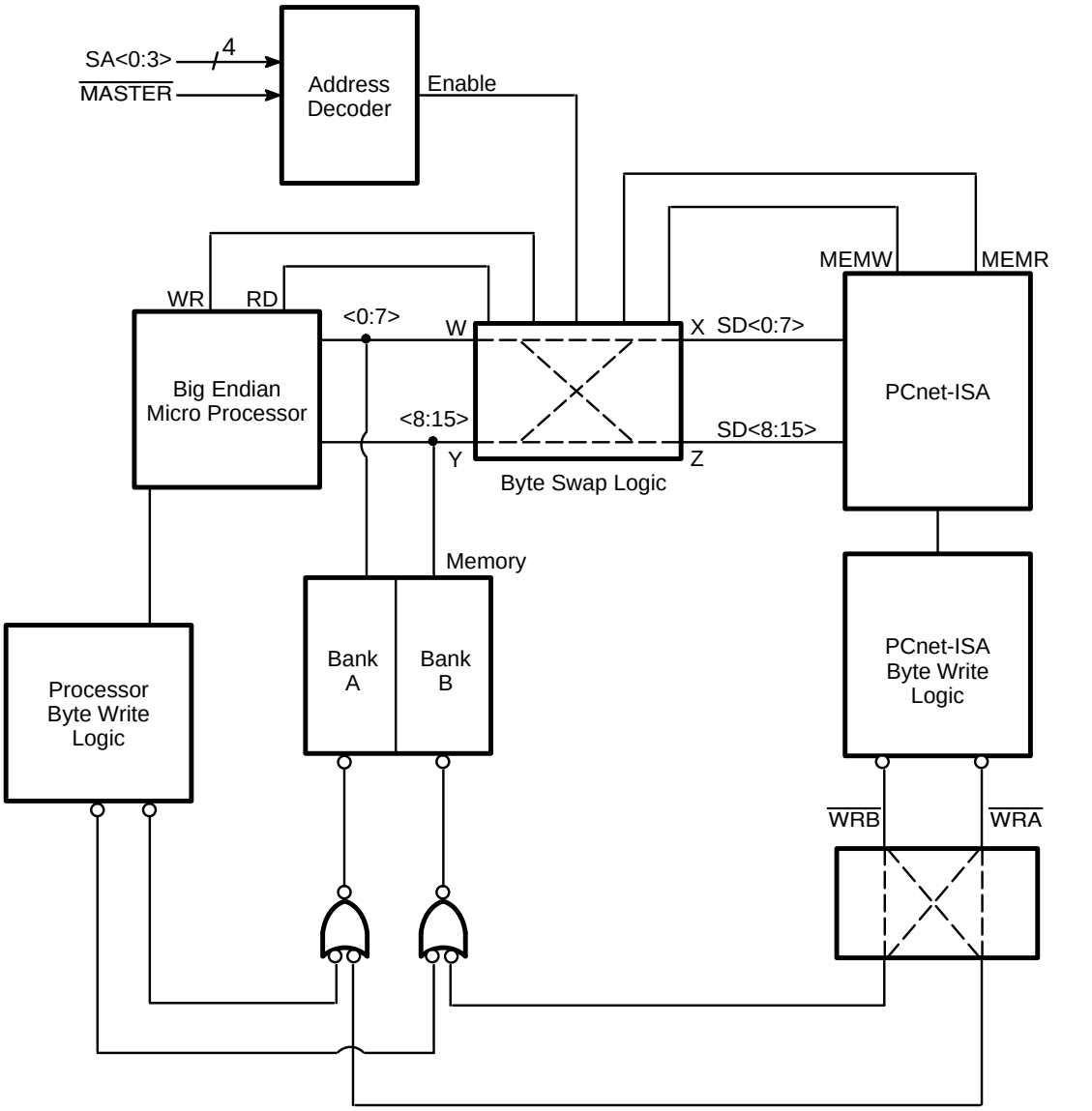


Figure 19. Byte Reverse Wiring



Byte Swap Path Combinations

Enable = 0	W-X, Y-Z
Enable = 1	W-Z, X-Y

Figure 20. Logical Block Diagram for Byte Swap

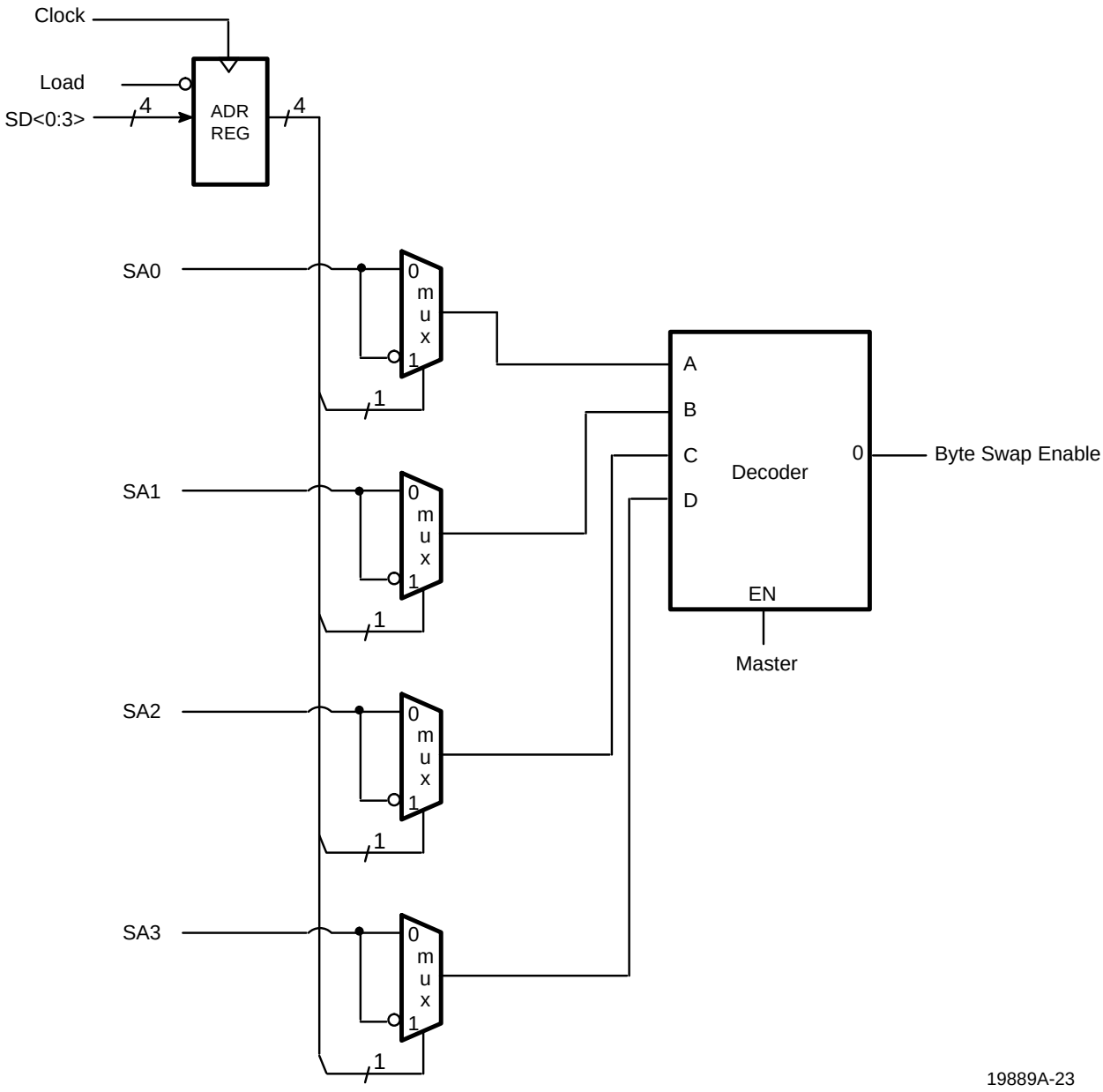


Figure 21a. Logical Implementation for Programmable Transmit/Receive Buffer

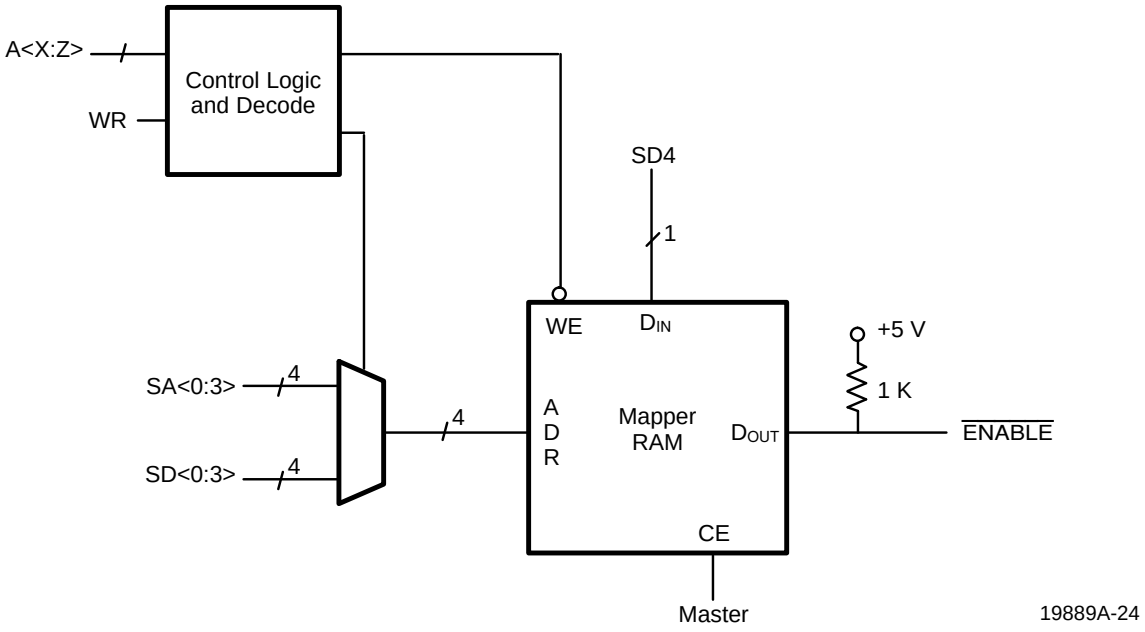


Figure 21b. Mapper RAM

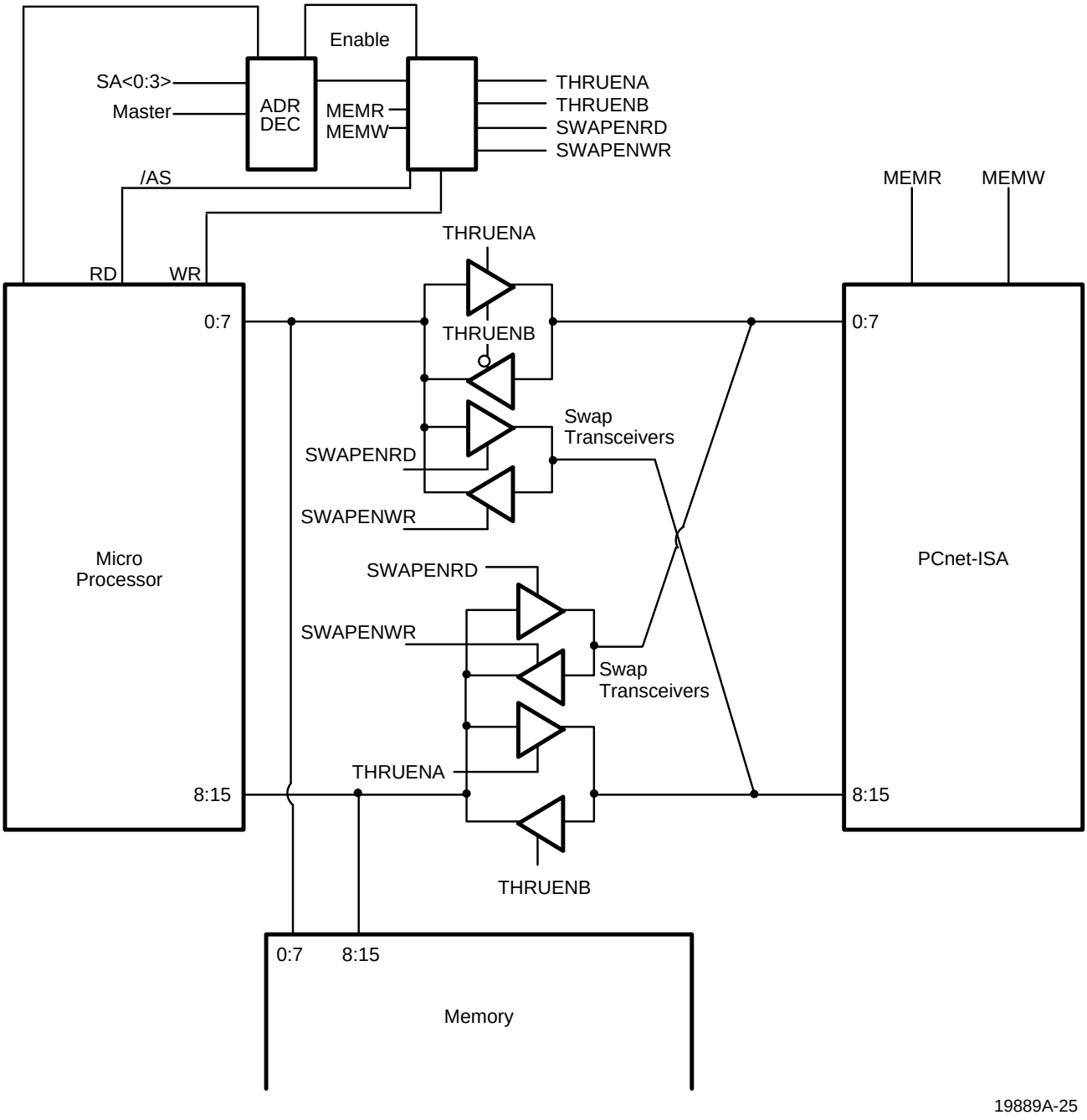


Figure 22. Byte Swap Transceivers

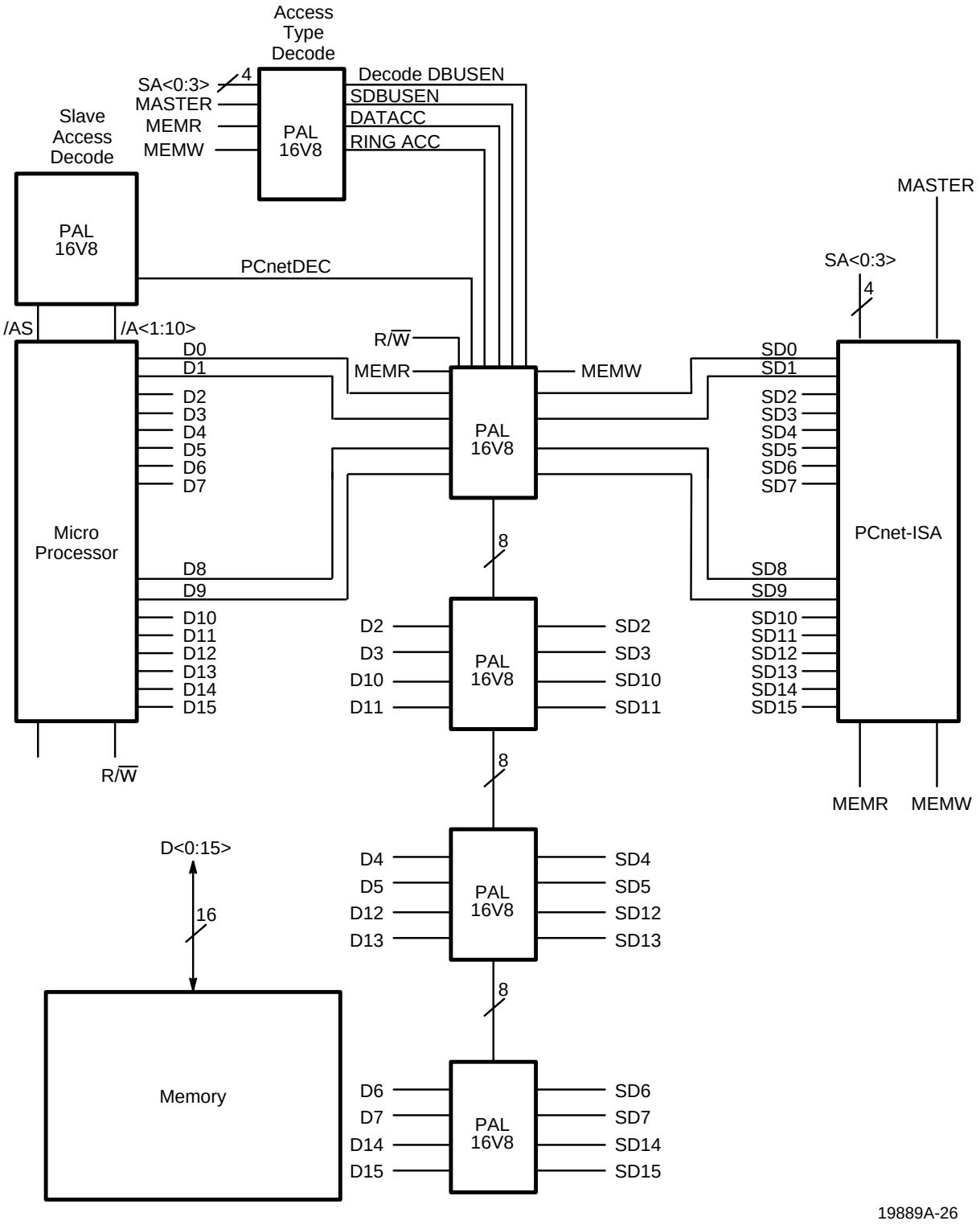
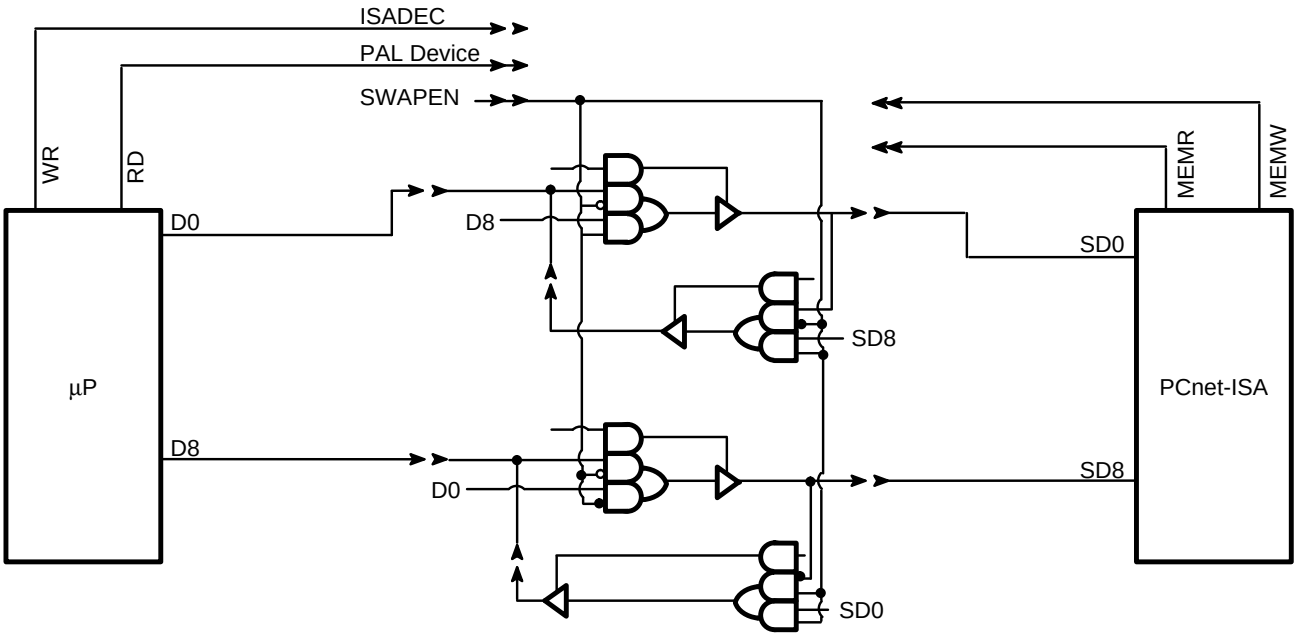


Figure 23a. PAL MUX Byte Swap



Note: *Symbol indicates PAL external pin.*

Example Equations:

SWAPEN = LA20 • LA21 • LA22 • LA23 • MASTER

SD0 = D0 •/SWAPEN
+ D8 • SWAPEN;

19889A-27

SD0.TRST = WR • ISADECODE
+ MEMR • MASTER;

D0 = SD0 • /SWAPEN
+ SD8 • SWAPEN;

DO.TRST = RD • ISADECODE
+ MEMW • MASTER

Figure 23b. Expanded View of Byte Swap (2 bits)

Implementation Discussion for Solution #3 [Hardware Only]

It is probably best to partition the swapping hardware into two separate blocks (reference Figure 20): one for address decode and one for byte swapping. Different options for implementing byte swapping will be discussed later. The address decode logic must decode the address area for buffer memory when the PCnet-ISA controller is a bus master and generate an enable to the byte swap logic. To do this, some number of address bits are required to decode the data ring buffer area of memory. In this example, four upper address bits are being used. The fact that the PCnet-ISA controller is performing a Master mode transfer is known when the signal MASTER is asserted by the PCnet-ISA controller. An ENABLE is then generated and gated with the PCnet-ISA controller read/write command signals, MEMR and MEMW, to drive data in the proper direction to or from memory. The decode logic must also control bus drivers associated with the byte swap logic for all processor reads and writes to the PCnet-ISA controller as well as all the PCnet-ISA controller Master mode transfers.

Since PCnet-ISA controller addresses will set up before read/write commands, it may be better to start the address decode early, that is, not include address decode in the byte swap logic. In this way, the byte swap ENABLE signal can have time to set up before the read/write command is asserted. The logic that controls the byte swap hardware then only need gate the ENABLE signal with MEMR or MEMW, which can be done in a fast PAL or fast gate to minimize delay. Note that in some system designs where the PCnet-ISA

controller has tri-state bus drivers, there may already be decode logic to enable these drivers. The decode enable for the byte swap logic may be shared, for example, in same logic as the existing tri-state enables (this is usually easy if the decode is done using a PAL).

Address Decode Hardware with Fixed Address Transmit/Receive Buffers

For this implementation the decode of the ring buffer area is fixed. The Transmit or Receive Buffer ring buffer area has been pre-defined and its decode fixed in hardware.

In this case (reference Figure 20), we are assuming four upper address bits can be used to decode this area. When an access to this area is detected with the cycle being a PCnet-ISA controller's Master mode access, the byte swap logic will be enabled. Positive logic equations are shown below (assumes that SA<0:3> are <1001> when transmit/receive ring buffers are accessed).

ENABLE = SA0 *$\overline{SA1}$ *$\overline{SA2}$ * SA3 * MASTER.

Address Decode Hardware With Programmable Address Transmit/Receive Buffers

For this implementation, some means of programming the polarity of the address bits seen by the decode logic must be implemented. Figure 21a depicts a logic implementation that would work. The ADR REG is loaded as an access to a peripheral and could share the peripheral address space of the PCnet-ISA controller. The address register is loaded with a bit pattern that will generate an enable when locations of the transmit/receive buffer memory are accessed by the PCnet-ISA controller. For example, assume the decode logic will generate an enable when 1111 is on SA<0:3>, which relates to the address space of transmit/receive data buffer area in memory. The ADR REG would be loaded with 1111 (D<0:3> respectively). If the transmit/receive address space is changed to SA<0:3> = 1010, the ADR REG is loaded with 1010. The mux will now present the correct sense of address bits to generate a decode when SA<0:3> is 1010.

Once the ADR REG is loaded, the decoder can enable the byte swap logic when an access to this area in Master mode is performed by the PCnet-ISA controller. If the system decides to change the memory space of the transmit/receive buffers, the ADR REG is reprogrammed. One issue here is that an extra layer of logic is required. The address delay to decode is a bit longer but, since addresses set up about 55ns before the MEMR or MEMW signal, the extra delay will probably not be a problem for most implementations.

Another implementation uses a RAM to decode addresses (reference Figure 21b). The mapper RAM locations are loaded by system software via the data bus as a peripheral. The Mapper RAM is programmed to generate an ENABLE when MASTER mode transmit/receive buffer accesses are performed. That is, when SA and MASTER indicate a master data buffer access, the mapper RAM is accessed such that Dout generates ENABLE low.

Byte Swap Logic Options

There are several options to perform the byte swapping that will be outlined here.

Byte Swap Transceivers (Reference Figure 22):

This may be the best approach from a timing viewpoint. A set of transceivers are added to swap bytes for the PCnet-ISA controller's access. The address decoder, as previously discussed, provides an enable that then is qualified with MEMR, MEMW or R/W to enable the transceivers in the proper direction. The system design may already have a set of transceivers between the PCnet-ISA controller and memory anyway. In those cases, this scheme does not add any additional delay to the PCnet-ISA controller related transfers since data is traveling through a set of buffers anyway. The design will probably already have tri-state driver enable logic which may be shared with the byte swap enable logic. For this case, the additional logic may add a little overhead, and no additional delay to the PCnet-ISA controller transfers.

Note:

The SWAP and THRU transceivers should be enabled as follows:

1) Assume SA<0:3> = 1111 for a data buffer access.

2) PCnetDEC is a decode of A<0:3> bits denoting that the PCnet-ISA controller registers are being accessed.

```
SWAPENRD = MASTER * SA0 * SA1 * SA2 * SA3 * MEMR          ; PCnet-ISA data
                                                          ; buffer read

SWAPENWR = MASTER * SA0 * SA1 * SA2 * SA3 * MEMW          ; PCnet-ISA data
                                                          ;  buffer  write

THRUENA = (PCnetDEC)*/R/W                                 ; PCnet-ISA slave
        + (MEMR * MASTER)(SA0 +SA1 +SA2 +SA3)             ; write PCnet-ISA
                                                          ;  ring buffer read

THRUENB = (PCnetDEC)*R/W                                  ; PCnet-ISA slave
     + (MEMW * MASTER)(SA0 +SA1 +SA2 +SA3)                ; read PCnet-ISA
                                                          ; ring buffer write
```

Byte Swap PAL Mux (Reference Figures 23):

A rough diagram of the byte swap logic using a PAL mux is shown in Figure 23. The PAL implements a mux, providing byte swapped and non swapped paths, as well as tri-state enables for the uP and the PCnet-ISA controller buses.

Two PALs are used for decoding access information. The SLAVE ACCESS DECODE PAL determines whether the PCnet-ISA controller is being accessed as a slave. The ACCESS TYPE DECODE PAL determines the type of access: PCnet-ISA controller Ring buffer or Data buffer access, etc. The byte swapping is accomplished using one PAL per 4 bits of data.

Note, another programmable logic implementation for byte swapping could be done using AMD MACH devices. Use of these parts could save board space over a PAL implementation.

Sample PAL logic equations are as follows:

***Note**:*
These equations assume that the PCnet-ISA controller is accessing data buffer memory when SA<0:3> = <1111>

Slave Access Decode PAL:
This PAL will generate the signal PCNetDEC. It will use the $\overline{AS}$ signal and A<1:10> to determine if the PCnet-ISA controller is being accessed as a slave. Therefore the decode of the address bits will be whatever peripheral address has been assigned to the PCnet-ISA controller.

Access Type Decode PAL:

```
DBUSEN = MASTER * MEMW                  ; PCnet-ISA memory write
    + PCnetDEC * R/W                    ; PCnet-ISA slave read

SDBUSEN = MASTER * MEMR                 ; PCnet-ISA mem read
    + PCnetDEC *R/W                     ; PCnet-ISA slave write

DATACC = MASTER *SA0*SA*SA2*SA3         ; PCnet-ISA data buffer access

RINGACC = MASTER *SA0                   ; PCnet-ISA
    + MASTER *SA1                       ; Ring
    + MASTER *SA2                       ; Buffer
    + MASTER *SA3                       ; access
```

Sample Byte Swap PAL Equations:

```
SD0 = D8 * DATACC * MEMR                ; PCnet-ISA swapped data buffer read
      + D0 * PCnetDEC *R/W              ; PCnet-ISA slave write
      + D0 * RINGACC * MEMR             ; PCnet-ISA ring buffer read

SD0.TRST = SDBUSEN                      ; tri-state enable for SD bus

D0 = SD8 * DATACC * MEMW                ; PCnet-ISA swapped data buffer write
     + SD0 * PCnetDEC * R/W             ; PCnet-ISA slave read
     + SD0 * RINGACC * MEMW             ; PCnet-ISA ring buffer write

D0.TRST = DBUSEN
```