NBS-DIO48

NuBus Digital I/O Interface

Software Reference Manual

PROM Rev 1.0

This page intentionally left blank

fishcamp engineering
4860 Ontario way
Santa Maria, CA 93455

TEL: (805) 345-2324
FAX: (805) 345-2325

Printed in U.S.A. January, 1998

Limited Warranty

The information provided in this manual is believed to be correct, however fishcamp engineering assumes no responsibility for errors contained within. The software programs are provided "as is" without warranty of any kind, either expressed or implied.

No other warranty is expressed or implied. Fishcamp engineering shall not be liable or responsible for any kind of damages, including direct, indirect, special, incidental, or consequential damages, arising or resulting from its products, the use of its products, or the modification to its products. The warranty set forth above is exclusive and in lieu of all others, oral or written, express or implied.

The information covered in this manual is subject to change without notice.

This page intentionally left blank

Section 1
Introduction

The NBS-DIO48 interface card was designed to have its driver code reside in ROM resident on the card. The driver code delivered with the card contains a set of routines which compliment the hardware capabilities of the interface card. Full control of the hardware is provided with most of the low level details of programming for the NBS-DIO48 interface handled by the routines of the driver.

The driver code conforms to the interface guide-lines set forth by Apple Computer in *Inside Macintosh* for device drivers. All driver routine calls can be made thru the Macintosh device manager thus assuring a high level of compatibility with future releases of the Mac operating system.

Along with the NBS-DIO48 card is included a PASCAL interface file which makes the job of coding software for applications even easier.

This manual documents the software routines of the driver code as well as that of the provided interface files. For further information regarding the 82C55A Programmable Peripheral Interface chip used on the NBS-DIO48 card refer to:

Microsystem Components Handbook

Published by:

Intel

Intel Literature Sales
P.O. Box 58130
Santa Clara, CA 95052-8130

Order Number - 230843

Section 2
Software Overview

Application programs written to take full advantage of the NBS-DIO48 interface card will be written in a hierarchical format. As can be seen in figure 2.1, most I/O calls are made from the application to the PASCAL interface routine (or other language) for the appropriate call. The PASCAL interface routine is the highest level interface provided for the user with the NBS-DIO48 card.

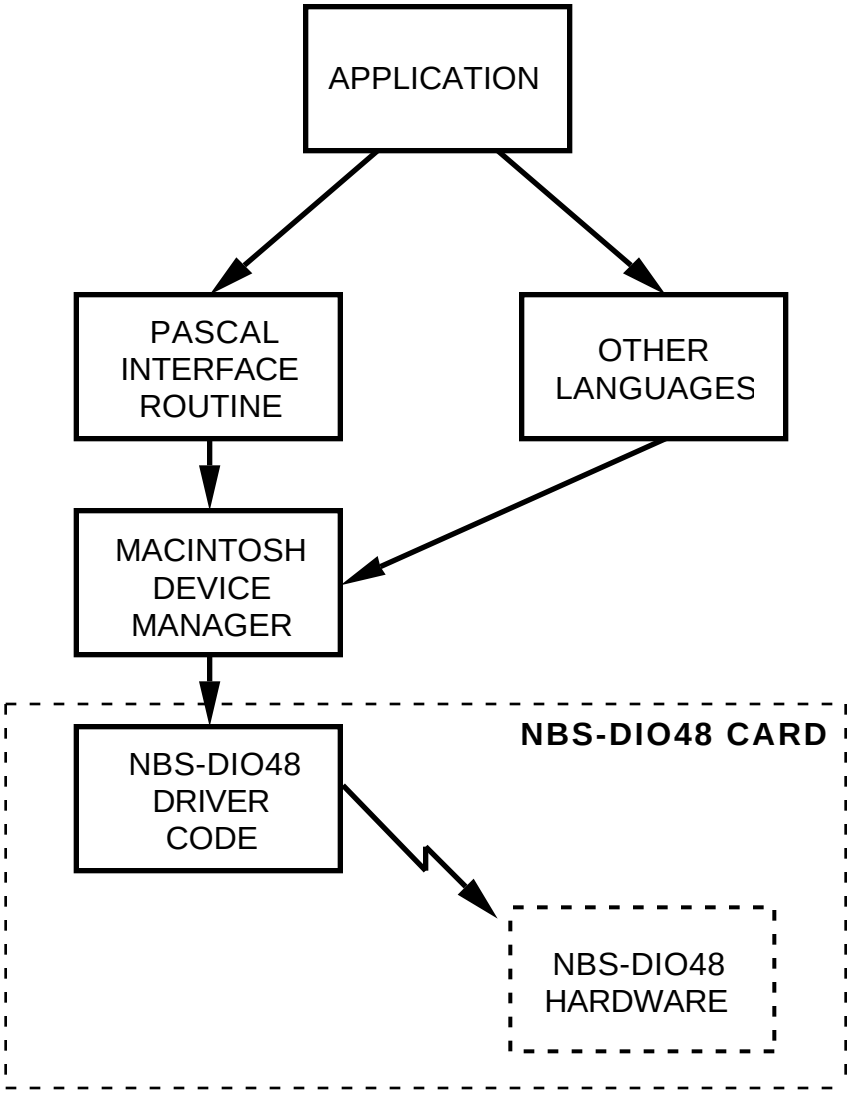


Figure 2.1 - Software Hierarchy

The interface routines take care of accepting/providing parameters from/to the calling program in the most concise and understandable manor. Only the values absolutely necessary for the proper functioning of the respective driver call are included in the pass parameter list of the interface routines. The interface routine further takes care of allocating temporary storage and setting up the parameters for calls to the driver code. These calls are all made thru the Macintosh device manager.

With the exception of the 'dio48Open' and 'dio48Close' routines, all driver calls are made with the device manager 'PBControl' call.

As stated above, the application writer wishing to use the NBS-DIO48 interface card in the execution of his/her program will most likely want the utilize the PASCAL glue routines provided on disk with the board. This is the easiest way of developing software that uses the card because most of the work has been already done for the user by the fishcamp people in writing the code for these routines. Most likely the user would generate routines looking very similar to these if they had not been provided with the card.

On a lower interface level, the routines can be called from any language capable of calling the device manager routines of the Macintosh operating system. The NBS-DIO48 driver code has been written to conform to the guide-lines set forth by Apple Computer for device drivers, and thus is compatible with many other programming languages the user may wish to use. As long as the pass-parameter conventions established by fishcamp engineering for the calls to the driver routines are adhered to, the programmer should have little problem in using the card with other languages. Please refer to the section on driver usage for information on calling the routines thru the Macintosh device manger.

And lastly, the programmer can always by-pass any of the supplied software routines and access the hardware directly. This may be desired when specialized routines peculiar to an application are required or maybe when the user wants to optimize the execution of a certain portion of code. This task will require a significant amount of work to implement, as well as requiring the user to have a thorough understanding of the architecture of the NBS-DIO48 card. Every effort to provide the pertinent information on the design of the card has been done in order to assist the programmer in this task. Please reference the NBS-DIO48 hardware reference manual for information specific to the architecture of the card.

This page intentionally left blank

Section 3
NBS-DIO48 Card Memory Map

The NBS-DIO48 card is an 8-bit interface card with all hardware devices on the card memory mapped to distinct memory locations in the NuBus address space. All data accesses to/from the card are carried out over byte lane three of the NuBus interface. This translates to MC68020 cpu memory accesses from the Mac II with A0 and A1 bits set to 1's. The NBS-DIO48 driver routines take care of selecting the proper byte address in the card slot space.

The card maps the NuBus slot address space into four distinct sections:

- PROM
- Interrupt Mask Logic
- PPI controller registers
- Interrupt Logic

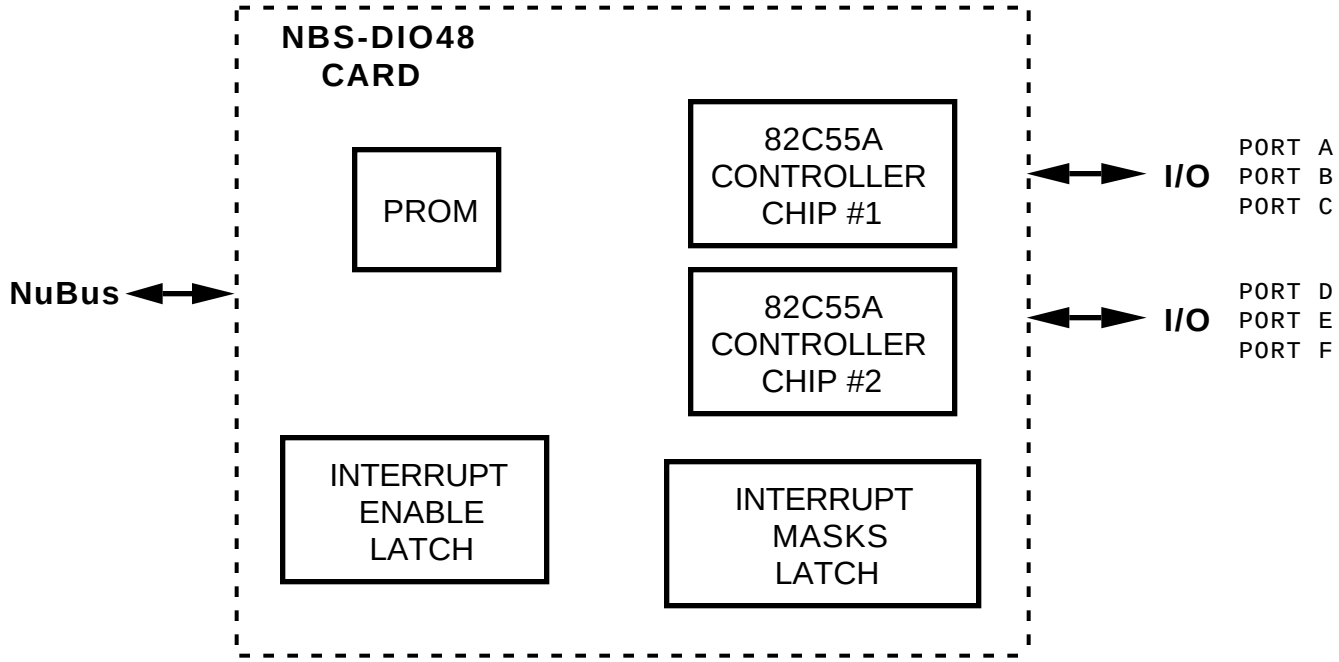


Figure 3.1 - NBS-DIO48 Logical Devices.

The first section occupies the upper portion of the address space allocated to the card in the NuBus slot address space and is used to address the contents of the PROM containing the system driver code for the card. This PROM has an 8K-byte total capacity. The Mac operating reads the driver code from this PROM into system memory at reset time and then executes the code out of system memory from then on. The PROM is usually never accessed after this.

The second memory device on the card is an interrupt mask latch used to enable and disable any of the four possible interrupt sources on the NBS-DIO48 card.

The latch occupies a single byte in the memory map and is a write-only hardware device. Only the four least significant bits of the latch are used on the card. A '1' written to any of these bits of the latch will enable the bit's respective interrupt source such that it will pass the interrupt on to the MAC's processor. A '0' will prevent the interrupt from interrupting the MAC. The interrupt sources are the 82C55A's PC0 and PC3 I/O lines. Refer to the Intel documentation on this device for information on how to use these interrupts. Figure 3.2 shows the mapping of the interrupt mask latch bits to the interrupt sources.

The third and most important block of memory addresses on the card map directly to the I/O registers of the 82C55A controller chips used on the board. There are two 82C55A chips on the NBS-DIO48 card. The chips data bus lines D7-D0 are mapped to the NuBus AD24-AD31 lines respectively. For definitions of the bits of the controller chip's registers consult the Intel documentation on the device.

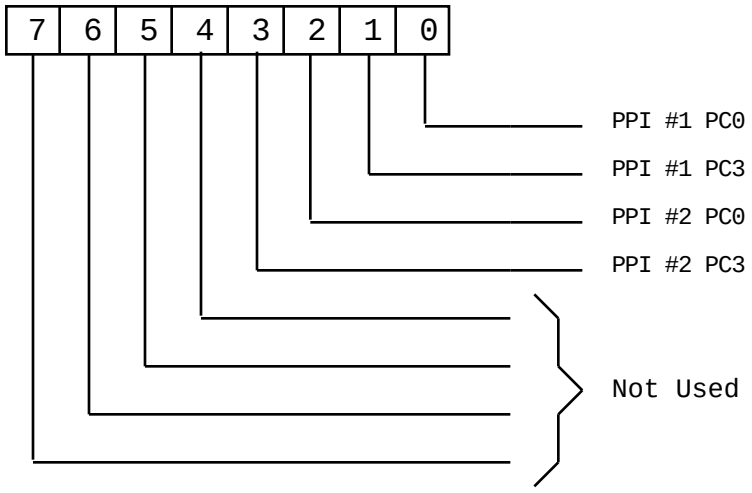
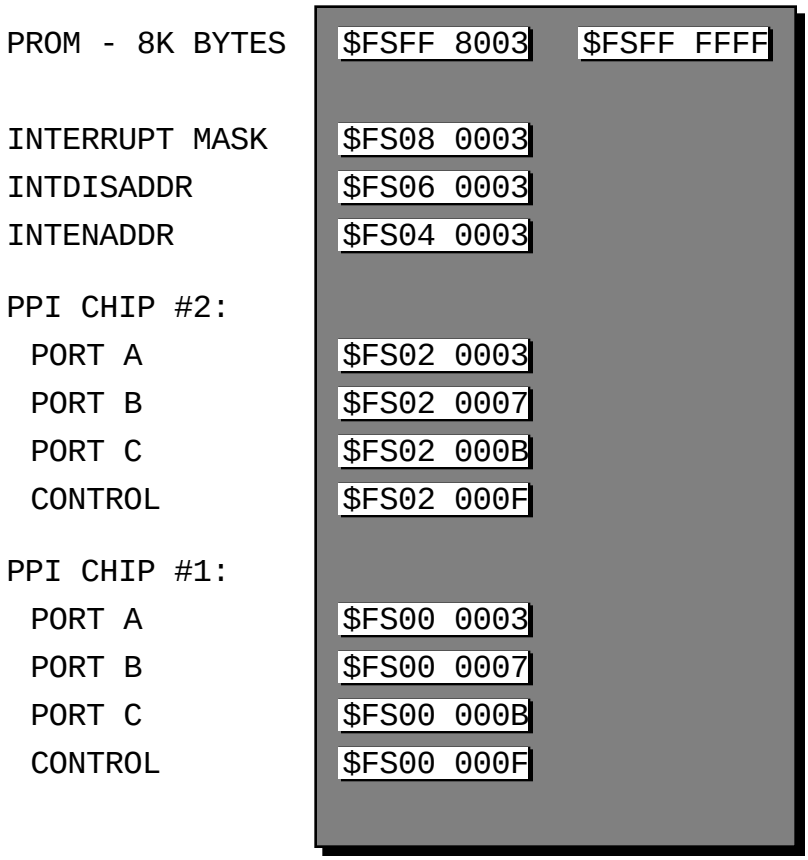


Figure 3.2 - Interrupt Mask Latch Bit Definition.

The last hardware device in the memory map is really two memory locations used in conjunction with each other to set the state of the interrupt enable latch on the card. The latch needs to be set if ANY interrupts from the NBS-DIO48 card are to be sent to the MAC's cpu. The hardware design of the card uses the state of the interrupt enable latch to qualify any interrupts from the 82C55A chips before passing them along to the NuBus 'NMRQ' interface line. Thus, to utilize interrupt operation on the NBS-DIO48 card, the application must first setup the mode properly for the 82C55A controller chip in order to enable the interrupt condition to be detected by the chip, and then secondly, set the interrupt mask bit for the particular interrupt desired as detailed in figure 3.2 above and, lastly, set the interrupt enable latch in order to pass the interrupt on to the MAC. Any access to 'intenaddr' will enable interrupts from the card. Similarly, any access to 'intdisaddr' will disable interrupts from the interface card. The interrupt enable latch is always reset (interrupts disabled) after a power-up or system reset of the MAC.

Because only byte lane three of the NuBus interface is used on the card, only every fourth memory location is valid in the NuBus address space. For instance, the 8K byte block of PROM is addressed starting at NuBus address $FSFF 8003. The next byte of PROM is located at address $FSFF 8007. And so on thru the remaining addresses. Application writers need to keep this in mind when writing the code for their program.



NOTE:
Only byte lane-3 addresses used by card.

Figure 3.3 - NBS-DIO48 Memory Map Details

All I/O port lines from the two 82C55A chips used on the NBS-DIO48 card are brought out directly to the 'D' connector accessible from the back panel of the MAC when the card is installed in the computer. Each 82C55A chip provides 24 digital signal lines for use externally. The Intel documentation for the chip groups the I/O lines into three separate ports of 8 bits each. These ports are labeled 'Port A', 'Port B', and 'Port C'. On the NBS-DIO48 card, there are 6 ports labeled 'Port A', 'Port B', 'Port

C', 'Port D', 'Port E', and 'Port F'. The 82C55A chip #1 (at base address $FS00 0003) maps its ports A thru C to the NBS-DIO48 card's ports A thru C respectively. The 82C55A chip #2 (at base address $FS02 0003) maps its ports A thru C to the NBS-DIO48 card's ports D thru F respectively. Refer to the schematic diagram in the 'NBS-DIO48 Hardware Reference Manual' for more information.

Section 4
Driver Variable Definitions

Included on the disk that comes with the interface card is an 'include file' the user may wish to use while writing programs which utilize the NBS-DIO48 card. This file defines certain data structures and constants which are used by the driver routines for the card.

The 'dio48CtlBlk' structure is the single most important data type defined, in that all information passed to or from the driver routines are passed in various fields of this structure. This record is a 12 byte long data type with 5 distinct fields within it used. The format of 'dio48CtlBlk' is:

```
dio48CtlBlk = RECORD
        csVar:     INTEGER;       { general purpose word has call specific
                                      data.  Refer to control call desired
                                      for variable definition. }
        csFlag:    INTEGER;       { general purpose word has call specific
                                      data.  Refer to control call desired
                                      for variable definition. }
        csStatus:  INTEGER;       { call returned status information }
        csError:   INTEGER;       { call returned error information }
        csAddr:    Ptr;           { pointer to an address on the dio48 card. }

        END;

dio48CtlBlkPtr = ^dio48CtlBlk;
```

Figure 4.1 - dio48CtlBlk Structure Definition.

Before calling the driver the application must first set the fields of the dio48CtlBlk correctly for the particular driver routine it is about to call. Each driver routine expects certain parameters in the various fields of the dio48CtlBlk. Not all of the fields are used at all times. Refer to the 'Driver Functions Interface' section of this manual for specifics about the field definitions for the driver function of interest.

Two fields within the dio48CtlBlk always have a consistent definition across the driver routines and are used to return error and status codes back to the calling program. These variables are the .csError and the .csStatus fields of the record.

The .csError field of the dio48CtlBlk structure is a 2 byte word used to return error code words about the operation of the driver routine during its execution. The following error codes have been defined for the current version of the driver:

```
*       Control call Error codes returned in 'csError'
ctlNoErr       EQU     $0000                ; default error code for control calls
ctlUnkErr      EQU     $0003                ; unknown error
```

Normal execution of a driver routine will return the ctlNoErr error code and the application should invoke its error recovery handler if the driver returns anything but this value.

Upon completion of driver routine calls, a status word is also returned along with the .csError word just described. The .csStatus field of the dio48CtlBlk structure is a 2 byte word used to return status bits about the operation of the driver routine during its execution. Each bit within the .csStatus word has been defined to signify a particular status condition. Figure 4.2 shows the status bits that have been defined for the current version of the driver.

dio48CtlBlk.csStatus Word:

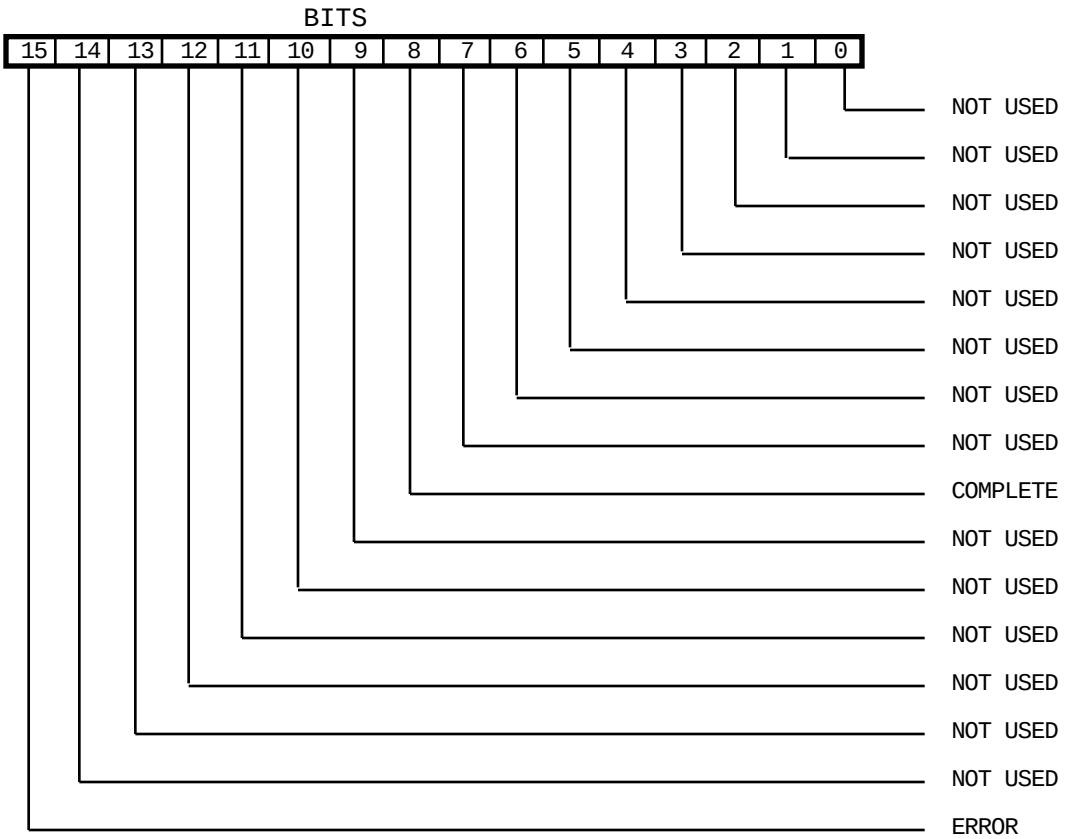


Figure 4.2 - dio48CtlBlk.csStatus Bit Definitions.

These bit definitions are defined in the include file as constants:

```
*       Status bit codes returned in 'csStatus'
stErr           EQU     $8000                   ; error occurred during call
stCmplt         EQU     $0100                   ; I/O operation completed during call
```

Bit 15 of the .csStatus field signifies that an error occurred during the execution of the driver call. It will always be accompanied by an error code of non-zero in the .csError field of the record. The other bits of the status word give the user more detailed

information about the execution of the driver call and usually do not indicate error conditions.

This page intentionally left blank

Section 5
Cookbook

All calls to the driver routines should be made thru the Device Manager of the Macintosh operating system. Consult the *Inside Macintosh* documentation for more specific information about device driver calls.

In order to make any calls to the driver of the NBS-DIO48 card, or any driver for that matter, the driver must first be opened. The driver may be opened by calling the 'OpenSlot' function call of the Macintosh slot manager. Refer to the 'dio48Open' function call documented in the 'Driver Functions Interface' section of this manual for more information. The call to open the driver will return a driver reference number which must be used for all subsequent calls to the driver.

All other calls to the NBS-DIO48 driver, with the exception of the open and close calls, are made via device manager 'Control' calls. The driver does not support 'Prime' or 'Status' calls. The standard way of calling the Control call routine of a device driver is made with a call to the Device Manager 'Control' function or the lower level 'PBControl' function. The PASCAL interface routines supplied uses the 'PBControl' routine.

In either case the application must first set up the 'dio48CtlBlk' record as defined in the previous section of this manual. Then, a pointer to the dio48CtlBlk is passed in the first four bytes of a ParamBlockRec.csParam field. Finally, the PBControl call is made by using the driver reference number and a pointer to the ParamBlockRec as pass parameters. The NBS-DIO48 driver only supports synchronous calls so the 'async' parameter should always be set to FALSE.

All driver control calls are made this way. The only difference is the ParamBlockRec.csCode parameter used and the way the dio48CtlBlk record is set up. The ParamBlockRec.csCode field should be set to the number of the particular control call being made.

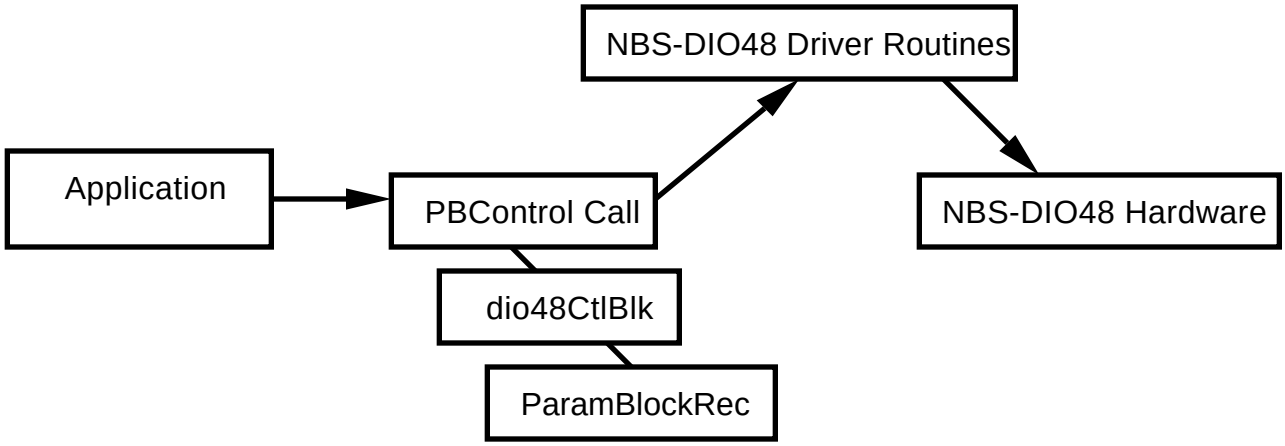
The NBS-DIO48 driver only supports the following .csCode values:

Value	Driver routine
1	KillIo;
23	EnInter;
25	Read;
26	Write;

The way the dio48CtlBlk is set up is determined by the particular driver routine being called. Each driver routine documents these values in the 'Driver Functions Interface' section of this manual.

Finally, after all calls to the driver have been made, the driver must be closed. This is done with the 'CloseDriver' function call.

The reader should refer to the PASCAL interface file source code listing for examples of what was just described.



Open Driver

For Each Driver Call Made:

Set up dio48CtlBlk

Set up ParamBlockRec

Put Pointer to dio48CtlBlk in ParamBlockRec.csParam

Call PBControl

Close Driver

Figure 5.1 - Driver call method.

As a real live example we will detail the program steps necessary to input and output data from the NBS-DIO48 interface card. We will assume that this particular application needs to output 24 bits of data and also read 24 bits of data. Each data bit is to be assigned a separate I/O line from the NBS-DIO48 card. We will use Port A, Port B, and Port C for the output data bits and Port D, Port E, and Port F for the input data bits.

The first thing which must be done is to open the driver for the NBS-DIO48 card by using the dio48Open call. This call will return a driver reference number which will be used in all succeeding calls. Once the driver has been opened we need to setup the 82C55A chips for the proper mode of operation. Since simple input and output operations are required over the interface lines, we will program the 82C55A chips to operate in mode 0 which is its basic input/output mode. We will utilize the 'WRITE' driver call to setup the I/O ports.

82C55A #1 will be setup to have its I/O lines in the 'output' configuration. To do this, the Intel documentation for the 82C55A chip shows us that we need to write a 0x80 to the control register for the chip. Likewise, 82C55A #2 will be setup to have it's I/O lines in the input configuration. To do this, the Intel documentation for the 82C55A chip shows us that we need to write a 0x9B to the control register for the chip.

After the two peripheral chips have been set up properly, it is simply a matter of reading or writing to the individual I/O port addresses in order to transfer the data bytes. Refer to the comments in the source code for this example for more information.

After all is done, we will close the driver with a call to the dio48Close routine.

```
PROCEDURE DoNbsDio48;

VAR
    gWrNum:        LONGINT;       { the value used to output to the output ports }
    gRdNum:        LONGINT;       { the value read from the input ports }
    gRefNum:       INTEGER;
    gStatus:       INTEGER;
    gError:        INTEGER;
    gBoardAddr:    INTEGER;

    myByte:        SignedByte;
    addr:          LONGINT;
    aByte:         INTEGER;
    err:           OSErr;
```

```
BEGIN
    gWrNum := $123456;       { output data }
    gRdNum := 0;             { where we will store the input data }
    gBoardAddr := $b;        { the slot we installed our board }

    { Open the driver.  Store the driver's reference number in 'gRefNum' }
    err := dio48Open(SignedByte(gBoardAddr), gRefNum);

    { set mode 0 - 'OUTPUT' operation for Port A, Port B, and Port C }
    addr := $0c;             { address on card of the control register for 82C55A #1 }
    aByte := $80;            { command for mode '0', outputs }
    err := dio48WrAddr(gRefNum, addr, SignedByte(aByte), gStatus, gError);

    { set mode 0 - 'INPUT' operation for Port D, Port E, and Port F }
    addr := $2000c;          { address on card of the control register for 82C55A #2 }
    aByte := $9b;            { command for mode '0', inputs }
    err := dio48WrAddr(gRefNum, addr, SignedByte(aByte), gStatus, gError);

    { Output the data to the output ports. The low three bytes of 'gWrNum' will
      be written to the three output ports. }
    addr := $0;              { address on card of the Port 'A' for 82C55A #1 }
    aByte := INTEGER(BitAnd(gWrNum, 255));
    err := dio48WrAddr(gRefNum, addr, SignedByte(aByte), gStatus, gError);

    addr := $4;              { address on card of the Port 'B' for 82C55A #1 }
    aByte := INTEGER(BitAnd(BitShift(gWrNum, -8), 255));
    err := dio48WrAddr(gRefNum, addr, SignedByte(aByte), gStatus, gError);

    addr := $8;              { address on card of the Port 'C' for 82C55A #1 }
    aByte := INTEGER(BitAnd(BitShift(gWrNum, -16), 255));
    err := dio48WrAddr(gRefNum, addr, SignedByte(aByte), gStatus, gError);

    { read the three input ports.  Store the result in 'gRdNum' }
    addr := $20008;          { address on card of the Port 'C' for 82C55A #2 }
    err := dio48RdAddr(gRefNum, addr, myByte, gStatus, gError);
    gRdNum := gRdNum + BitAnd(myByte, $0ff);
    gRdNum := BitShift(gRdNum, 8);

    addr := $20004;          { address on card of the Port 'B' for 82C55A #2 }
    err := dio48RdAddr(gRefNum, addr, myByte, gStatus, gError);
    gRdNum := gRdNum + BitAnd(myByte, $0ff);
    gRdNum := BitShift(gRdNum, 8);

    addr := $20000;          { address on card of the Port 'A' for 82C55A #2 }
    err := dio48RdAddr(gRefNum, addr, myByte, gStatus, gError);
    gRdNum := gRdNum + BitAnd(myByte, $0ff);

    { finally, close the driver }
    err := dio48Close(gRefNum);         { close the driver }


END;
```

This page intentionally left blank

Section 6
Driver Functions Interface

EnInter	Enable/Disable Board Interrupts	**EnInter**

Purpose: This call is used to enable or disable the ability of the NBS-DIO48 board to interrupt the MAC.

Format:
```
FUNCTION PBControl(@paramBlock, FALSE): OSErr;
```

Parameters: Input:
```
dio48CtlBlk.csFlag    - Enable/Disable BOOLEAN.
paramBlock.ioRefNum   - value returned from 'dio48Open' call
paramBlock.csCode     - '23' for this call
```
Output:
```
dio48CtlBlk.csStatus  - call return status information
dio48CtlBlk.csError   - call return error code
```

Details: Application programs call this routine in order to enable or disable the ability of the NBS-DIO48 card to interrupt the MAC. The interface card has been designed such that any interrupts generated by the Intel 82C55A controller chips used on the card can be allowed to interrupt the MAC.

In order for an interrupt to be generated however, three conditions must be met. First, the 82C55A chip must be programmed to allow the controller chip to generate an interrupt. Refer to the documentation from Intel on the 82C55A chip for more detailed information regarding the possible interrupt conditions available for the controller chip. The second condition which must be satisfied is that the interrupt mask latch for the board must be setup properly to allow the particular interrupt to pass thru to the interrupt enable latch. Refer to section 3 of this manual for information on the interrupt mask latch. The last condition which must be satisfied before the NBS-DIO48 card can interrupt the MAC is that the interrupt enable latch for the board must be set in order to pass the interrupt from the controller chip thru to the MAC. This driver function call has been provided to allow the application to set the state of the board's interrupt enable latch.

The interrupt enable latch defaults to the state which inhibits all interrupts from the board upon a reset of the MAC computer. Revision 1.0 of the NBS-DIO48 driver does not utilize the interrupt capability of the card. This routine is provided for the use of those application developers wishing to write their own interrupt driven routines for the card. The application should disable interrupts before closing the driver if they had been previously enabled during the execution of the program. Refer to 'The Device Manager' chapter of *Inside Macintosh Volume V* for more information regarding interrupts and slot devices.

```
EnInter:
     IF .csFlag non-zero THEN
           Enable board interrupts
     ELSE
           Disable Board interrupts
```

Example:

```
VAR
     err:                OSErr;
     paramBlock:         ParamBlockRec;
     mydio48CtlBlk:dio48CtlBlk;
     paramAddr:          LONGINT;
     refNum:       INTEGER;
     myStatus:           INTEGER;
     myError:            INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     mydio48CtlBlk.csVar := 0;          { not used }
     mydio48CtlBlk.csFlag := $ffff;     { enable interrupts }
     mydio48CtlBlk.csStatus := 0; { a return value }
     mydio48CtlBlk.csError := 0;        { a return value }
     mydio48CtlBlk.csAddr := NIL; { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;    { not used }
     paramBlock.ioVRefNum := 0;         { not used }
     paramBlock.ioRefNum := refNum;     { from 'dio48Open' call }
     paramBlock.csCode := 23;           { for 'EnInter' call }
     paramAddr := LONGINT(@mydio48CtlBlk);      { address of DIO48 params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := mydio48CtlBlk.csStatus; { interface's status }
     myError := mydio48CtlBlk.csError;   { driver's result code }

  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in mydio48CtlBlk.csStatus and
  mydio48CtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'dio48Open'.}
END;
```

dio48Close	Close Driver	**dio48Close**

Purpose: This call is used to close the previously opened NBS-DIO48 driver.

Format:
```
FUNCTION CloseDriver(refNum: INTEGER): OSErr;
```

Parameters: Input:

refNum - the driver reference number returned from the 'dio48Open' call.

Output:

none

Details: Application programs should call this routine after all I/O is done. It is customary to do this at the end of the application program just before terminating. The driver should have been previously opened by a call to 'dio48Open'.

Upon return from this function, the card will be left with the interrupt mask bits and the interrupt enable latch reset.

```
dio48Close:
     Reset the interrupt mask bits
     Reset the interrupt enable latch
```

Example:

```
VAR
     err:          OSErr;
     refNum:INTEGER;

BEGIN
     err := CloseDriver(refNum);

{ The success of the call is returned in 'err'.  The driver reference number
is that which was returned by the call to 'dio48Open'.}
END;
```

dio48Open	Open Driver	**dio48Open**

Purpose: This call is used to open and initialize the NBS-DIO48 driver.

Format: `FUNCTION OpenSlot(@paramBlock, FALSE): OSErr;`

Parameters: Parameters required by 'OpenSlot' function.

Details: Application programs must call this routine before making any calls to the other routines in the driver package. This is usually done once at the beginning of the application. The complimentary 'dio48Close' routine should be called after all I/O is done. Again, it is customary to do this at the end of the application program just before terminating.

The call to the slot manager routine 'OpenSlot', documented in Inside Macintosh Volume V, is used to indirectly open the NBS-DIO48 driver. See the example below for sample code.

```
dio48Open:
     Reset the interrupt mask bits
     Reset the interrupt enable latch
```

Example:

```
VAR
     err:          OSErr;
     paramBlock:   ParamBlockRec;
     nameStr:      Str255;
     mySlot:SignedByte;
     refNum:INTEGER;

BEGIN
     mySlot := $B;               { slot number board is plugged into }
     paramBlock.ioCompletion := NIL;
     nameStr := '.Fc_dio48';            { taken from driver header }
     paramBlock.ioNamePtr := @nameStr;
     paramBlock.ioPermssn := fsCurPerm;
     paramBlock.ioMix := NIL;
     paramBlock.ioFlags := 0;
     paramBlock.ioSlot := mySlot;
     paramBlock.ioId := -128;           { the DIO48 driver ID }

     err := OpenSlot(@paramBlock, FALSE);
     refnum := paramBlock.ioRefNum;

{ The success of the call is returned in 'err'.  The driver reference number
is returned in 'paramBlock.ioRefNum and is used to reference the open driver
in all subsequent calls.}

END;
```

KillIO	Halt any I/O in process	**KillIO**

Purpose: This call is used to terminate any I/O operation on the device driver.

Format: `FUNCTION PBControl(@paramBlock, FALSE): OSErr;`

Parameters: Input:

```
paramBlock.ioRefNum   - value returned from 'dio48Open' call
paramBlock.csCode     - '1' for this call
```

Output:

```
dio48CtlBlk.csStatus  - call return status information
dio48CtlBlk.csError   - call return error code
```

Details: This call has limited use with the NBS-DIO48 driver because it currently supports only synchronous calls from the device manager. It is included to provide conformance with the Mac's device manager control calls.

```
KillIO:
     Return noErr to caller
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     mydio48CtlBlk:dio48CtlBlk;
     paramAddr:           LONGINT;
     refNum:       INTEGER;
     myStatus:            INTEGER;
     myError:             INTEGER;

BEGIN
     { first set up the driver's control call parameters }
     mydio48CtlBlk.csVar := 0;                  { not used }
     mydio48CtlBlk.csFlag := 0;                 { not used }
     mydio48CtlBlk.csStatus := 0;        { not used }
     mydio48CtlBlk.csError := 0;                { not used }
     mydio48CtlBlk.csAddr := NIL;        { not used }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;            { not used }
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'dio48Open' call }
     paramBlock.csCode := 1;                    { for KillIO }
     paramAddr := LONGINT(@mydio48CtlBlk);      { address of DIO48 params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);

{ The success of the device manager call is returned in 'err'.  The driver
reference number used is that which was returned by the call to 'dio48Open'.}
END;
```

Read	Read Memory Location	**Read**

Purpose: This call is used to allow the application to read a memory address from the NBS-DIO48 card.

Format:
```
FUNCTION PBControl(@paramBlock, FALSE): OSErr;
```

Parameters: Input:
```
dio48CtlBlk.csAddr    - desired memory address.
paramBlock.ioRefNum   - value returned from 'dio48Open' call
paramBlock.csCode     - '25' for this call
```
Output:
```
dio48CtlBlk.csVar     - Byte at specified memory address.
dio48CtlBlk.csStatus  - call return status information
dio48CtlBlk.csError   - call return error code
```

Details: Applications call this routine in order to read a byte from the memory space of the NBS-DIO48 card.

This routine has been included in the driver to allow the application programmer complete access to all of the hardware functions with which the interface card is capable of. With this call an application can, for instance, read any of the registers on the 82C55A chips.

Addresses should be specified by sending the lower 24 bits of the address desired on the card, with the two LSB's zero. The driver will complete the address used for the access by adding $FS000003 to the value passed to the routine ('S' being the slot address where the card is installed). Remember that the NBS-DIO48 card only supports data transfers over byte lane 3 of the NuBus interface.

The user should consult the NBS-DIO48 memory map given in another part of this manual for a list of addresses used on the card.

```
Read:
    Get specified address.
    AND address with $00FFFFFF.
    ADD address with the board's base address ($FS000003).
    Read the byte at the calculated address.
    AND byte with $000000FF.
    Put requested byte in the low byte of the .csVar field.
    Return.
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     mydio48CtlBlk:dio48CtlBlk;
     paramAddr:           LONGINT;
     refNum:       INTEGER;
     myStatus:            INTEGER;
     myError:             INTEGER;
     theByte:             SignedByte;

BEGIN
     { first set up the driver's control call parameters }
     mydio48CtlBlk.csVar := 0;                  { a return value }
     mydio48CtlBlk.csFlag := 0;                 { not used }
     mydio48CtlBlk.csStatus := 0;        { a return value }
     mydio48CtlBlk.csError := 0;                { a return value }
     mydio48CtlBlk.csAddr := $000000;           { address of 'port A' }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;            { not used }
     paramBlock.ioVRefNum := 0;                 { not used }
     paramBlock.ioRefNum := refNum;             { from 'dio48Open' call }
     paramBlock.csCode := 25;                   { for 'Read' call }
     paramAddr := LONGINT(@mydio48CtlBlk);      { address of DIO48 params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := mydio48CtlBlk.csStatus;        { interface's status }
     myError := mydio48CtlBlk.csError;          { driver's result code }
     theByte := SignedByte(mydio48CtlBlk.csVar);{ the returned byte }

  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in mydio48CtlBlk.csStatus and
  mydio48CtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'dio48Open'.}
END;
```

Write	Write Memory Location	**Write**

Purpose: This call is used to allow the application to write a byte to a memory address on the NBS-DIO48 card.

Format:
```
FUNCTION PBControl(@paramBlock, FALSE): OSErr;
```

Parameters: Input:
```
     dio48CtlBlk.csVar     - byte to write in lower 8 bits of .csVar
     dio48CtlBlk.csAddr    - desired memory address.
     paramBlock.ioRefNum   - value returned from 'dio48Open' call
     paramBlock.csCode     - '26' for this call
```
Output:
```
     dio48CtlBlk.csStatus  - call return status information
     dio48CtlBlk.csError   - call return error code
```

Details: Applications call this routine in order to write a byte to the memory space of the NBS-DIO48 card.

This routine has been included in the driver to allow the application programmer complete access to all of the hardware functions with which the interface card is capable of. With this call an application can, for instance, write to any of the control registers on the 82C55A chips.

Addresses should be specified by sending the lower 24 bits of the address desired on the card, with the two LSB's zero. The driver will complete the address used for the access by adding $FS000003 to the value passed to the routine ('S' being the slot address where the card is installed). Remember that the NBS-DIO48 card only supports data transfers over byte lane 3 of the NuBus interface.

The user should consult the NBS-DIO48 memory map given in another part of this manual for a list of addresses used on the card.

```
Write:
     Get specified address.
     AND address with $00FFFFFF.
     ADD address with the board's base address ($FS000003).
     Write the byte at the calculated address.
     Return.
```

Example:

```
VAR
     err:                 OSErr;
     paramBlock:          ParamBlockRec;
     mydio48CtlBlk:dio48CtlBlk;
     paramAddr:           LONGINT;
     refNum:       INTEGER;
     myStatus:            INTEGER;
     myError:             INTEGER;
     theByte:             SignedByte;

BEGIN
     theByte := SignedByte($20);               { set bit 5 }

     { next set up the driver's control call parameters }
     mydio48CtlBlk.csVar := theByte;           { value to be written }
     mydio48CtlBlk.csFlag := 0;                { not used }
     mydio48CtlBlk.csStatus := 0;        { a return value }
     mydio48CtlBlk.csError := 0;               { a return value }
     mydio48CtlBlk.csAddr := $000000;          { address of 'port A' }

     { now set up the device manager's control call parameters }
     paramBlock.ioCompletion := NIL;           { not used }
     paramBlock.ioVRefNum := 0;                { not used }
     paramBlock.ioRefNum := refNum;            { from 'dio48Open' call }
     paramBlock.csCode := 26;                  { for 'Write' call }
     paramAddr := LONGINT(@mydio48CtlBlk);     { address of DIO48 params }
     paramBlock.csParam[1] := LoWord(paramAddr);
     paramBlock.csParam[0] := HiWord(paramAddr);

     err := PBControl(@paramBlock, FALSE);
     myStatus := mydio48CtlBlk.csStatus;       { interface's status }
     myError := mydio48CtlBlk.csError;         { driver's result code }

  { The success of the device manager call is returned in 'err'.  The driver's
  status and result codes are returned in mydio48CtlBlk.csStatus and
  mydio48CtlBlk.csError respectively.  The driver reference number used is that
  which was returned by the call to 'dio48Open'.}
END;
```

Section 7
dio48Glu.p Listing

```
{
  File: dio48Glu.p

    Version 1.0   15 September, 1989

Copyright © 1989-1990 by fishcamp engineering.  All rights reserved.
}


UNIT Dio48Glu;


INTERFACE

USES
    {$LOAD MacIntf.LOAD}
        MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf;
    {$LOAD}


{
CONST
}


TYPE



{
    the following structure will be used for all driver 'Control calls'
    for passing information into/from the driver.
}




    dio48CtlBlk =  RECORD
                       csVar:     INTEGER;       { general purpose word has call specific
                                                       data.  Refer to control call desired
                                                       for variable definition. }
                       csFlag:    INTEGER;       { general purpose word has call specific
                                                       data.  Refer to control call desired
                                                       for variable definition. }
                       csStatus:  INTEGER;       { call returned status information }
                       csError:   INTEGER;       { call returned error information }
                       csAddr:    Ptr;           { pointer to an address on the dio48 card. }

                   END;

    dio48CtlBlkPtr = ^dio48CtlBlk;
```

```
FUNCTION dio48Open(dio48Slot: SignedByte; VAR refNum: INTEGER): OSErr;



    { enable/disable board interrupts }
FUNCTION dio48IntEn(refNum: INTEGER; operation: BOOLEAN; VAR status, error: INTEGER): OSErr;


    { write a byte to an address on the card }
FUNCTION dio48WrAddr(refNum: INTEGER; address: UNIV Ptr; theByte: SignedByte;
                                                    VAR status, error: INTEGER): OSErr;


    { read a byte from an address on the card }
FUNCTION dio48RdAddr(refNum: INTEGER; address: UNIV Ptr; VAR theByte: SignedByte;
                                                    VAR status, error: INTEGER): OSErr;



FUNCTION dio48Close(refNum: INTEGER): OSErr;



IMPLEMENTATION



FUNCTION dio48Open(dio48Slot: SignedByte; VAR refNum: INTEGER): OSErr;
VAR
    err:          OSErr;
    paramBlock:       ParamBlockRec;
    nameStr:      Str255;

BEGIN
    paramBlock.ioCompletion := NIL;
    nameStr := '.Fc_dio48';                    { taken from driver header (not needed ???) }
    paramBlock.ioNamePtr := @nameStr;
    paramBlock.ioPermssn := fsCurPerm;         { any available permission }
    paramBlock.ioMix := NIL;
    paramBlock.ioFlags := 0;
    paramBlock.ioSlot := dio48Slot;            { the slot the user plugged into }
    paramBlock.ioId := -128;                   { the dio48 driver id }

    err := OpenSlot(@paramBlock, FALSE);
    refNum := paramBlock.ioRefNum;             { return the driver reference number }
    dio48Open := err;                          { success code }
END;
```

```
       { enable/disable board interrupts }

FUNCTION dio48IntEn(refNum: INTEGER; operation: BOOLEAN; VAR status, error: INTEGER): OSErr;
VAR
    err:              OSErr;
    paramBlock:       ParamBlockRec;
    mydio48CtlBlk:dio48CtlBlk;
    paramAddr:    LONGINT;

BEGIN
    { first set up the driver's control call parameters }
    mydio48CtlBlk.csVar := 0;              { not used }
    IF operation = TRUE THEN
        mydio48CtlBlk.csFlag := $ffff          { flag interupt enabled }
    ELSE
        mydio48CtlBlk.csFlag := 0;             { flage inteerrupt disabled }
    mydio48CtlBlk.csStatus := 0;           { a return value }
    mydio48CtlBlk.csError := 0;                { a return value }
    mydio48CtlBlk.csAddr := NIL;           { not used }

    { now set up the device manager's control call parameters }
    paramBlock.ioCompletion := NIL;
    paramBlock.ioVRefNum := 0;                 { not used }
    paramBlock.ioRefNum := refNum;             { from 'dio48Open' call }
    paramBlock.csCode := 23;                   { for 'EnInter' call }
    paramAddr := LONGINT(@mydio48CtlBlk);      { address of dio48 params }
    paramBlock.csParam[1] := LoWord(paramAddr);
    paramBlock.csParam[0] := HiWord(paramAddr);

    err := PBControl(@paramBlock, FALSE);
    status := mydio48CtlBlk.csStatus;          { interface's status }
    error := mydio48CtlBlk.csError;            { driver's result code }

    dio48IntEn := err;
END;









       { write to a board address }

FUNCTION dio48WrAddr(refNum: INTEGER; address: UNIV Ptr; theByte: SignedByte;
                                                    VAR status, error: INTEGER): OSErr;
VAR
    err:          OSErr;
    paramBlock:       ParamBlockRec;
    mydio48CtlBlk:dio48CtlBlk;
    paramAddr:    LONGINT;

BEGIN
    { first set up the driver's control call parameters }
    mydio48CtlBlk.csVar := theByte;            { the byte we are writing }
    mydio48CtlBlk.csFlag := 0;                 { not used }
    mydio48CtlBlk.csStatus := 0;           { a return value }
    mydio48CtlBlk.csError := 0;                { a return value }
    mydio48CtlBlk.csAddr := address;       { the address we wish to write }
```

```
    { now set up the device manager's control call parameters }
    paramBlock.ioCompletion := NIL;
    paramBlock.ioVRefNum := 0;                  { not used }
    paramBlock.ioRefNum := refNum;              { from 'dio48Open' call }
    paramBlock.csCode := 26;                    { for 'Write' call }
    paramAddr := LONGINT(@mydio48CtlBlk);       { address of dio48 params }
    paramBlock.csParam[1] := LoWord(paramAddr);
    paramBlock.csParam[0] := HiWord(paramAddr);

    err := PBControl(@paramBlock, FALSE);
    status := mydio48CtlBlk.csStatus;           { interface's status }
    error := mydio48CtlBlk.csError;             { driver's result code }

    dio48WrAddr := err;
END;




       { read from a board address }

FUNCTION dio48RdAddr(refNum: INTEGER; address: UNIV Ptr; VAR theByte: SignedByte;
                                                        VAR status, error: INTEGER): OSErr;
VAR
    err:              OSErr;
    paramBlock:       ParamBlockRec;
    mydio48CtlBlk:dio48CtlBlk;
    paramAddr:    LONGINT;

BEGIN
    { first set up the driver's control call parameters }
    mydio48CtlBlk.csVar := 0;               { a return value }
    mydio48CtlBlk.csFlag := 0;                 { not used }
    mydio48CtlBlk.csStatus := 0;            { a return value }
    mydio48CtlBlk.csError := 0;                { a return value }
    mydio48CtlBlk.csAddr := address;        { the address we wish to write }

    { now set up the device manager's control call parameters }
    paramBlock.ioCompletion := NIL;
    paramBlock.ioVRefNum := 0;                  { not used }
    paramBlock.ioRefNum := refNum;              { from 'dio48Open' call }
    paramBlock.csCode := 25;                    { for 'Read' call }
    paramAddr := LONGINT(@mydio48CtlBlk);       { address of dio48 params }
    paramBlock.csParam[1] := LoWord(paramAddr);
    paramBlock.csParam[0] := HiWord(paramAddr);

    err := PBControl(@paramBlock, FALSE);
    status := mydio48CtlBlk.csStatus;           { interface's status }
    error := mydio48CtlBlk.csError;             { driver's result code }
    theByte := SignedByte(mydio48CtlBlk.csVar);

    dio48RdAddr := err;
END;
```

```
FUNCTION dio48Close(refNum: INTEGER): OSErr;
VAR
    err:           OSErr;

BEGIN
    err := CloseDriver(refNum);

    dio48Close := err;
END;




END.
```

Section 8
dio48incl.a Listing

```
*   Version 1.0   15 September, 1989


* File dio48incl.a
*{Copyright © 1989-1990 by fishcamp engineering.  All rights reserved.}



*************
* Constants *
*************

dio1addr      EQU       $000000          ; first pia
dio2addr      EQU       $020000          ; second pia

intenaddr     EQU       $040000          ; interupt enable address
intdisaddr    EQU       $060000          ; interupt disable address
maskaddr      EQU       $080000          ; address of on board int mask latch
romaddr       EQU       $ff8000          ; start of rom from base address of board


*********************************************************************************
*   The following structure is used to pass data to and from the driver during all
*   Control calls to the driver.
*
*   dio48CtlBlk =  RECORD
*                      csVar:     INTEGER;      { general purpose word has call specific
*                                                   data.  Refer to control call desired
*                                                   for variable definition. }
*                      csFlag:    INTEGER;      { general purpose word has call specific
*                                                   data.  Refer to control call desired
*                                                   for variable definition. }
*                      csStatus:  INTEGER;      { call returned status information }
*                      csError:   INTEGER;      { call returned error information }
*                      csAddr:    Ptr;          { pointer to an address on the dio48 card. }
*
*                  END;
*
*   dio48CtlBlkPtr = ^dio48CtlBlk;
*
*
*
csVar         EQU       0                ; (word)      - call specific data
csFlag        EQU       csVar+2          ; (word)      - call specific data
csStatus      EQU       csFlag+2         ; (word)      - returned driver status
csError       EQU       csStatus+2       ; (word)      - returned error code
csaddr        EQU       csError+2        ; Pointer to device card address
*
*********************************************************************************



*   Control call operating system Error codes
dio48Err      EQU       -127             ; returned to the O.S.


*   Control call Error codes returned in 'csError'
ctlNoErr      EQU       $0000            ; default error code for control calls
ctlUnkErr     EQU       $0003            ; unknown error
```

```
*   Status bit codes returned in 'csStatus'
stGood         EQU        $0000             ; Default status returned
stErr          EQU        $8000             ; error occured during call
stCmplt        EQU        $0100             ; I/O operation completed during call


*   The following need to be supplied by Apple
*       sRsrc_Type values
*


dio48BoardId   EQU        $0308             ; As assigned by Apple DTS
CatDataAcq     EQU        $0010             ;
Typ82C55       EQU        $0008             ;
DrSwNBS_DI048  EQU        $0001             ;
DrHwNBS_DI048  EQU        $0001             ;


DrSwBoard      EQU        $0000             ; always 0 for board sResource
DrHwBoard      EQU        $0000             ; always 0 for board sResource

ROMSIZE        EQU        8192              ; size of on-board ROM
fhBlockSize    EQU        20                ; format/header is 20 bytes long
Rev1           EQU        1                 ; current revision level of this ROM
sRsrc_Board    EQU        1                 ; board sResource list ID
sRsrc_dio48    EQU        128               ; dio48 sResource list ID

*   Apple defined sResource list ID numbers
sRsrc_Type     EQU        1                 ; type of resource
sRsrc_Name     EQU        2                 ; name of sResource
sRsrc_Icon     EQU        3                 ; Icon for the sResource
sRsrc_DrvrDir  EQU        4                 ; Driver directory for the sResource
sRsrc_LoadRec  EQU        5                 ; Load record for the sResource
sRsrc_BootRec  EQU        6                 ; Boot record
sRsrc_Flags    EQU        7                 ; sResource flags
sRsrc_HWDevId  EQU        8                 ; Hardware device Id

*   Apple defined Board sResource entry ID numbers
STimeOut       EQU        35                ; TimeOut constant
```

This page intentionally left blank

Section 9
Driver Listing

```
MC680xx Assembler - Ver 3.10                                                                    10-Mar-90  Page   1
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

                                    * File dio48rom.a
                                    *{Copyright © 1989-1990 by fishcamp engineering.  All rights reserved.}

                                                        MACHINE                 MC68020
                                                        STRING                  C
                                                        PRINT             ON



                                    ********************************************************************************
                                    *     Begin declaration ROM
                                    ********************************************************************************

00000                               dio48DeclRom            MAIN
00000
00000
00000                               ********************************************************************************
00000                               *                       Directory
00000                               ********************************************************************************
00000                               _sRsrcDir               OSLstEntry        sRsrc_Board,_sRsrc_Board      ; References the Board sResource
00000   0100 000C           1                               DC.L        (sRsrc_Board<<24) ++ ((_sRsrc_Board-*) ** $00FFFFFF)
00004                                                       OSLstEntry        sRsrc_dio48,_sRsrc_dio48      ; References the dio48 sResource
00004   8000 0084           1                               DC.L        (sRsrc_dio48<<24) ++ ((_sRsrc_dio48-*) ** $00FFFFFF)
00008                                                       DatLstEntry       EndOfList,0       ; end of the list
00008   FF00 0000           1                               DC.L        (EndOfList<<24)+0
0000C
0000C                               ********************************************************************************
0000C                               *                       sRsrc_Board List
0000C                               ********************************************************************************
0000C                               _sRsrc_Board            OSLstEntry        sRsrc_Type,_BoardType         ; References the sResource type
0000C   0100 0014           1                               DC.L        (sRsrc_Type<<24) ++ ((_BoardType-*) ** $00FFFFFF)
00010                                                       OSLstEntry        sRsrc_Name,_BoardName         ; References the sResource name
00010   0200 0018           1                               DC.L        (sRsrc_Name<<24) ++ ((_BoardName-*) ** $00FFFFFF)
00014                                                       DatLstEntry       BoardId,dio48BoardId    ; the board Id
00014   2000 0308           1                               DC.L        (BoardId<<24)+dio48BoardId
00018                                                       OSLstEntry        VendorInfo,_VendorInfo        ; references the vendor information list
00018   2400 0034           1                               DC.L        (VendorInfo<<24) ++ ((_VendorInfo-*) ** $00FFFFFF)
0001C                                                       DatLstEntry       EndOfList,0       ; end of the list
0001C   FF00 0000           1                              DC.L        (EndOfList<<24)+0
00020
00020   0001                        _BoardType              DC.W              CatBoard                    ; the Board sResource:  <Category>
00022   0000                                                DC.W              TypBoard                    ;                                <Type>
00024   0000                                                DC.W              DrSwBoard                   ;                                <DrvrSw>
00026   0000                                                DC.W              DrHwBoard                   ;                                <DrvrHw>
00028   6669736863616D               _BoardName             DC.L              'fishcamp engineering NBS-DIO48 card'     ; board's official product name
0004C
0004C
0004C                               ********************************************************************************
0004C                               *                       Vendor info record
0004C                               ********************************************************************************
0004C                               _VendorInfo       OSLstEntry        VendorId,_VendorId            ; references the vendor Id
0004C   0100 0010           1                               DC.L        (VendorId<<24) ++ ((_VendorId-*) ** $00FFFFFF)
00050                                                       OSLstEntry        RevLevel,_RevLevel            ; references the revision level
00050   0300 0024           1                               DC.L        (RevLevel<<24) ++ ((_RevLevel-*) ** $00FFFFFF)
00054                                                       OSLstEntry        PartNum,_PartNum              ; references the part number
00054   0400 0028           1                               DC.L        (PartNum<<24) ++ ((_PartNum-*) ** $00FFFFFF)
00058                                                       DatLstEntry       EndOfList,0       ; end of the list
00058   FF00 0000           1                               DC.L        (EndOfList<<24)+0
0005C
0005C   6669736863616D               _VendorId              DC.L              'fishcamp engineering'        ; the vendor id
00074   52657620312E30               _RevLevel              DC.L              'Rev 1.0'                     ; the revision level
0007C   4E42532D44494F               _PartNum               DC.L              'NBS-DIO48'             ; the part number
00088
00088
00088                               ********************************************************************************
00088                               *                       sRsrc_dio48
00088                               ********************************************************************************
00088                               _sRsrc_dio48            OSLstEntry        sRsrc_Type,_dio48Type         ; references the sResource type
00088   0100 0014           1                               DC.L        (sRsrc_Type<<24) ++ ((_dio48Type-*) ** $00FFFFFF)
0008C                                                       OSLstEntry        sRsrc_Name,_dio48Name         ; references the sResource name
0008C   0200 0018           1                               DC.L        (sRsrc_Name<<24) ++ ((_dio48Name-*) ** $00FFFFFF)
00090                                                       OSLstEntry        sRsrc_DrvrDir,_dio48DrvrDir   ; references the driver directory
00090   0400 0034           1                               DC.L        (sRsrc_DrvrDir<<24) ++ ((_dio48DrvrDir-*) ** $00FFFFFF)
00094                                                       DatLstEntry       sRsrc_HWDevId,1         ; the hardware device Id
00094   0800 0001           1                               DC.L        (sRsrc_HWDevId<<24)+1
00098                                                       DatLstEntry       EndOfList,0       ; end of the list
00098   FF00 0000           1                               DC.L        (EndOfList<<24)+0
0009C
0009C   0010                        _dio48Type              DC.W              CatDataAcq                  ; dio48 sResource:      <Category>
0009E   0008                                                DC.W              Typ82C55                    ;                             <Type>
000A0   0001                                                DC.W              DrSwNBS_DI048               ;                             <DrvrSw>
000A2   0001                                                DC.W              DrHwNBS_DI048               ;                             <DrvrHw>
000A4
000A4   6469676974616C               _dio48Name             DC.L              'digital48_fishcamp_NBS-DIO48'
000C4
000C4                               ********************************************************************************
000C4                               *                       driver directory
000C4                               ********************************************************************************
000C4                               _dio48DrvrDir           OSLstEntry        sMacOS68020,_sMacOS68020      ; references the Macintosh-OS 68020
000C4   0200 0008           1                               DC.L        (sMacOS68020<<24) ++ ((_sMacOS68020-*) ** $00FFFFFF)
000C8                                                       DatLstEntry       EndOfList,0       ; end of the list
000C8   FF00 0000           1                               DC.L        (EndOfList<<24)+0
000CC
```

```
MC680xx Assembler - Ver 3.10                                                                    10-Mar-90  Page   2
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

000CC                                 ; Driver-1 (68020)
000CC   0000 0292                     _sMacOS68020            DC.L            _End020Drvr-_sMacOS68020      ; the physical block size
000D0                                 INCLUDE                         'dio48drvr.a'           ; the header/code
000D0                                 *     Version 1.0   15 September, 1989
000D0
000D0
000D0
000D0                                 * File dio48drvr.a
000D0                                 *{Copyright © 1989-1990 by fishcamp engineering.  All rights reserved.}
000D0
000D0
000D0
000D0
000D0                                                       BLANKS          ON
000D0                                                       STRING          ASIS
000D0
000D0
000D0
000D0
000D0
000D0                                 ********************************************************************************************
000D0                                 *                     local vars, definitions etc.
000D0                                 ********************************************************************************************
000D0
000D0
000D0
000D0
000D0
000D0
000D0                                 ;-------------------------------------------
000D0                                 ; Write the specified byte to the specified NuBus address.
000D0                                 ; The address actually used will be the address specified
000D0                                 ; added to the board's base address which should be contained in A1.
000D0                                 ; The board's base address for slot 9 with byte lane 3 used
000D0                                 ; would be $f9000003.
000D0                                 ;
000D0                                 ;     Call:                       A1 - board base address
000D0                                 ;
000D0                                 ;     Registers affected:         None
000D0                                 ;
000D0                                                       MACRO
000D0                                                       MWrite      &Address,&Data
000D0                                                       MOVEM.L     D0/A0,-(SP)             ; save work registers
000D0
000D0                                                       MOVE.L      A1,D0                   ; from board base address
000D0                                                       ADD.L       #&Address,D0            ; add to where byte will go
000D0                                                       MOVEA.L     D0,A0                   ; A0 has address
000D0                                                       MOVE.B      #&Data,D0               ; set data
000D0                                                       BSR         NbWrite
000D0
000D0                                                       MOVEM.L     (SP)+,D0/A0             ; restore registers
000D0                                                       ENDM
000D0                                 ;-------------------------------------------
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0
000D0                                 ********************************************************************************************
000D0                                 *                     dio48 driver header
000D0                                 ********************************************************************************************
000D0
000D0   4400                          dio48Drvr               DC.W        $4400                       ; ctl,needslock
000D2   0000 0000 0000                                        DC.W        0,0,0                       ; not used but required values
000D8
000D8                                 ; Entry point offset table
000D8
000D8   001E                                                  DC.W        dio48Open-dio48Drvr         ; open
000DA   0000                                                  DC.W        dio48Drvr-dio48Drvr         ; no prime
000DC   00BE                                                  DC.W        dio48Ctl-dio48Drvr          ; control
000DE   0000                                                  DC.W        dio48Drvr-dio48Drvr         ; no status
000E0   0074                                                  DC.W        dio48Close-dio48Drvr        ; close
000E2
000E2                                                         STRING      Pascal
000E2   092E46635F6469                dio48Title              DC.B        '.Fc_dio48'                 ; driver name
000EC                                                         STRING      ASIS
000EC   0000 00EC                                             ALIGN 2                                 ; force alignment
000EC   0000                                                  DC.W        0                           ; version - 0
000EE
000EE
000EE
000EE
000EE                                 ********************************************************************************************
000EE                                 *     dio48Open initializes registers.
000EE                                 *
000EE                                 *     Entry:            A0 - param blk pointer
000EE                                 *                       A1 - DCE pointer
000EE                                 *
```

```
MC680xx Assembler - Ver 3.10                                                    10-Mar-90  Page   3
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

000EE                                *     Locals:           A0 - temporary
000EE                                *                       A1 - board base address
000EE                                *                       A2 - Saved param block pointer
000EE                                *                       A3 - Saved DCE pointer
000EE                                *                       D0 - temporary
000EE                                *                       D1 - temporary
000EE                                ********************************************************************************************
000EE                                ; save registers
000EE                                dio48Open
000EE G 2448                                                 MOVE.L      A0,A2                   ; A2 <- param block pointer
000F0 G 2649                                                 MOVE.L      A1,A3                   ; A3 <- DCE pointer
000F2
000F2                                ; get base address of board
000F2   102B 0028                                            MOVE.B      dCtlSlot(A3),D0         ; get the slot address
000F6   E188                                                 LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
000F8   E188                                                 LSL.L       #8,D0
000FA   E188                                                 LSL.L       #8,D0
000FC   0080 F000 0003                                       ORI.L       #$f0000003,D0           ; Slot space
00102   2240                                                 MOVEA.L     D0,A1                   ; A1 = board base address
00104
00104                                                        MWrite      intdisaddr,0            ; disable board interrupts
00104   48E7 8080            1                             MOVEM.L     D0/A0,-(SP)             ; save work registers
00108                        1
00108   2009                1                             MOVE.L      A1,D0                   ; from board base address
0010A G 0680 0006 0000       1                             ADD.L       #intdisaddr,D0          ; add to where byte will go
00110   2040                 1                             MOVEA.L     D0,A0                   ; A0 has address
00112 G 4200                 1                             MOVE.B      #0,D0                   ; set data
00114   6100 022C       00342 1                             BSR         NbWrite
00118                        1
00118   4CDF 0101            1                             MOVEM.L     (SP)+,D0/A0             ; restore registers
0011C                                                        MWrite      maskaddr,0              ; disable pia interrupts
0011C   48E7 8080            1                             MOVEM.L     D0/A0,-(SP)             ; save work registers
00120                        1
00120   2009                1                             MOVE.L      A1,D0                   ; from board base address
00122 G 0680 0008 0000       1                             ADD.L       #maskaddr,D0            ; add to where byte will go
00128   2040                 1                             MOVEA.L     D0,A0                   ; A0 has address
0012A G 4200                 1                             MOVE.B      #0,D0                   ; set data
0012C   6100 0214       00342 1                             BSR         NbWrite
00130                        1
00130   4CDF 0101            1                             MOVEM.L     (SP)+,D0/A0             ; restore registers
00134
00134 G 426A 0010                                           MOVE.W      #noErr,ioResult(A2)     ; flag no error
00138   7000                                                MOVEQ       #noErr,D0;
0013A   6006            00142                               BRA.S       EndOpen
0013C
0013C                                ; error exit
0013C   357C FFE9 0010               OpError                MOVE.W      #openErr,ioResult(A2)   ; couldn't open driver
00142
00142   4E75                         EndOpen                RTS                                 ; return
00144
00144
00144
00144
00144
00144
00144
00144
00144
00144
00144
00144
00144                                ********************************************************************************************
00144                                *     dio48Close disables interrupts
00144                                *
00144                                *     Entry:            A0 - param blk pointer
00144                                *                       A1 - DCE pointer
00144                                *
00144                                *     Locals:           A0 - temporary
00144                                *                       A1 - board base address
00144                                *                       A2 - Saved param block pointer
00144                                *                       A3 - Saved DCE pointer
00144                                *                       D0 - temporary
00144                                *
00144                                *     Return:           D0 - error
00144                                ********************************************************************************************
00144                                dio48Close
00144 G 2448                                                 MOVE.L      A0,A2                   ; A2 <- param block pointer
00146 G 2649                                                 MOVE.L      A1,A3                   ; A3 <- DCE pointer
00148
00148                                ; get base address of board
00148   102B 0028                                            MOVE.B      dCtlSlot(A3),D0         ; get the slot address
0014C   E188                                                 LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
0014E   E188                                                 LSL.L       #8,D0
00150   E188                                                 LSL.L       #8,D0
00152   0080 F000 0003                                       ORI.L       #$f0000003,D0           ; Slot space
00158   2240                                                 MOVEA.L     D0,A1                   ; A1 = board base address
0015A
0015A                                                        MWrite      intdisaddr,0            ; disable board interrupts
0015A   48E7 8080            1                             MOVEM.L     D0/A0,-(SP)             ; save work registers
0015E                        1
0015E   2009                1                             MOVE.L      A1,D0                   ; from board base address
00160 G 0680 0006 0000       1                             ADD.L       #intdisaddr,D0          ; add to where byte will go
00166   2040                 1                             MOVEA.L     D0,A0                   ; A0 has address
00168 G 4200                 1                             MOVE.B      #0,D0                   ; set data
0016A   6100 01D6       00342 1                             BSR         NbWrite
0016E                        1
0016E   4CDF 0101            1                             MOVEM.L     (SP)+,D0/A0             ; restore registers
00172                                                        MWrite      maskaddr,0              ; disable pia interrupts
```

```
MC680xx Assembler - Ver 3.10                                                    10-Mar-90  Page   4
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

00172   48E7 8080             1                           MOVEM.L     D0/A0,-(SP)             ; save work registers
00176                         1
00176   2009                  1                           MOVE.L      A1,D0                   ; from board base address
00178 G 0680 0008 0000        1                           ADD.L       #maskaddr,D0            ; add to where byte will go
0017E   2040                  1                           MOVEA.L     D0,A0                   ; A0 has address
00180 G 4200                  1                           MOVE.B      #0,D0                   ; set data
00182   6100 01BE       00342 1                           BSR         NbWrite
00186                         1
00186   4CDF 0101              1                           MOVEM.L     (SP)+,D0/A0             ; restore registers
0018A
0018A   7000                                              MOVEQ       #noErr,D0               ; get error into D0
0018C   4E75                                              RTS                                 ; return to caller
0018E
0018E
0018E
0018E
0018E                               ********************************************************************************************
0018E                               *     dio48Ctl control call handler. 3 different calls:
0018E                               *
0018E                               *           (1)   KillIo
0018E                               *           (23)  EnInter;
0018E                               *           (25)  Read;
0018E                               *           (26)  Write;
0018E                               *
0018E                               *     Entry:      A0    - param blk pointer
0018E                               *                 A1    - DCE pointer
0018E                               *
0018E                               *     Uses:       A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0018E                               *                 A3    - scratch ( doesn't need to be preserved )
0018E                               *                 A4    - scratch ( must be preserved )
0018E                               *                 D0-D3 - scratch ( doesn't need to be preserved )
0018E                               *
0018E                               *     Exit:       D0    - error code
0018E                               *
0018E                               ********************************************************************************************
0018E                               ; decode the requested call
0018E   48E7 0888                   dio48Ctl                MOVEM.L     A0/A4/D4,-(SP)          ; save work registers (A0 is saved
00192                                                                                               ;  because it is used by ExitDrvr)
00192   3028 001A                                          MOVE.W      csCode(A0),D0           ; get the opcode
00196 G 2468 001C                                          MOVE.L      csParam(A0),A2          ; A2 <- Ptr to control parameters
0019A
0019A G 0C40 001A                                          CMP.W       #26,D0                  ; IF csCode NOT IN [0..26] THEN
0019E   6240              001E0                            BHI.S       CtlBad                  ; error, csCode out of bounds
001A0   E348                                               LSL.W       #1,D0                   ; Adjust csCode to be an index into the table
001A2   303B 0006         001AA                            MOVE.W      CtlJumpTbl(PC,D0.W),D0  ; Get the relative offset to the routine
001A6   4EFB 0002         001AA                            JMP         CtlJumpTbl(PC,D0.W)     ; GOTO the proper routine
001AA
001AA   0054                        CtlJumpTbl              DC.W        NoSupport-CtlJumpTbl    ; no support                                0
001AC   003A                                                DC.W        CtlGood-CtlJumpTbl      ; KillIo                                    1
001AE   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                2
001B0   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                3
001B2   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                4
001B4   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                5
001B6   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                6
001B8   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                7
001BA   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                8
001BC   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                9
001BE   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                10
001C0   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                11
001C2   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                12
001C4   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                13
001C6   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                14
001C8   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                15
001CA   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                16
001CC   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                17
001CE   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                18
001D0   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                19
001D2   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                20
001D4   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                21
001D6   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                22
001D8   0076                                                DC.W        EnInter-CtlJumpTbl      ; enable/disable board interrupts           23
001DA   0054                                                DC.W        NoSupport-CtlJumpTbl    ; no support                                24
001DC   00E4                                                DC.W        Read-CtlJumpTbl         ; Read byte from board memory       25
001DE   0134                                                DC.W        Write-CtlJumpTbl        ; Write byte from board memory              26
001E0
001E0   70EF                        CtlBad                  MOVEQ       #controlErr,D0          ; say we don't do this one
001E2   6002              001E6                             BRA.S       CtlDone                 ; and return
001E4
001E4   7000                        CtlGood                 MOVEQ       #noErr,D0               ; return no error
001E6
001E6   4CDF 1110                   CtlDone                 MOVEM.L     (SP)+,A0/A4/D4          ; restore registers
001EA   6002              001EE                             BRA.S       ExitDrvr
001EC
001EC   4E71                                                NOP
001EE
001EE
001EE
001EE
001EE                               ********************************************************************************************
001EE                               *     Exit from control calls
001EE                               ********************************************************************************************
001EE
001EE   0828 0009 0006              ExitDrvr                BTST        #NoQueueBit,ioTrap(A0)  ; no queue bit set ?
001F4   6702              001F8                             BEQ.S       GoIODone                ; => no, not immediate
001F6   4E75                                                RTS                                 ; otherwise, it was an immediate call
```

```
MC680xx Assembler - Ver 3.10                                                    10-Mar-90  Page   5
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

001F8
001F8 G 2078 08FC                   GoIODone                MOVE.L      JIODone,A0              ; get IODone address
001FC   4ED0                                                JMP         (A0)                    ; invoke it
001FE
001FE
001FE
001FE
001FE
001FE                               ***************************************************************************************
001FE                               *     NoSupport - control call not supported
001FE                               *
001FE                               *     Entry:            A0 - param blk pointer
001FE                               *                       A1 - DCE pointer
001FE                               *                       A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
001FE                               *
001FE                               ***************************************************************************************
001FE                               NoSupport
001FE   48E7 8040                                           MOVEM.L     A1/D0,-(SP)             ; save local work registers
00202
00202                               ; get base address of board
00202   1029 0028                                           MOVE.B      dCtlSlot(A1),D0         ; get the slot address
00206   E188                                                LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
00208   E188                                                LSL.L       #8,D0
0020A   E188                                                LSL.L       #8,D0
0020C   0080 F000 0003                                      ORI.L       #$f0000003,D0           ; Slot space
00212   2240                                                MOVEA.L     D0,A1                   ; A1 = board base address
00214
00214   70EF                        NoSprt1                 MOVEQ       #controlErr,D0          ; say we don't do this one
00216
00216   4CDF 0201                   NoSprtDone              MOVEM.L     (SP)+,A1/D0             ; restore local registers
0021A   4CDF 1110                                           MOVEM.L     (SP)+,A0/A4/D4          ; restore registers
0021E   60CE            001EE                               BRA.S       ExitDrvr
00220
00220
00220
00220
00220
00220
00220                               ***************************************************************************************
00220                               *     EnInter - control call to enable/disable board interrupts
00220                               *
00220                               *     Entry:            A0 - param blk pointer
00220                               *                       A1 - DCE pointer
00220                               *                       A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00220                               *
00220                               ***************************************************************************************
00220                               EnInter
00220   48E7 8040                                           MOVEM.L     A1/D0,-(SP)             ; save local work registers
00224
00224                               ; get base address of board
00224   1029 0028                                           MOVE.B      dCtlSlot(A1),D0         ; get the slot address
00228   E188                                                LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
0022A   E188                                                LSL.L       #8,D0
0022C   E188                                                LSL.L       #8,D0
0022E   0080 F000 0003                                      ORI.L       #$f0000003,D0           ; Slot space
00234   2240                                                MOVEA.L     D0,A1                   ; A1 = board base address
00236
00236   302A 0002                                           MOVE.W      csFlag(A2),D0           ; get desired operation
0023A G 0C40 0000                                           CMP.W       #0,D0                   ; enable or disable ?
0023E   671A            0025A                               BEQ.S       EnInter1                ; to diasable
00240                                                       MWrite      intenaddr,0             ; enable interrupts
00240   48E7 8080               1                          MOVEM.L     D0/A0,-(SP)             ; save work registers
00244                           1
00244   2009                    1                          MOVE.L      A1,D0                   ; from board base address
00246 G 0680 0004 0000          1                          ADD.L       #intenaddr,D0           ; add to where byte will go
0024C   2040                    1                          MOVEA.L     D0,A0                   ; A0 has address
0024E G 4200                    1                          MOVE.B      #0,D0                   ; set data
00250   6100 00F0       00342  1                          BSR         NbWrite
00254                           1
00254   4CDF 0101              1                          MOVEM.L     (SP)+,D0/A0             ; restore registers
00258   6018            00272                               BRA.S       EnInterGood
0025A
0025A                               EnInter1                MWrite      intdisaddr,0            ; disable interrupts
0025A   48E7 8080               1                          MOVEM.L     D0/A0,-(SP)             ; save work registers
0025E                           1
0025E   2009                    1                          MOVE.L      A1,D0                   ; from board base address
00260 G 0680 0006 0000          1                          ADD.L       #intdisaddr,D0          ; add to where byte will go
00266   2040                    1                          MOVEA.L     D0,A0                   ; A0 has address
00268 G 4200                    1                          MOVE.B      #0,D0                   ; set data
0026A   6100 00D6       00342  1                          BSR         NbWrite
0026E                           1
0026E   4CDF 0101              1                          MOVEM.L     (SP)+,D0/A0             ; restore registers
00272
00272   7000                                EnInterGood     MOVEQ       #noErr,D0               ; return no error
00274 G 426A 0006                                           MOVE.W      #ctlNoErr,csError(A2)
00278 G 426A 0004                                           MOVE.W      #stGood,csStatus(A2)    ; Default status
0027C   006A 0100 0004                                      ORI.W       #stCmplt,csStatus(A2)   ; flag call complete
00282
00282   4CDF 0201                   EnInterDone             MOVEM.L     (SP)+,A1/D0             ; restore local registers
00286   4CDF 1110                                           MOVEM.L     (SP)+,A0/A4/D4          ; restore registers
0028A   6000 FF62       001EE                               BRA         ExitDrvr
0028E
0028E
0028E
0028E
0028E
```

```
MC680xx Assembler - Ver 3.10                                                    10-Mar-90  Page   6
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

0028E                               ********************************************************************************
0028E                               *     Read - read byte from board memory
0028E                               *
0028E                               *     Entry:            A0 - param blk pointer
0028E                               *                       A1 - DCE pointer
0028E                               *                       A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
0028E                               *
0028E                               ********************************************************************************
0028E                               Read
0028E   48E7 C0C0                                           MOVEM.L     A0/A1/D0/D1,-(SP)       ; save local work registers
00292                               
00292                               ; get base address of board
00292   1029 0028                                           MOVE.B      dCtlSlot(A1),D0         ; get the slot address
00296   E188                                               LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
00298   E188                                               LSL.L       #8,D0
0029A   E188                                               LSL.L       #8,D0
0029C   0080 F000 0003                                     ORI.L       #$f0000003,D0           ; Slot space
002A2   2240                                               MOVEA.L     D0,A1                   ; A1 = board base address
002A4                               
002A4                               ; get address
002A4   200A                                               MOVE.L      A2,D0
002A6 G 5080                                               ADD.L       #csAddr,D0
002A8   2040                                               MOVEA.L     D0,A0
002AA   2010                                               MOVE.L      (A0),D0                 ; D0 = address on board
002AC   0280 00FF FFFF                                     ANDI.L      #$00ffffff,D0           ; mask to 24 bits
002B2   2209                                               MOVE.L      A1,D1                   ; get board base address
002B4   D081                                               ADD.L       D1,D0                   ; add it to the requested address
002B6   2040                                               MOVEA.L     D0,A0                   ; A0 = requested address on board
002B8   616E            00328                              BSR.S       NbRead                  ; get the byte
002BA   0280 0000 00FF                                     ANDI.L      #$000000ff,D0           ; mask off lower byte
002C0   3480                                               MOVE.W      D0,csVar(A2)            ; Return to caller the requested byte.
002C2                               
002C2   7000                        ReadGood               MOVEQ       #noErr,D0               ; return no error
002C4 G 426A 0006                                          MOVE.W      #ctlNoErr,csError(A2)
002C8 G 426A 0004                                          MOVE.W      #stGood,csStatus(A2)    ; Default status
002CC   006A 0100 0004                                     ORI.W       #stCmplt,csStatus(A2)   ; flag call complete
002D2                               
002D2   4CDF 0303                   ReadDone               MOVEM.L     (SP)+,A0/A1/D0/D1       ; restore local registers
002D6   4CDF 1110                                          MOVEM.L     (SP)+,A0/A4/D4          ; restore registers
002DA   6000 FF12       001EE                              BRA         ExitDrvr
002DE
002DE
002DE
002DE
002DE                               ********************************************************************************
002DE                               *     Write - write byte to board memory
002DE                               *
002DE                               *     Entry:            A0 - param blk pointer
002DE                               *                       A1 - DCE pointer
002DE                               *                       A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
002DE                               *
002DE                               ********************************************************************************
002DE                               Write
002DE   48E7 C0C0                                           MOVEM.L     A0/A1/D0/D1,-(SP)       ; save local work registers
002E2                               
002E2                               ; get base address of board
002E2   1029 0028                                           MOVE.B      dCtlSlot(A1),D0         ; get the slot address
002E6   E188                                               LSL.L       #8,D0                   ; shift the 4 slot bits into proper position
002E8   E188                                               LSL.L       #8,D0
002EA   E188                                               LSL.L       #8,D0
002EC   0080 F000 0003                                     ORI.L       #$f0000003,D0           ; Slot space
002F2   2240                                               MOVEA.L     D0,A1                   ; A1 = board base address
002F4                               
002F4                               ; get address
002F4   200A                                               MOVE.L      A2,D0
002F6 G 5080                                               ADD.L       #csAddr,D0
002F8   2040                                               MOVEA.L     D0,A0
002FA   2010                                               MOVE.L      (A0),D0                 ; D0 = address on board
002FC   0280 00FF FFFF                                     ANDI.L      #$00ffffff,D0           ; mask to 24 bits
00302   2209                                               MOVE.L      A1,D1                   ; get board base address
00304   D081                                               ADD.L       D1,D0                   ; add it to the requested address
00306   2040                                               MOVEA.L     D0,A0                   ; A0 = requested address on board
00308   3012                                               MOVE.W      csVar(A2),D0            ; D0 contains the byte to be written
0030A   6136            00342                              BSR.S       NbWrite                 ; write the byte
0030C                               
0030C   7000                        WriteGood              MOVEQ       #noErr,D0               ; return no error
0030E G 426A 0006                                          MOVE.W      #ctlNoErr,csError(A2)
00312 G 426A 0004                                          MOVE.W      #stGood,csStatus(A2)    ; Default status
00316   006A 0100 0004                                     ORI.W       #stCmplt,csStatus(A2)   ; flag call complete
0031C                               
0031C   4CDF 0303                   WriteDone              MOVEM.L     (SP)+,A0/A1/D0/D1       ; restore local registers
00320   4CDF 1110                                          MOVEM.L     (SP)+,A0/A4/D4          ; restore registers
00324   6000 FEC8       001EE                              BRA         ExitDrvr
00328
00328
00328
00328
00328
00328
00328                               ********************************************************************************
00328                               *     NbRead - reads a byte from a NUBUS card
00328                               *
00328                               *     Enter:      A0 - pointer to address in 32-bit address space
00328                               *
00328                               *     Uses:       no other registers
00328                               *
00328                               *     Exit:       D0 - the byte in low 8 bits
```

```
MC680xx Assembler - Ver 3.10                                                    10-Mar-90  Page   7
Copyright Apple Computer, Inc. 1984-1989

Loc   F Object Code     Addr  M     Source Statement

00328                                 *
00328                                 ********************************************************************************************
00328                                 NbRead
00328   48E7 4000                                       MOVEM.L           D1,-(SP)              ; save work register
0032C
0032C   2038 0001                                      MOVE.L            true32b,D0            ; set 32-bit mode
00330   A05D                                            _SwapMMUMode
00332
00332   1210                                            MOVE.B            (A0),D1               ; Get value specified
00334
00334   2038 0000                                       MOVE.L            false32b,D0
00338   A05D                                            _SwapMMUMode                            ; back to 24-bit mode
0033A
0033A   1001                                            MOVE.B            D1,D0                 ; return value
0033C   4CDF 0002                                      MOVEM.L           (SP)+,D1              ; restore work register
00340   4E75                                            RTS
00342
00342
00342                                 ********************************************************************************************
00342                                 *     NbWrite - writes a byte to a NUBUS card
00342                                 *
00342                                 *     Enter:            A0 - pointer to address in 32-bit address space
00342                                 *                       D0 - the byte in low 8 bits
00342                                 *
00342                                 *     Uses:       no other registers
00342                                 *
00342                                 ********************************************************************************************
00342                                 NbWrite
00342   48E7 4000                                       MOVEM.L           D1,-(SP)              ; save work registers
00346
00346   2200                                            MOVE.L            D0,D1                 ; save value in D1
00348
00348   2038 0001                                       MOVE.L            true32b,D0            ; set 32-bit mode
0034C   A05D                                            _SwapMMUMode
0034E
0034E   1081                                            MOVE.B            D1,(A0)               ; write value specified
00350
00350   2038 0000                                       MOVE.L            false32b,D0
00354   A05D                                            _SwapMMUMode                            ; back to 24-bit mode
00356
00356   2001                                            MOVE.L            D1,D0                 ; restore entry value
00358   4CDF 0002                                       MOVEM.L           (SP)+,D1              ; restore work register
0035C   4E75                                            RTS
0035E
0035E
0035E
0035E
0035E   0000 035E                     _End020Drvr       EQU                     *               ; the end of the driver
0035E                                                   STRING                  C
0035E
0035E
0035E
0035E
0035E
0035E
0035E   0000 1FEC                                       ORG               ROMSize-fhBlockSize
01FEC
01FEC                                 ********************************************************************************************
01FEC                                 *                 format/header block
01FEC                                 ********************************************************************************************
01FEC   00FF E014                                       DC.L              (_sRsrcDir-*)**$00ffffff      ; offset to sResource directory
01FF0   0000 2000                                       DC.L              ROMSize                       ; length of declaration data
01FF4   0000 0000                                       DC.L              0                             ; CRC (Patched by crcPatch, an MPW tool
01FF8   01                                              DC.B              Rev1                          ; revision level
01FF9   01                                              DC.B              AppleFormat                   ; format
01FFA   5A93 2BC7                                       DC.L              TestPattern                   ; test pattern
01FFE   00                                              DC.B              0                             ; Reserved byte (must be zero)
01FFF   78                                              DC.B              $78                           ; Byte lanes: 0111 1000 (bytelane 3)
02000
02000
02000                                                   ENDMAIN

                                                        END

Elapsed time: 6.46 seconds.

Assembly complete - no errors found.  5396 lines.
```