fishcamp engineering

NBS-DIO96 - Digital Input/Output Card for NuBus

fishcamp engineering
4860 Ontario Way
Santa Maria, CA 93455
TEL: 805-345-2324
Fax: 805-345-2325
eMail: bobtek@fishcamp.com
Internet: http://www.fishcamp.com

Limited Warranty

The NBS-DIO96 interface hardware is warranted to be free from defects in materials and workmanship for a period of one year from date of shipment from fishcamp engineering. Defects caused by misuse, abuse, or shipment are not covered.

Defective equipment that is subject to this limited warranty will be repaired or replaced at the option of fishcamp engineering if we are notified during the warranty period. The customer must obtain a Return Material Authorization (RMA) number before returning any equipment. Shipping costs from fishcamp engineering will be paid by fishcamp engineering. Equipment should be packaged in the original shipping container if possible, and the RMA number must be clearly marked on the outside of the package.

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.

1.0 Introduction

The NBS-DIO96 card is a very simple digital interface to the Apple Macintosh line of personal computers. It provides for 96 TTL compatible signal lines which may be configured, in groups of eight, to act as either input or output signals. Interconnection between the user's circuitry and the NBS-DIO96 card is via three 50-pin ribbon cable headers on the card. Each of the connectors carries 32 signal lines as well as ground reference and fused +5V power from the computer.

Software control of the NBS-DIO96 card is facilitated by a device driver compatible with the device manager of the Macintosh operating system. The device driver is written in 68K assembly language and resides in a PROM on the card. The MAC loads the driver into system memory upon startup so that the services of the driver are available to user programs.

We have tried to provide as much information as possible on the NBS-DIO96 card so that users will never be stymied in their development cycle because of the lack of relevant information. To this end we have provided both schematic diagrams of the hardware logic on the card and software source code listings of the driver and utility programs shipped with the card. The user is encouraged to examine these documents when more detail is needed on the architecture of the NBS-DIO96 card.

2.0 Hardware

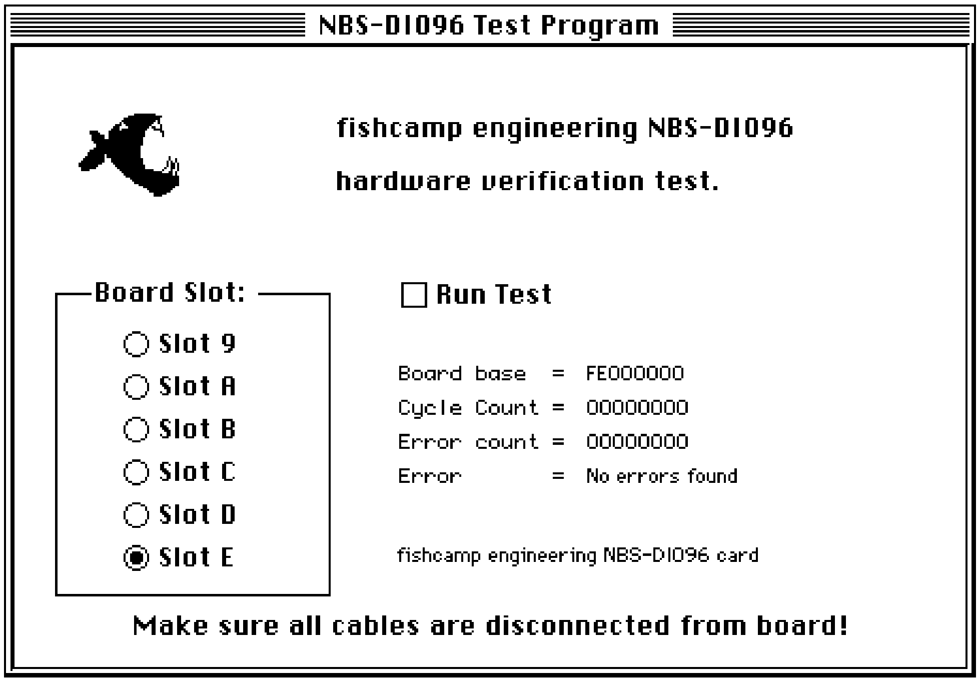
2.1 Installation

The NBS-DIO96 card may be installed in any Macintosh NuBus expansion slot. Make certain the computer is turned off prior to installing the card. Follow Apple Computer's recommendation regarding installation of cards in your computer.

The first thing to do after card installation is to run the 'NBS-DIO96 Check' program. Do this before connecting any of your interface cables to the I/O ports on the card. The NBS-DIO96 Check program will drive the interface pins with a data pattern during its test so you want to make certain that nothing else is connected to these pins at the same time. You also do not want any of your own equipment to respond to the data signals on the card during the test.

The check program will read a 'romimage' file upon startup. This file is a carbon copy of the PROM contents on the card and the program uses this to verify the correct contents of the PROM during its test. The 'romimage' file should be in the same folder as the NBS-DIO96 Check' program itself when you start it up. If the program doesn't find the file, it will ask you to locate it with a standard MAC file navigator window.

To run the check program, double-click the application icon from the Finder and the program will begin executing. The program will display the window illustrated in figure 1. The user should select the slot number where the NBS-DIO96 card was installed and then click the 'Run Test' button. Any errors will be displayed on the 'Error' line. If any errors are found please contact fishcamp engineering for repair instructions.



```
Board base   =  FE000000
Cycle Count  =  00000000
Error count  =  00000000
Error        =  No errors found

fishcamp engineering NBS-DIO96 card
```

Make sure all cables are disconnected from board!

Figure 1 - Main window of the NBS-DIO96 test program.

2.2 Hardware Overview

The NBS-DIO96 card's logic is implemented on a 7" long NuBus card. This 'half-length' card should be capable of being used in any of the Macintosh computers which support the NuBus interface. Only +5 volt power is required by the card.

The I/O signals to the card are made available to the user via three 50-pin ribbon cable headers. They are referenced in the software as 'port1', 'port2', and 'port3'. The connectors are labeled 'P1', 'P2' and 'P3' on the silkscreen on the printed circuit card. Each port/connector carries 32 of the 96 signals supported by the NBS-DIO96 card. Along with the 32 I/O signals are included 17 ground connections and fused +5v power from the MAC.

The +5v power can be used to power custom circuitry in the user's interface provided that the limits of the computer's power supply is not exceeded. Each model of computer has its own power specification and the user should consult the documentation for the particular model being used. In addition, each of the +5v power lines made available to the user are fuse protected with a 5 Amp fuse in order to protect the computer incase of power shorts in the user's equipment.

A total of seventeen ground connections are provided on each connector in order to help maintain signal integrity in noisy environments or when using long cable runs. Each of the three connectors have the same pinout thus simplifying interconnection to the user's equipment.

2.3 NuBus Interface Logic

The NBS-DIO96 card implements a full 32-bit interface to the Macintosh NuBus. The control logic allows the NBS-DIO96 card to respond as a slave only device on the NuBus. Therefore, only CPU generated read and write operations are supported. The card cannot assume bus mastership nor generate interrupts. Block mode is not supported.

Normally, the user's program simply reads from or writes to the addresses assigned to the I/O ports on the card. In this way the user may acquire the state of the input signals or write new values to the output signals.

There are four memory registers on the NBS-DIO96 which may be written or read by the MAC's processor. Three of these registers are assigned to the I/O ports on the card. One register for each port. These port registers are a full 32 bits in length with a bit corresponding to its respective signal on the I/O port. Because the NuBus interface logic implements the full 32 bit interface, all of the signals on a port may be accessed in a single memory reference operation by the processor.

The fourth register on the card is a control register which, among other things, allows the user to specify the direction of the I/O signals. Only 14 bits are defined in the control register. For more information on the control register please refer to section 2.4.

One last memory block is defined on the NBS-DIO96 card's NuBus slot space. This is the range of memory in which the MAC's processor may read the contents of a PROM on the card. Within this PROM resides the device manager's driver firmware which is executed by the MAC as well as the fuse map for a field programmable gate array on the card. Normally the fuse map data is used to configure the logic on the card upon power-up and is not normally read by the CPU. It is accessible by the CPU for diagnostic purposes during board manufacture. The PROM memory space on the NBS-DIO96 is mapped as a byte-wide interface to the NuBus. This was done in order to save on the cost to manufacture the NBS-DIO96 card. It does not compromise performance during the normal operation of the card since the driver code within the PROM is read into system memory upon startup and then executed out of system memory after that. All I/O operations to the ports on the card are always performed in full 32 bit operations.

2.4 Port Logic

As mentioned in the previous section, there are four registers defined in the memory map of the NBS-DIO96 card. The first register is a control register which is located at address $fs00000c. Where 's' is the number of the slot in which the NBS-DIO96 card is inserted in the computer. The control register is cleared to all zero's upon restart of the computer.

Only 14 of the bits are valid within the control register. The 12 least significant bits control the direction of the I/O port signals on the card. Each of the three I/O ports on the NBS-DIO96 card provide for 32 user signal lines on its interface. These 32 signal bits are grouped into four groups of eight bits (four bytes). Within each byte the user has control over the direction of the signal flow within that particular byte. All eight of the signals in the byte will assume the same signal direction flow. The way the user specifies the signal direction is by that byte's corresponding direction control bit in the control register. There are 12 direction control bits defined in the register. One for each of the 12 bytes or 96 signal lines provided for by the NBS-DIO96 card. A '1' written into the data direction bit will set the 'output' mode for that particular byte. Conversely, a '0' programmed into the data direction bit will set that byte to the 'input' mode of operation. As mentioned above, the NBS-DIO96 card will clear all of the bits in the control register each time the Macintosh computer is restarted. Therefore all 96 signal lines will be placed in the 'input' mode of operation and the card will not drive any of the I/O signals. The driver's 'open' routine also clears all of the registers on the card.

The other two bits defined in the control register are used to define certain modes of operation of the NBS-DIO96 card. The 'latch_data' bit is used to enable a data sampling clock on the card. This clock is used to latch the state of the input signals being acquired by the NBS-DIO96 card from the user's equipment before being read by the CPU. Even though it is true that any data acquisition program running on the MAC will usually be sampling the user's inputs asynchronously to any events in the user's equipment, it is sometimes advantageous to enable the 'latch_data' bit in the control register on the NBS-DIO96 card. This is usually done if the NBS-DIO96 card will be sampling very fast changing data or when the input signal's transition time is very slow. When the 'latch_data' bit is set in the control register, the NBS-DIO96 card will clock the input data into an intermediate register with a sample clock signal prior to sending the data on to the MAC's processor during a 'read' operation. This input latch has a much smaller sample window than the normal input logic on the card and thus will be better able to correctly sample data during these periods of uncertainty. The 'latch_data' bit is cleared to '0' upon restart of the Macintosh computer.

The last bit defined in the control register is a bit which is used to facilitate testing of the NBS-DIO96 board during its manufacture at the fishcamp facilities. Normally this bit is left in the cleared ('0') state during operation. The user should never set this bit to a '1' during operation of the NBS-DIO96 card. This bit is cleared upon restart of the Macintosh computer and whenever the driver is opened for the card.

The remaining three registers defined in the memory map of the NBS-DIO96 card correspond to the three I/O ports of the card. It is thru these three registers that the user's program can input or output data from the NBS-DIO96 card.

Each port is addressed from the software by means of long-word memory reads and writes to that port's register memory location on the NBS-DIO96. Each of the 32 bits in the long-word corresponds to a particular signal on the connector of the I/O port for that register. Since the NBS-DIO96 card is designed as a full 32 bit NuBus interface, all 32 signals lines may be written or read in a single long-word memory operation from the Macintosh processor. The following table lists the I/O port mapping of the card:

Port #	Register Memory Address	Port Connector
1	$fs000000	P3
2	$fs000004	P2
3	$fs000008	P1

Figure 2 - I/O Port Mapping.

The programmer must pay particular attention to the byte-swapping mechanism inherent in the NuBus architecture of the Macintosh when writing application programs for the NBS-DIO96. The 68K processor used in the Macintosh computer will have its byte lanes swapped when transferring long-words across the NuBus interface. The following diagram shows the byte-swapping operation.

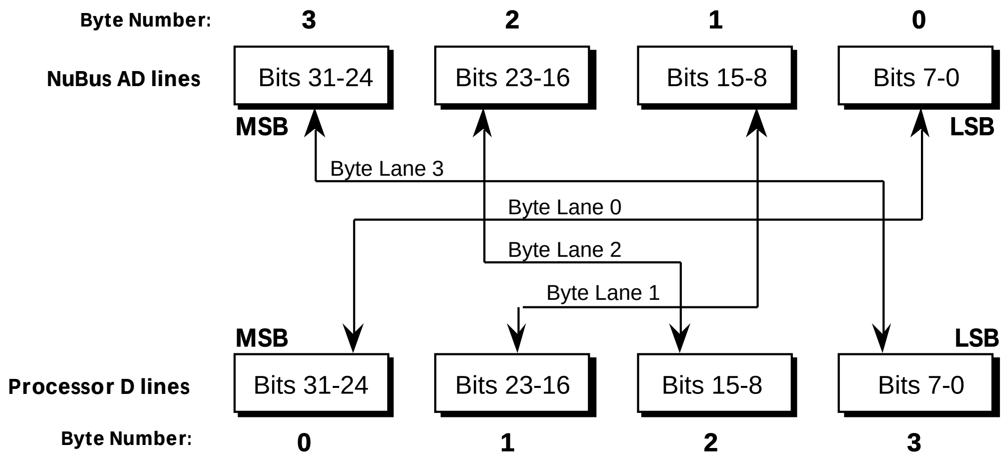


Figure 3 - NuBus Byte-Swapping.

Given the above diagram, the following tables will be useful to the programmer:

MAC Processor Data Bits	NBS-DIO96 Data Bits	Port Connector Pins	Data Direction Bit
31	7	P3-15	Dir 0
30	6	P3-13	↓
29	5	P3-11	↓
28	4	P3-9	↓
27	3	P3-7	↓
26	2	P3-5	↓
25	1	P3-3	↓
24	0	P3-1	
23	15	P3-26	Dir 1
22	14	P3-25	↓
21	13	P3-23	↓
20	12	P3-22	↓
19	11	P3-20	↓
18	10	P3-19	↓
17	9	P3-17	↓
16	8	P3-16	
15	23	P3-38	Dir 2
14	22	P3-37	↓
13	21	P3-35	↓
12	20	P3-34	↓
11	19	P3-32	↓
10	18	P3-31	↓
9	17	P3-29	↓
8	16	P3-28	
7	31	P3-50	Dir 3
6	30	P3-49	↓
5	29	P3-47	↓
4	28	P3-46	↓
3	27	P3-44	↓
2	26	P3-43	↓
1	25	P3-41	↓
0	24	P3-40	

Table 1 - NBS-DIO96 Port 1 Bit Mapping

MAC Processor Data Bits	NBS-DIO96 Data Bits	Port Connector Pins	Data Direction Bit
31	7	P2-15	Dir 4
30	6	P2-13	
29	5	P2-11	
28	4	P2-9	
27	3	P2-7	
26	2	P2-5	
25	1	P2-3	
24	0	P2-1	
23	15	P2-26	Dir 5
22	14	P2-25	
21	13	P2-23	
20	12	P2-22	
19	11	P2-20	
18	10	P2-19	
17	9	P2-17	
16	8	P2-16	
15	23	P2-38	Dir 6
14	22	P2-37	
13	21	P2-35	
12	20	P2-34	
11	19	P2-32	
10	18	P2-31	
9	17	P2-29	
8	16	P2-28	
7	31	P2-50	Dir 7
6	30	P2-49	
5	29	P2-47	
4	28	P2-46	
3	27	P2-44	
2	26	P2-43	
1	25	P2-41	
0	24	P2-40	

Table 2 - NBS-DIO96 Port 2 Bit Mapping

MAC Processor Data Bits	NBS-DIO96 Data Bits	Port Connector Pins	Data Direction Bit
31	7	P1-15	Dir 8
30	6	P1-13	
29	5	P1-11	
28	4	P1-9	
27	3	P1-7	
26	2	P1-5	
25	1	P1-3	
24	0	P1-1	
23	15	P1-26	Dir 9
22	14	P1-25	
21	13	P1-23	
20	12	P1-22	
19	11	P1-20	
18	10	P1-19	
17	9	P1-17	
16	8	P1-16	
15	23	P1-38	Dir 10
14	22	P1-37	
13	21	P1-35	
12	20	P1-34	
11	19	P1-32	
10	18	P1-31	
9	17	P1-29	
8	16	P1-28	
7	31	P1-50	Dir 11
6	30	P1-49	
5	29	P1-47	
4	28	P1-46	
3	27	P1-44	
2	26	P1-43	
1	25	P1-41	
0	24	P1-40	

Table 3 - NBS-DIO96 Port 3 Bit Mapping

2.5 Connector Pinouts

Each of the three ports on the NBS-DIO96 card has its signals made available for the customer's use via a standard 50-pin ribbon cable header. These headers have contacts made up of 25mil square pins. There are two rows of 25 pins each for a total of 50 pins. The numbering of the pins is as shown in figure X.x.

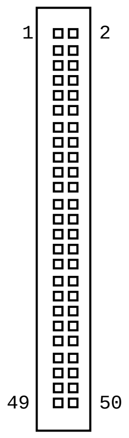


Figure 4 - I/O port pin numbering.
(The view shown is from the component side of the NBS-DIO96 card.)

Each of the connectors contains all of the signals for a single port on the NBS-DIO96 card or 32 signal lines. The remaining pins carry logic ground (17 pins) and fused +5V power (1 pin) from the computer's power supply. All three connectors have the same pinout defined for the signals on their respective ports.

I/O Connector Pin	Signal
P3-1	Port 1 - Data Bit 0
P3-2	+5V
P3-3	Port 1 - Data Bit 1
P3-4	Ground
P3-5	Port 1 - Data Bit 2
P3-6	Ground
P3-7	Port 1 - Data Bit 3
P3-8	Ground
P3-9	Port 1 - Data Bit 4
P3-10	Ground
P3-11	Port 1 - Data Bit 5
P3-12	Ground
P3-13	Port 1 - Data Bit 6
P3-14	Ground
P3-15	Port 1 - Data Bit 7
P3-16	Port 1 - Data Bit 8
P3-17	Port 1 - Data Bit 9
P3-18	Ground
P3-19	Port 1 - Data Bit 10
P3-20	Port 1 - Data Bit 11
P3-21	Ground
P3-22	Port 1 - Data Bit 12
P3-23	Port 1 - Data Bit 13
P3-24	Ground
P3-25	Port 1 - Data Bit 14
P3-26	Port 1 - Data Bit 15
P3-27	Ground
P3-28	Port 1 - Data Bit 16
P3-29	Port 1 - Data Bit 17
P3-30	Ground
P3-31	Port 1 - Data Bit 18
P3-32	Port 1 - Data Bit 19
P3-33	Ground
P3-34	Port 1 - Data Bit 20
P3-35	Port 1 - Data Bit 21
P3-36	Ground
P3-37	Port 1 - Data Bit 22
P3-38	Port 1 - Data Bit 23
P3-39	Ground
P3-40	Port 1 - Data Bit 24
P3-41	Port 1 - Data Bit 25
P3-42	Ground
P3-43	Port 1 - Data Bit 26
P3-44	Port 1 - Data Bit 27
P3-45	Ground
P3-46	Port 1 - Data Bit 28
P3-47	Port 1 - Data Bit 29
P3-48	Ground
P3-49	Port 1 - Data Bit 30
P3-50	Port 1 - Data Bit 31

Figure 5 - Port 1 Pinouts.

I/O Connector Pin	Signal
P2-1	Port 2 - Data Bit 0
P2-2	+5V
P2-3	Port 2 - Data Bit 1
P2-4	Ground
P2-5	Port 2 - Data Bit 2
P2-6	Ground
P2-7	Port 2 - Data Bit 3
P2-8	Ground
P2-9	Port 2 - Data Bit 4
P2-10	Ground
P2-11	Port 2 - Data Bit 5
P2-12	Ground
P2-13	Port 2 - Data Bit 6
P2-14	Ground
P2-15	Port 2 - Data Bit 7
P2-16	Port 2 - Data Bit 8
P2-17	Port 2 - Data Bit 9
P2-18	Ground
P2-19	Port 2 - Data Bit 10
P2-20	Port 2 - Data Bit 11
P2-21	Ground
P2-22	Port 2 - Data Bit 12
P2-23	Port 2 - Data Bit 13
P2-24	Ground
P2-25	Port 2 - Data Bit 14
P2-26	Port 2 - Data Bit 15
P2-27	Ground
P2-28	Port 2 - Data Bit 16
P2-29	Port 2 - Data Bit 17
P2-30	Ground
P2-31	Port 2 - Data Bit 18
P2-32	Port 2 - Data Bit 19
P2-33	Ground
P2-34	Port 2 - Data Bit 20
P2-35	Port 2 - Data Bit 21
P2-36	Ground
P2-37	Port 2 - Data Bit 22
P2-38	Port 2 - Data Bit 23
P2-39	Ground
P2-40	Port 2 - Data Bit 24
P2-41	Port 2 - Data Bit 25
P2-42	Ground
P2-43	Port 2 - Data Bit 26
P2-44	Port 2 - Data Bit 27
P2-45	Ground
P2-46	Port 2 - Data Bit 28
P2-47	Port 2 - Data Bit 29
P2-48	Ground
P2-49	Port 2 - Data Bit 30
P2-50	Port 2 - Data Bit 31

Figure 6 - Port 2 Pinouts.

I/O Connector Pin	Signal
P1-1	Port 3 - Data Bit 0
P1-2	+5V
P1-3	Port 3 - Data Bit 1
P1-4	Ground
P1-5	Port 3 - Data Bit 2
P1-6	Ground
P1-7	Port 3 - Data Bit 3
P1-8	Ground
P1-9	Port 3 - Data Bit 4
P1-10	Ground
P1-11	Port 3 - Data Bit 5
P1-12	Ground
P1-13	Port 3 - Data Bit 6
P1-14	Ground
P1-15	Port 3 - Data Bit 7
P1-16	Port 3 - Data Bit 8
P1-17	Port 3 - Data Bit 9
P1-18	Ground
P1-19	Port 3 - Data Bit 10
P1-20	Port 3 - Data Bit 11
P1-21	Ground
P1-22	Port 3 - Data Bit 12
P1-23	Port 3 - Data Bit 13
P1-24	Ground
P1-25	Port 3 - Data Bit 14
P1-26	Port 3 - Data Bit 15
P1-27	Ground
P1-28	Port 3 - Data Bit 16
P1-29	Port 3 - Data Bit 17
P1-30	Ground
P1-31	Port 3 - Data Bit 18
P1-32	Port 3 - Data Bit 19
P1-33	Ground
P1-34	Port 3 - Data Bit 20
P1-35	Port 3 - Data Bit 21
P1-36	Ground
P1-37	Port 3 - Data Bit 22
P1-38	Port 3 - Data Bit 23
P1-39	Ground
P1-40	Port 3 - Data Bit 24
P1-41	Port 3 - Data Bit 25
P1-42	Ground
P1-43	Port 3 - Data Bit 26
P1-44	Port 3 - Data Bit 27
P1-45	Ground
P1-46	Port 3 - Data Bit 28
P1-47	Port 3 - Data Bit 29
P1-48	Ground
P1-49	Port 3 - Data Bit 30
P1-50	Port 3 - Data Bit 31

Figure 7 - Port 3 Pinouts.

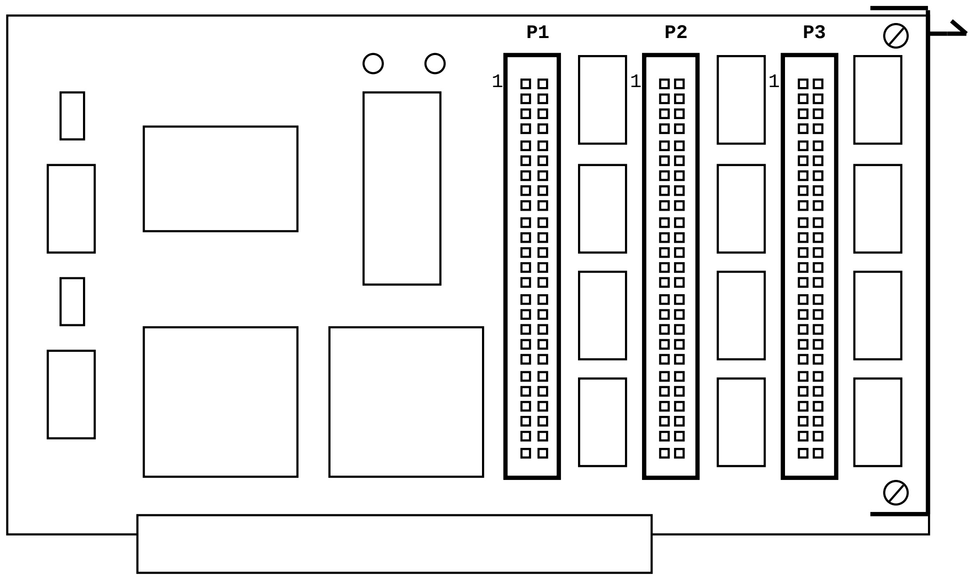


Figure 8 - Port connector locations.

Note that the I/O connector locations of the silkscreen of the board are labeled P1, P2, and P3. The documentation and the software refer to the ports as Port1, Port2, and Port3, however they correspond in reverse order. That is, Port 1 referenced in the documentation has its signals brought out on connector P3.

3.0 Software

3.1 Overview

Programming of the NBS-DIO96 card is facilitated by two software packages included with the card. The first is the 'Driver' code, which is a MAC 'Device Manager' compatible code segment. It is located on the PROM of the NBS-DIO96 card and is read by the MAC OS upon restart. The user may call the driver routines via any programming language which supports the MAC's device manager. As long as the calling parameters are followed as directed, the routines will execute regardless of what programming language was used to call them.

The second software package included, is a C++ class library which will take care of all of the lower-level details of calling the driver. The source code of the class library is included on a disk which is shipped with the NBS-DIO96 card.

3.2 Register Map

As stated in the hardware section of this manual, there are four memory mapped registers on the NBS-DIO96 card. All control of the card is done by means of long-word memory reads and writes to these registers. The driver routines for the card will need to be passed the address of the particular register being accessed when it is called.

The following mapping assignments are made on the card:

Register	**Address**
Port #1	$Fs000000
Port #2	$Fs000004
Port #3	$Fs000008
Control Reg.	$Fs00000C

Figure 9 - Register address locations.

All registers are accessed from the processor via long-word (32 bit) memory accesses. The control register however, only has 14 valid bits assigned within it. The other bits are undefined. The following bit assignments are made within the Control Register:

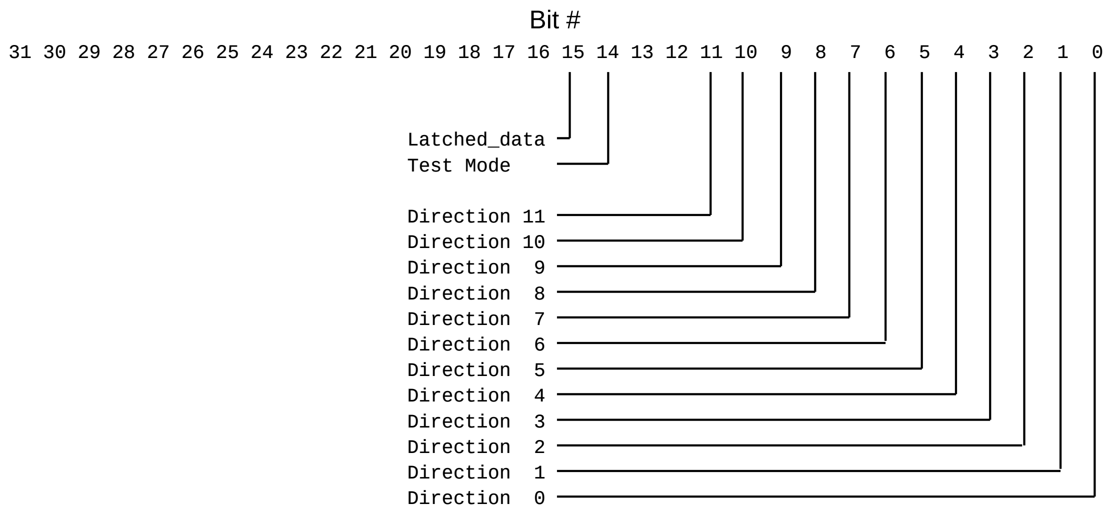


Figure 10 - Control Register Bit Mapping.

3.3 Driver Firmware

All interaction with the NBS-DIO96 card from the user's application code should be done via the device driver included with the card. This will assure the utmost compatibility with future operating system releases from Apple. Compatibility with the PowerPC MACs should also be assured. The device driver has only three routines which can be called from the user's program. They are the 'Open', 'Close', and 'Control' routines. The 'Control' routine has a selector code which is passed to it and it is via the value of this code that the driver will select either a register 'read' or a register 'write' operation. The 'Control' routine accomplishes most of the work in the user's program. The'Open' and 'Close' routines are simply called at the beginning and the end of your application to initialize the driver and then to clean up before your application quits.

The following data structure is used to communicate with the driver.

```
*****************************************************************************************
*       The following structure is used to pass data to and from the driver during all
*       Control calls to the driver.
*
*       struct dio96CtlBlk
*               {
*                       long    csVar;          // general purpose word has call specific
*                                               // data.  Refer to control call desired
*                                               // for variable definition.
*                       long    csFlag;         // general purpose word has call specific
*                                               // data.  Refer to control call desired
*                                               // for variable definition.
*                       short   csStatus;       // call returned status information
*                       short   csError;        // call returned error information
*                       Ptr     csAddr;         // pointer to an address on the dio96 card
*               };
*
*       typedef struct dio96CtlBlk dio96CtlBlk;
*       typedef dio96CtlBlk* dio96CtlBlkPtr;
*
*
*
*
csVar                   EQU     0               ; (long)     - call specific data
csFlag                  EQU     csVar+4         ; (long)     - call specific data
csStatus        EQU     csFlag+4                ; (word)     - returned driver status
csError                 EQU     csStatus+2      ; (word)     - returned error code
csAddr                  EQU     csError+2       ; Pointer to device card address
*
*****************************************************************************************
```

The source code listing of the driver is included with this documentation for the user's reference. It is further recommended that the user refer to the C++ class library included on the disk for examples of how to call the driver from your application program.

3.4 Cookbook

This section of the manual will take you thru an example program of how to call the routines in the driver for the NBS-DIO96 card. The example is almost trivial because of the fact that we are using the C++ class library for the card. The user may wish to translate the code from C++ to straight C or PASCAL if desired. All that is required is that the pass parameter conventions expected by the driver are adhered to. On with the example...

```
TDio96Handler*    myDio96Handler;   // the handler for the board
OSErr             err;
long              input_data;


// allocate the handler object and then open the driver
myDio96Handler = new TDio96Handler;
err = myDio96Handler->Dio96Open(9);       // use board slot #9
                                          // in this example

// set byte #1 port #1 for 'output' mode and all other bytes
// to 'input' mode.  The '1' bit sets the 'output' mode.
myDio96Handler->Dio96WrCntrlReg(0x00000001);

// output a value of hex '55' to the byte.  Remember about
// the NuBus byte-swapping mechanism so we need to put the
// value of 0x55 into bits 24-31.  This value will be output
// on connector pins P3-1,3,5,7,9,11,13,15.
myDio96Handler->Dio96WrDataReg1(0x55000000);

// read port #1 data register.  The input data will contain the
// value we just outputed (0x55) on bits 31-24.  The other data
// bits will contain the signal level of what is being driven by
// the user's equipment.  If nothing is connected to a signal pin
// the NBS-DIO96 card will return a '1' for that bit.  By default
// unconnected input pins on the NBs-DIO96 card float to a logic 1
input_data = myDio96Handler->Dio96RdDataReg1();

// all done so close the driver and free
err = myDio96Handler->Dio96Close();

myDio96Handler->Free();
myDio96Handler = nil;
```

Appendix A - dio96rom.a listing

```
* File dio96rom.a
*{Copyright © 1994 by fishcamp engineering.  All rights reserved.}

                             MACHINE       MC68020
                             STRING        C
                             PRINT         OFF
                             INCLUDE       'Traps.a'
                             INCLUDE       'ToolEqu.a'
                             INCLUDE       'SysEqu.a'
                             INCLUDE       'SysErr.a'
                             INCLUDE       'SlotEqu.a'
                             INCLUDE       'ROMEqu.a'

                             INCLUDE       'dio96incl.a'
                             PRINT         ON




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



dio96DeclRom         MAIN


;                    ORG           32768         ; XILINX configuration code is below this


**************************************************************************************************
*                            Directory
**************************************************************************************************
_sRsrcDir            OSLstEntry    sRsrc_Board,_sRsrc_Board     ; References the Board sResource
                     OSLstEntry    sRsrc_dio96,_sRsrc_dio96     ; References the dio96 sResource
                     DatLstEntry   EndOfList,0                  ; end of the list

**************************************************************************************************
*                            sRsrc_Board List
**************************************************************************************************
_sRsrc_Board         OSLstEntry    sRsrc_Type,_BoardType ; References the sResource type
                     OSLstEntry    sRsrc_Name,_BoardName ; References the sResource name
                     DatLstEntry   BoardId,dio96BoardId         ; the board Id
                     OSLstEntry    VendorInfo,_VendorInfo       ; references the vendor information list
                     DatLstEntry   EndOfList,0                  ; end of the list

_BoardType           DC.W          CatBoard                     ; the Board sResource: <Category>
                     DC.W          TypBoard                     ;                      <Type>
                     DC.W          DrSwBoard                    ;                      <DrvrSw>
                     DC.W          DrHwBoard                    ;                      <DrvrHw>
_BoardName           DC.L          'fishcamp engineering NBS-DIO96 card'; board's official product name


**************************************************************************************************
*                            Vendor info record
**************************************************************************************************
_VendorInfo          OSLstEntry    VendorId,_VendorId           ; references the vendor Id
                     OSLstEntry    RevLevel,_RevLevel           ; references the revision level
                     OSLstEntry    PartNum,_PartNum             ; references the part number
                     DatLstEntry   EndOfList,0                  ; end of the list

_VendorId            DC.L          'fishcamp engineering'       ; the vendor id
_RevLevel            DC.L          'Rev 1.0'                    ; the revision level
_PartNum             DC.L          'NBS-DIO96'                  ; the part number


**************************************************************************************************
*                            sRsrc_dio96
**************************************************************************************************
_sRsrc_dio96         OSLstEntry    sRsrc_Type,_dio96Type ; references the sResource type
                     OSLstEntry    sRsrc_Name,_dio96Name ; references the sResource name
                     OSLstEntry    sRsrc_DrvrDir,_dio96DrvrDir  ; references the driver directory
                     DatLstEntry   sRsrc_HWDevId,1              ; the hardware device Id
                     DatLstEntry   EndOfList,0                  ; end of the list

_dio96Type           DC.W          CatCommunication             ; dio96 sResource:    <Category>
                     DC.W          TypParallel                  ;                     <Type>
                     DC.W          DrSwFishcampEngr             ;                     <DrvrSw>
                     DC.W          DrHwNBS_DI096                ;                    <DrvrHw>

_dio96Name           DC.L          'digital96_fishcamp_NBS-DIO96'

*************************************************************************************************
*                           driver directory
*************************************************************************************************
_dio96DrvrDir        OSLstEntry    sMacOS68020,_sMacOS68020     ; references the Macintosh-OS 68020
                     DatLstEntry   EndOfList,0                  ; end of the list

; Driver-1 (68020)
_sMacOS68020         DC.L          _End020Drvr-_sMacOS68020     ; the physical block size
                     INCLUDE'dio96drvr.a'                       ; the header/code
_End020Drvr          EQU           *                            ; the end of the driver
                     STRING        C




                     ORG           ROMSize-fhBlockSize

*************************************************************************************************
*                           format/header block
*************************************************************************************************
                     DC.L          (_sRsrcDir-*)**$00ffffff     ; offset to sResource directory
                     DC.L          ROMSize                      ; length of declaration data
                     DC.L          0                            ; CRC (Patched by crcPatch, an MPW tool
                     DC.B          Rev1                         ; revision level
                     DC.B          AppleFormat                  ; format
                     DC.L          TestPattern                  ; test pattern
                     DC.B          0                            ; Reserved byte (must be zero)
                     DC.B          $78                          ; Byte lanes: 0111 1000 (bytelane 3)


                     ENDMAIN

                     END
```

Appendix B - dio96incl.a listing

```
*       Version 1.0   15 April, 1994



* File dio96incl.a
*{Copyright © 1994 by fishcamp engineering.  All rights reserved.}




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


*
* The following define addresses of valid locations on the NBS-DIO96 card.
* They are defined as offsets from the board's base address.  Accesses to the
* PROM on the card are made with byte wide accesses while all other addresses
* on the card are made with long word (32 bit) references.
*
* The three data registers on the card correspond to the three ports of the card.
* When the data direction for a register is set for OUTPUT any data written to
* these registers will appear on the output connector pins for the port.  Likewise,
* when the data direction mode is set to INPUT, the signal levels on the connector
* pins can be input by reading the port's data register.  Actually, the design
* of the NBS-DIO96 card is such that an INPUT operation can be performed at any time.
* The rEAD operation will correctly return the pin signal levels regardless of
* who is driving the signal line (the customers circuitry or the NBS-DIO96 card's
* output drivers).
*
* The CONTROL register is used to select whether or not the data in the card's output
* register is allowed to drive the connector pins for the particular port.  Data
* direction control is defineable on a byte by byte basis in each of the ports.
* The NBS-DIO96 card has three ports.  Each port is 32-bits wide with each of the four
* bytes within the port having a dedicated data directrion control bit.  Once the
* data direction control bit for a particular byte is set by the user, ALL 8-bits within
* that byte assume the new direction as programmed.  There are a total of 12 data direction
* control bits provided for in the NBS-DIO96 card's control register.  One for each of the
* four bytes in each of the three ports on the card.  These bits are assigned to the
* least significant 12 bits in the control register.  Upon power-up reset or when the
* driver for the NBS-DIO96 card is opened, these bits are cleared.  The reset state of
* any particular bit selects the INPUT mode for its respective port.  A '1' programmed
* into the data direction bit for a port will enable the port's output register drivers
* to allow the NBS-DIO96 card to drive the port lines.
*
*
*       cntrlRegAddr bit  0 - undefined
*       cntrlRegAddr bit  1 - undefined
*       cntrlRegAddr bit  2 - undefined
*       cntrlRegAddr bit  3 - undefined
*       cntrlRegAddr bit  4 - undefined
*       cntrlRegAddr bit  5 - undefined
*       cntrlRegAddr bit  6 - undefined
*       cntrlRegAddr bit  7 - undefined
*       cntrlRegAddr bit  8 - undefined
*       cntrlRegAddr bit  9 - undefined
*       cntrlRegAddr bit 10 - undefined
*       cntrlRegAddr bit 11 - undefined
*       cntrlRegAddr bit 12 - undefined
*       cntrlRegAddr bit 13 - undefined
*       cntrlRegAddr bit 14 - undefined
*       cntrlRegAddr bit 15 - undefined
*       cntrlRegAddr bit 16 - DDR bit  0 , port A, byte 0 (LSB)
*       cntrlRegAddr bit 17 - DDR bit  1 , port A, byte 1
*       cntrlRegAddr bit 18 - DDR bit  2 , port A, byte 2
*       cntrlRegAddr bit 19 - DDR bit  3 , port A, byte 3 (MSB)
*       cntrlRegAddr bit 20 - DDR bit  4 , port B, byte 0 (LSB)
*       cntrlRegAddr bit 21 - DDR bit  5 , port B, byte 1
*       cntrlRegAddr bit 22 - DDR bit  6 , port B, byte 2
*       cntrlRegAddr bit 23 - DDR bit  7 , port B, byte 3 (MSB)
*       cntrlRegAddr bit 24 - DDR bit  8 , port C, byte 0 (LSB)
*       cntrlRegAddr bit 25 - DDR bit  9 , port C, byte 1
*       cntrlRegAddr bit 26 - DDR bit 10 , port C, byte 2
*       cntrlRegAddr bit 27 - DDR bit 11 , port C, byte 3 (MSB)
*       cntrlRegAddr bit 28 - undefined
*       cntrlRegAddr bit 29 - undefined
*       cntrlRegAddr bit 30 - undefined
*       cntrlRegAddr bit 31 - Latch data stobe disable ( default = 0, enabled strobe)
*

dataReg1Addr   EQU    $000000; first data register
dataReg2Addr   EQU    $000004; second data register
```

```
dataReg3Addr  EQU    $000008; third data register
cntrlRegAddr  EQU    $00000C; control register

romaddrEQU    $fd0000; 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.
*
*      struct dio96CtlBlk
*      {
*      long   csVar;                // general purpose word has call specific
*                                   // data.  Refer to control call desired
*                                   // for variable definition.
*      long   csFlag;               // general purpose word has call specific
*                                   // data.  Refer to control call desired
*                                   // for variable definition.
*      short  csStatus;             // call returned status information
*      short  csError;              // call returned error information
*      Ptr    csAddr;               // pointer to an address on the dio96 card
*      };
*
*      typedef struct dio96CtlBlk dio96CtlBlk;
*      typedef dio96CtlBlk* dio96CtlBlkPtr;
*
*
*
*
csVar         EQU    0              ; (long)      - call specific data
csFlag        EQU    csVar+4        ; (long)      - call specific data
csStatus      EQU    csFlag+4       ; (word)      - returned driver status
csErrorEQU    csStatus+2            ; (word)      - returned error code
csAddr        EQU    csError+2      ; Pointer to device card address
*
*****************************************************************************************



*      Control call operating system Error codes
dio96Err      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
stCmpltEQU    $0100                 ; I/O operation completed during call


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


dio96BoardId         EQU    $0647                        ; As assigned by Apple DTS
CatCommunication     EQU    $0006                        ;
TypParallel          EQU    $0003                        ;
DrSwFishcampEngr     EQU    $0101                        ;
DrHwNBS_DI096        EQU    $0101                        ;


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

ROMSize       EQU    32768                               ; size of on-board ROM
;ROMSize             EQU    8192                         ; size of on-board ROM
;ROMSize             EQU    65536                        ; 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_dio96          EQU    128                          ; dio96 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
```

Appendix C - dio96drvr.a listing

```
*      Version 1.0   15 April, 1994


* File dio96drvr.a
*{Copyright © 1989-1994 by fishcamp engineering.  All rights reserved.}



                      BLANKS        ON
                      STRING        ASIS




*********************************************************************************************
*                            local vars, definitions etc.
*********************************************************************************************





;---------------------------------------------
; Write the specified LONG to the specified NuBus address.
; The address actually used will be the address specified
; added to the board's base address which should be contained in A1.
; The board's base address for slot 9 would be $f9000000.
;
;      Call:                A1 - board base address
;
;      Registers affected:  None
;
                      MACRO
                      MWrite        &Address,&Data
                      MOVEM.LD0/A0,-(SP)                          ; save work registers

                      MOVE.L        A1,D0                         ; from board base address
                      ADD.L         #&Address,D0                  ; add to where byte will go
                      MOVEA.LD0,A0                                ; A0 has address
                      MOVE.L        #&Data,D0                     ; set data
                      BSR           NbWrite

                      MOVEM.L(SP)+,D0/A0                          ; restore registers
                      ENDM
;---------------------------------------------











*********************************************************************************************
*                            dio96 driver header
*********************************************************************************************

dio96Drvr             DC.W          $4400                         ; ctl,needslock
                      DC.W          0,0,0                         ; not used but required values

; Entry point offset table

                      DC.W          dio96Open-dio96Drvr           ; open
                      DC.W          dio96Drvr-dio96Drvr           ; no prime
                      DC.W          dio96Ctl-dio96Drvr            ; control
                      DC.W          dio96Drvr-dio96Drvr           ; no status
                      DC.W          dio96Close-dio96Drvr          ; close

                      STRING        Pascal
dio96Title            DC.B          '.Fc_dio96'                   ; driver name
                      STRING        ASIS
                      ALIGN         2                             ; force alignment
                      DC.W          0                             ; version - 0
```

```
*********************************************************************************************
*      dio96Open initializes registers, sets INPUT mode.
*
*      Entry:        A0 - param blk pointer
*                    A1 - DCE pointer
*
*      Locals:A0 - temporary
*                    A1 - board base address
*                    A2 - Saved param block pointer
*                    A3 - Saved DCE pointer
*                    D0 - temporary
*                    D1 - temporary
*********************************************************************************************
; save registers
dio96Open
                     MOVE.L        A0,A2                 ; A2 <- param block pointer
                     MOVE.L        A1,A3                 ; A3 <- DCE pointer


; get base address of board
                     MOVE.B        dCtlSlot(A3),D0       ; get the slot address
                     LSL.L         #8,D0                 ; shift the 4 slot bits into proper position
                     LSL.L         #8,D0
                     LSL.L         #8,D0
                     ORI.L         #$f0000000,D0         ; Slot space
                     MOVEA.LD0,A1                        ; A1 = board base address

                     MWrite        cntrlRegAddr,0        ; make sure the data direction register bits
                                                         ; are set for INPUT and the latch data strobe
                                                         ; disable bits are cleared.
                     MWrite        dataReg1Addr,0        ; default output data for port A
                     MWrite        dataReg2Addr,0        ; default output data for port B
                     MWrite        dataReg3Addr,0        ; default output data for port C

                     MOVE.W        #noErr,ioResult(A2)   ; flag no error
                     MOVEQ         #noErr,D0;
                     BRA.S         EndOpen

; error exit
OpError              MOVE.W        #openErr,ioResult(A2); couldn't open driver

EndOpen              RTS                                 ; return
```

```
*********************************************************************************************
*      dio96Close - simply exit
*
*      Entry:        A0 - param blk pointer
*                    A1 - DCE pointer
*
*      Locals:A0 - temporary
*                    A1 - board base address
*                    A2 - Saved param block pointer
*                    A3 - Saved DCE pointer
*                    D0 - temporary
*
*      Return:D0 - error
*********************************************************************************************
dio96Close
                     MOVE.L        A0,A2                 ; A2 <- param block pointer
                     MOVE.L        A1,A3                 ; A3 <- DCE pointer


; get base address of board
                     MOVE.B        dCtlSlot(A3),D0       ; get the slot address
                     LSL.L         #8,D0                 ; shift the 4 slot bits into proper position
                     LSL.L         #8,D0
                     LSL.L         #8,D0
                     ORI.L         #$f0000000,D0         ; Slot space
                     MOVEA.L       D0,A1                 ; A1 = board base address

;
```

```
;      do nothing, and exit
;

                     MOVEQ         #noErr,D0            ; get error into D0
                     RTS                                ; return to caller




*******************************************************************************************
*      dio96Ctl control call handler. 3 different calls:
*
*             (1)    KillIo
*             (25)   Read;
*             (26)   Write;
*
*      Entry:        A0    - param blk pointer
*                    A1    - DCE pointer
*
*      Uses:         A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
*                    A3    - scratch ( doesn't need to be preserved )
*                    A4    - scratch ( must be preserved )
*                    D0-D3 - scratch ( doesn't need to be preserved )
*
*      Exit:         D0    - error code
*
*******************************************************************************************
; decode the requested call
dio96Ctl
                     MOVEM.L       A0/A4/D4,-(SP); save work registers (A0 is saved
                                                ; because it is used by ExitDrvr)


                     MOVE.W        csCode(A0),D0        ; get the opcode
                     MOVE.L        csParam(A0),A2; A2 <- Ptr to control parameters

                     CMP.W         #26,D0               ; IF csCode NOT IN [0..26] THEN
                     BHI.S         CtlBad               ; error, csCode out of bounds
                     LSL.W         #1,D0                ; Adjust csCode to be an index into the table
                     MOVE.W        CtlJumpTbl(PC,D0.W),D0 ; Get the relative offset to the routine
                     JMP           CtlJumpTbl(PC,D0.W)  ; GOTO the proper routine

CtlJumpTbl           DC.W          NoSupport-CtlJumpTbl ; no support                       0
                     DC.W          CtlGood-CtlJumpTbl   ; KillIo                           1
                     DC.W          NoSupport-CtlJumpTbl ; no support                       2
                     DC.W          NoSupport-CtlJumpTbl ; no support                       3
                     DC.W          NoSupport-CtlJumpTbl ; no support                       4
                     DC.W          NoSupport-CtlJumpTbl ; no support                       5
                     DC.W          NoSupport-CtlJumpTbl ; no support                       6
                     DC.W          NoSupport-CtlJumpTbl ; no support                       7
                     DC.W          NoSupport-CtlJumpTbl ; no support                       8
                     DC.W          NoSupport-CtlJumpTbl ; no support                       9
                     DC.W          NoSupport-CtlJumpTbl ; no support                       10
                     DC.W          NoSupport-CtlJumpTbl ; no support                       11
                     DC.W          NoSupport-CtlJumpTbl ; no support                       12
                     DC.W          NoSupport-CtlJumpTbl ; no support                       13
                     DC.W          NoSupport-CtlJumpTbl ; no support                       14
                     DC.W          NoSupport-CtlJumpTbl ; no support                       15
                     DC.W          NoSupport-CtlJumpTbl ; no support                       16
                     DC.W          NoSupport-CtlJumpTbl ; no support                       17
                     DC.W          NoSupport-CtlJumpTbl ; no support                       18
                     DC.W          NoSupport-CtlJumpTbl ; no support                       19
                     DC.W          NoSupport-CtlJumpTbl ; no support                       20
                     DC.W          NoSupport-CtlJumpTbl ; no support                       21
                     DC.W          NoSupport-CtlJumpTbl ; no support                       22
                     DC.W          NoSupport-CtlJumpTbl ; no support                       23
                     DC.W          NoSupport-CtlJumpTbl ; no support                       24
                     DC.W          Read-CtlJumpTbl      ; Read byte from board memory      25
                     DC.W          Write-CtlJumpTbl     ; Write byte from board memory     26

CtlBad               MOVEQ         #controlErr,D0       ; say we don't do this one
                     BRA.S         CtlDone              ; and return

CtlGood              MOVEQ         #noErr,D0            ; return no error

CtlDone              MOVEM.L       (SP)+,A0/A4/D4       ; restore registers
                     BRA.S         ExitDrvr

                     NOP
```

```
*********************************************************************************************
*      Exit from control calls
*********************************************************************************************

ExitDrvr            BTST          #NoQueueBit,ioTrap(A0); no queue bit set ?
                    BEQ.S         GoIODone             ; => no, not immediate
                    RTS                                ; otherwise, it was an immediate call

GoIODone            MOVE.L        JIODone,A0           ; get IODone address
                    JMP           (A0)                 ; invoke it




*********************************************************************************************
*      NoSupport - control call not supported
*
*      Entry:       A0 - param blk pointer
*                   A1 - DCE pointer
*                   A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
*
*********************************************************************************************
NoSupport
                    MOVEM.L       A1/D0,-(SP)          ; save local work registers

; get base address of board
                    MOVE.B        dCtlSlot(A1),D0      ; get the slot address
                    LSL.L         #8,D0                ; shift the 4 slot bits into proper position
                    LSL.L         #8,D0
                    LSL.L         #8,D0
                    ORI.L         #$f0000000,D0        ; Slot space
                    MOVEA.L       D0,A1                ; A1 = board base address

NoSprt1             MOVEQ         #controlErr,D0       ; say we don't do this one

NoSprtDone          MOVEM.L       (SP)+,A1/D0          ; restore local registers
                    MOVEM.L       (SP)+,A0/A4/D4       ; restore registers
                    BRA.S         ExitDrvr



*********************************************************************************************
*      Read - read LONG from board memory
*
*      Entry:       A0 - param blk pointer
*                   A1 - DCE pointer
*                   A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
*
*********************************************************************************************
Read
                    MOVEM.L       A0/A1/D0/D1,-(SP)    ; save local work registers

; get base address of board
                    MOVE.B        dCtlSlot(A1),D0      ; get the slot address
                    LSL.L         #8,D0                ; shift the 4 slot bits into proper position
                    LSL.L         #8,D0
                    LSL.L         #8,D0
                    ORI.L         #$f0000000,D0        ; Slot space
                    MOVEA.L       D0,A1                ; A1 = board base address

; get address
                    MOVE.L        A2,D0
                    ADD.L         #csAddr,D0
                    MOVEA.L       D0,A0
                    MOVE.L        (A0),D0              ; D0 = address on board
                    ANDI.L        #$00ffffff,D0        ; mask to 24 bits
                    MOVE.L        A1,D1                ; get board base address
                    ADD.L         D1,D0                ; add it to the requested address
                    MOVEA.L       D0,A0                ; A0 = requested address on board
                    BSR.S         NbRead               ; get the byte
                    MOVE.L        D0,csVar(A2)         ; Return to caller the requested byte.

ReadGood            MOVEQ         #noErr,D0            ; return no error
                    MOVE.W        #ctlNoErr,csError(A2)
                    MOVE.W        #stGood,csStatus(A2) ; Default status
                    ORI.W         #stCmplt,csStatus(A2) ; flag call complete

ReadDone            MOVEM.L       (SP)+,A0/A1/D0/D1    ; restore local registers
                    MOVEM.L       (SP)+,A0/A4/D4       ; restore registers
                    BRA.S         ExitDrvr
```

```
******************************************************************************************
*      Write - write LONG to board memory
*
*      Entry:         A0 - param blk pointer
*                     A1 - DCE pointer
*                     A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
*
******************************************************************************************
Write
                      MOVEM.L       A0/A1/D0/D1,-(SP)     ; save local work registers

; get base address of board
                      MOVE.B        dCtlSlot(A1),D0       ; get the slot address
                      LSL.L         #8,D0                 ; shift the 4 slot bits into proper position
                      LSL.L         #8,D0
                      LSL.L         #8,D0
                      ORI.L         #$f0000000,D0         ; Slot space
                      MOVEA.L       D0,A1                 ; A1 = board base address

; get address
                      MOVE.L        A2,D0
                      ADD.L         #csAddr,D0
                      MOVEA.L       D0,A0
                      MOVE.L        (A0),D0               ; D0 = address on board
                      ANDI.L        #$00ffffff,D0         ; mask to 24 bits
                      MOVE.L        A1,D1                 ; get board base address
                      ADD.L         D1,D0                 ; add it to the requested address
                      MOVEA.L       D0,A0                 ; A0 = requested address on board
                      MOVE.L        csVar(A2),D0          ; D0 contains the LONG to be written
                      BSR.S         NbWrite               ; write the byte

WriteGood             MOVEQ         #noErr,D0             ; return no error
                      MOVE.W        #ctlNoErr,csError(A2)
                      MOVE.W        #stGood,csStatus(A2)  ; Default status
                      ORI.W         #stCmplt,csStatus(A2) ; flag call complete

WriteDone             MOVEM.L       (SP)+,A0/A1/D0/D1     ; restore local registers
                      MOVEM.L       (SP)+,A0/A4/D4        ; restore registers
                      BRA           ExitDrvr




******************************************************************************************
*      NbRead - reads a LONG from a NUBUS card
*
*      Enter:         A0 - pointer to address in 32-bit address space
*
*      Uses:          no other registers
*
*      Exit:          D0 - the long word read
*
******************************************************************************************
NbRead
                      MOVEM.L       D1,-(SP)              ; save work register

                      MOVE.L        true32b,D0            ; set 32-bit mode
                      _SwapMMUMode

                      MOVE.L        (A0),D1               ; Get value specified

                      MOVE.L        false32b,D0
                      _SwapMMUMode                        ; back to 24-bit mode

                      MOVE.L        D1,D0                 ; return value
                      MOVEM.L       (SP)+,D1              ; restore work register
                      RTS

******************************************************************************************
*      NbWrite - writes a LONG to a NUBUS card
*
*      Enter:         A0 - pointer to address in 32-bit address space
*                     D0 - the long word to write
*
*      Uses:          no other registers
*
******************************************************************************************
NbWrite
                      MOVEM.L       D1,-(SP)              ; save work registers
```

```
MOVE.L        D0,D1                ; save value in D1

MOVE.L        true32b,D0           ; set 32-bit mode
_SwapMMUMode

MOVE.L        D1,(A0)              ; write value specified

MOVE.L        false32b,D0
_SwapMMUMode                       ; back to 24-bit mode

MOVE.L        D1,D0                ; restore entry value
MOVEM.L       (SP)+,D1             ; restore work register
RTS
```

Appendix D - dio96rom.a.lst listing

```
MC680xx Assembler - Ver 3.3.1                                                              12-Jun-94
Page   1
Copyright Apple Computer, Inc. 1984-1991

Loc   F Object Code     Addr  M Source Statement

                                * File dio96rom.a
                                *{Copyright © 1994 by fishcamp engineering.  All rights reserved.}

                                               MACHINE       MC68020
                                               STRING        C
                                               PRINT         ON




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

    **********************************************************************************************



00000                           dio96DeclRom   MAIN
00000
00000
00000                           ;              ORG           32768             ; XILINX configuration code is below this
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_dio96,_sRsrc_dio96     ; References the dio96
sResource
00004   8000 0084             1                DC.L       (sRsrc_dio96<<24)++((_sRsrc_dio96-*)**$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,dio96BoardId     ; the board Id
00014   2000 0647             1                DC.L       (BoardId<<24)+dio96BoardId
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-DIO96 card'   ; board's official
product name
0004C
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-DIO96'                  ; the part number
00088
00088
00088
    **********************************************************************************************
```

```
Loc   F Object Code     Addr  M  Source Statement

00088                             *             sRsrc_dio96
00088
   *********************************************************************************************
00088                             _sRsrc_dio96  OSLstEntry    sRsrc_Type,_dio96Type       ; references the sResource
type
00088   0100 0014             1                 DC.L       (sRsrc_Type<<24)++((_dio96Type-*)**$00FFFFFF)
0008C                                           OSLstEntry    sRsrc_Name,_dio96Name       ; references the sResource
name
0008C   0200 0018             1                 DC.L       (sRsrc_Name<<24)++((_dio96Name-*)**$00FFFFFF)
00090                                           OSLstEntry    sRsrc_DrvrDir,_dio96DrvrDir  ; references the driver
directory
00090   0400 0034             1                 DC.L       (sRsrc_DrvrDir<<24)++((_dio96DrvrDir-*)**$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   0006                      _dio96Type    DC.W          CatCommunication            ; dio96 sResource:<Category>
0009E   0003                                    DC.W          TypParallel                 ;                 <Type>
000A0   0101                                    DC.W          DrSwFishcampEngr            ;                 <DrvrSw>
000A2   0101                                    DC.W          DrHwNBS_DI096               ;                 <DrvrHw>
000A4
000A4   6469676974616C            _dio96Name    DC.L          'digital96_fishcamp_NBS-DIO96'
000C4
000C4
   *********************************************************************************************
000C4                             *             driver directory
000C4
   *********************************************************************************************
000C4                             _dio96DrvrDir 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
000CC                             ; Driver-1 (68020)
000CC   0000 021C                 _sMacOS68020  DC.L          _End020Drvr-_sMacOS68020     ; the physical block size
000D0                                           INCLUDE       'dio96drvr.a'                ; the header/code
000D0                             *  Version 1.0   15 April, 1994
000D0
000D0
000D0
000D0                             * File dio96drvr.a
000D0                             *{Copyright © 1989-1994 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 LONG 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 would be $f9000000.
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
```

```
Loc   F Object Code     Addr  M Source Statement

000D0                                          MOVE.L     #&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                           *              dio96 driver header
000D0
   *********************************************************************************************
000D0
000D0   4400                    dio96Drvr      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       dio96Open-dio96Drvr   ; open
000DA   0000                                   DC.W       dio96Drvr-dio96Drvr   ; no prime
000DC   00B6                                   DC.W       dio96Ctl-dio96Drvr    ; control
000DE   0000                                   DC.W       dio96Drvr-dio96Drvr   ; no status
000E0   009C                                   DC.W       dio96Close-dio96Drvr; close
000E2
000E2                                          STRING     Pascal
000E2   092E46635F6469          dio96Title     DC.B       '.Fc_dio96'          ; 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                           *   dio96Open initializes registers, sets INPUT mode.
000EE                           *
000EE                           *   Entry:     A0 - param blk pointer
000EE                           *              A1 - DCE pointer
000EE                           *
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                           dio96Open
000EE   2448                                   MOVE.L     A0,A2            ; A2 <- param block pointer
000F0   2649                                   MOVE.L     A1,A3            ; A3 <- DCE pointer
000F2
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 0000                         ORI.L      #$f0000000,D0    ; Slot space
00102   2240                                   MOVEA.L    D0,A1            ; A1 = board base address
00104
00104                                          MWrite     cntrlRegAddr,0   ; make sure the data direction register bits
00104   48E7 8080            1                 MOVEM.L    D0/A0,-(SP)      ; save work registers
00108                        1
```

```
Loc   F Object Code     Addr  M Source Statement

00108   2009                  1              MOVE.L     A1,D0                ; from board base address
0010A   0680 0000 000C        1              ADD.L      #cntrlRegAddr,D0     ; add to where byte will go
00110   2040                  1              MOVEA.L    D0,A0                ; A0 has address
00112 G 7000                  1              MOVE.L     #0,D0                ; set data
00114   6100 01B6       002CC 1              BSR        NbWrite
00118                         1
00118   4CDF 0101             1              MOVEM.L    (SP)+,D0/A0          ; restore registers
0011C                                                                        ; are set for INPUT and the latch data
strobe
0011C                                                                        ; disable bits are cleared.
0011C                                        MWrite     dataReg1Addr,0       ; default output data for port A
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   0680 0000 0000        1              ADD.L      #dataReg1Addr,D0     ; add to where byte will go
00128   2040                  1              MOVEA.L    D0,A0                ; A0 has address
0012A G 7000                  1              MOVE.L     #0,D0                ; set data
0012C   6100 019E       002CC 1              BSR        NbWrite
00130                         1
00130   4CDF 0101             1              MOVEM.L    (SP)+,D0/A0          ; restore registers
00134                                        MWrite     dataReg2Addr,0       ; default output data for port B
00134   48E7 8080             1              MOVEM.L    D0/A0,-(SP)          ; save work registers
00138                         1
00138   2009                  1              MOVE.L     A1,D0                ; from board base address
0013A G 5880                  1              ADD.L      #dataReg2Addr,D0     ; add to where byte will go
0013C   2040                  1              MOVEA.L    D0,A0                ; A0 has address
0013E G 7000                  1              MOVE.L     #0,D0                ; set data
00140   6100 018A       002CC 1              BSR        NbWrite
00144                         1
00144   4CDF 0101             1              MOVEM.L    (SP)+,D0/A0          ; restore registers
00148                                        MWrite     dataReg3Addr,0       ; default output data for port C
00148   48E7 8080             1              MOVEM.L    D0/A0,-(SP)          ; save work registers
0014C                         1
0014C   2009                  1              MOVE.L     A1,D0                ; from board base address
0014E G 5080                  1              ADD.L      #dataReg3Addr,D0     ; add to where byte will go
00150   2040                  1              MOVEA.L    D0,A0                ; A0 has address
00152 G 7000                  1              MOVE.L     #0,D0                ; set data
00154   6100 0176       002CC 1              BSR        NbWrite
00158                         1
00158   4CDF 0101             1              MOVEM.L    (SP)+,D0/A0          ; restore registers
0015C
0015C G 426A 0010                            MOVE.W     #noErr,ioResult(A2)  ; flag no error
00160   7000                                 MOVEQ      #noErr,D0;
00162   6006            0016A                BRA.S      EndOpen
00164
00164                           ; error exit
00164   357C FFE9 0010          OpError      MOVE.W     #openErr,ioResult(A2) ; couldn't open driver
0016A
0016A   4E75                    EndOpen      RTS                             ; return
0016C
0016C
0016C
0016C
0016C
0016C
0016C
0016C
0016C
0016C
0016C
0016C
   ******************************************************************************************
0016C                           *   dio96Close - simply exit
0016C                           *
0016C                           *   Entry:     A0 - param blk pointer
0016C                           *              A1 - DCE pointer
0016C                           *
0016C                           *   Locals:    A0 - temporary
0016C                           *              A1 - board base address
0016C                           *              A2 - Saved param block pointer
0016C                           *              A3 - Saved DCE pointer
0016C                           *              D0 - temporary
0016C                           *
0016C                           *   Return:    D0 - error
0016C
   ******************************************************************************************
0016C                           dio96Close
```

```
Loc    F Object Code      Addr  M  Source Statement

0016C    2448                                          MOVE.L     A0,A2               ; A2 <- param block pointer
0016E    2649                                          MOVE.L     A1,A3               ; A3 <- DCE pointer
00170
00170
00170                             ; get base address of board
00170    102B 0028                                     MOVE.B     dCtlSlot(A3),D0     ; get the slot address
00174    E188                                          LSL.L      #8,D0               ; shift the 4 slot bits into proper position
00176    E188                                          LSL.L      #8,D0
00178    E188                                          LSL.L      #8,D0
0017A    0080 F000 0000                                ORI.L      #$f0000000,D0       ; Slot space
00180    2240                                          MOVEA.L    D0,A1               ; A1 = board base address
00182
00182                             ;
00182                             ;   do nothing, and exit
00182                             ;
00182
00182    7000                                          MOVEQ      #noErr,D0           ; get error into D0
00184    4E75                                          RTS                            ; return to caller
00186
00186
00186
00186
00186
00186
   *******************************************************************************************
00186                             *   dio96Ctl control call handler. 3 different calls:
00186                             *
00186                             *       (1)    KillIo
00186                             *       (25)   Read;
00186                             *       (26)   Write;
00186                             *
00186                             *   Entry:     A0    - param blk pointer
00186                             *              A1    - DCE pointer
00186                             *
00186                             *   Uses:      A2    - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00186                             *              A3    - scratch ( doesn't need to be preserved )
00186                             *              A4    - scratch ( must be preserved )
00186                             *              D0-D3 - scratch ( doesn't need to be preserved )
00186                             *
00186                             *   Exit:      D0    - error code
00186                             *
00186
   *******************************************************************************************
00186                             ; decode the requested call
00186                             dio96Ctl
00186    48E7 0888                                     MOVEM.L    A0/A4/D4,-(SP)      ; save work registers (A0 is saved
0018A                                                                                 ; because it is used by ExitDrvr)
0018A
0018A
0018A    3028 001A                                     MOVE.W     csCode(A0),D0       ; get the opcode
0018E    2468 001C                                     MOVE.L     csParam(A0),A2      ; A2 <- Ptr to control parameters
00192
00192    0C40 001A                                     CMP.W      #26,D0              ; IF csCode NOT IN [0..26] THEN
00196    6240            001D8                         BHI.S      CtlBad              ; error, csCode out of bounds
00198    E348                                          LSL.W      #1,D0               ; Adjust csCode to be an index into the table
0019A    303B 0006       001A2                         MOVE.W     CtlJumpTbl(PC,D0.W),D0; Get the relative offset to the routine
0019E    4EFB 0002       001A2                         JMP        CtlJumpTbl(PC,D0.W)   ; GOTO the proper routine
001A2
001A2    0054                     CtlJumpTbl           DC.W       NoSupport-CtlJumpTbl  ; no support                        0
001A4    003A                                          DC.W       CtlGood-CtlJumpTbl    ; KillIo                            1
001A6    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        2
001A8    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        3
001AA    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        4
001AC    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        5
001AE    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        6
001B0    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        7
001B2    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        8
001B4    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        9
001B6    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        10
001B8    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        11
001BA    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        12
001BC    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        13
001BE    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        14
001C0    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        15
001C2    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        16
001C4    0054                                          DC.W       NoSupport-CtlJumpTbl  ; no support                        17
```

```
Loc   F Object Code     Addr  M Source Statement

001C6   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         18
001C8   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         19
001CA   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         20
001CC   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         21
001CE   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         22
001D0   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         23
001D2   0054                                  DC.W      NoSupport-CtlJumpTbl  ; no support                         24
001D4   0076                                  DC.W      Read-CtlJumpTbl       ; Read byte from board memory           25
001D6   00C2                                  DC.W      Write-CtlJumpTbl      ; Write byte from board memory          26
001D8
001D8   70EF                    CtlBad        MOVEQ     #controlErr,D0        ; say we don't do this one
001DA   6002            001DE                 BRA.S     CtlDone               ; and return
001DC
001DC   7000                    CtlGood       MOVEQ     #noErr,D0             ; return no error
001DE
001DE   4CDF 1110               CtlDone       MOVEM.L   (SP)+,A0/A4/D4        ; restore registers
001E2   6002            001E6                 BRA.S     ExitDrvr
001E4
001E4   4E71                                  NOP
001E6
001E6
001E6
001E6
001E6
001E6
   ********************************************************************************************
001E6                           *   Exit from control calls
001E6
   ********************************************************************************************
001E6
001E6   0828 0009 0006          ExitDrvr      BTST      #NoQueueBit,ioTrap(A0)   ; no queue bit set ?
001EC   6702            001F0                 BEQ.S     GoIODone                 ; => no, not immediate
001EE   4E75                                  RTS                                ; otherwise, it was an immediate call
001F0
001F0   2078 08FC               GoIODone      MOVE.L    JIODone,A0               ; get IODone address
001F4   4ED0                                  JMP       (A0)                     ; invoke it
001F6
001F6
001F6
001F6
001F6
001F6
   ********************************************************************************************
001F6                           *   NoSupport - control call not supported
001F6                           *
001F6                           *   Entry:     A0 - param blk pointer
001F6                           *              A1 - DCE pointer
001F6                           *              A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
001F6                           *
001F6
   ********************************************************************************************
001F6                           NoSupport
001F6   48E7 8040                             MOVEM.L   A1/D0,-(SP)        ; save local work registers
001FA
001FA                           ; get base address of board
001FA   1029 0028                             MOVE.B    dCtlSlot(A1),D0    ; get the slot address
001FE   E188                                  LSL.L     #8,D0              ; shift the 4 slot bits into proper position
00200   E188                                  LSL.L     #8,D0
00202   E188                                  LSL.L     #8,D0
00204   0080 F000 0000                        ORI.L     #$f0000000,D0      ; Slot space
0020A   2240                                  MOVEA.L   D0,A1              ; A1 = board base address
0020C
0020C   70EF                    NoSprt1       MOVEQ     #controlErr,D0     ; say we don't do this one
0020E
0020E   4CDF 0201               NoSprtDone    MOVEM.L   (SP)+,A1/D0        ; restore local registers
00212   4CDF 1110                             MOVEM.L   (SP)+,A0/A4/D4     ; restore registers
00216   60CE            001E6                 BRA.S     ExitDrvr
00218
00218
00218
00218
00218
00218
00218
00218
00218
   ********************************************************************************************
00218                           *   Read - read LONG from board memory
00218                           *
```

```
Loc   F Object Code     Addr  M Source Statement

00218                            *   Entry:     A0 - param blk pointer
00218                            *              A1 - DCE pointer
00218                            *              A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00218                            *
00218
    ***************************************************************************************
00218                            Read
00218   48E7 C0C0                              MOVEM.L    A0/A1/D0/D1,-(SP)     ; save local work registers
0021C
0021C                            ; get base address of board
0021C   1029 0028                              MOVE.B     dCtlSlot(A1),D0       ; get the slot address
00220   E188                                   LSL.L      #8,D0             ; shift the 4 slot bits into proper position
00222   E188                                   LSL.L      #8,D0
00224   E188                                   LSL.L      #8,D0
00226   0080 F000 0000                         ORI.L      #$f0000000,D0         ; Slot space
0022C   2240                                   MOVEA.L    D0,A1                 ; A1 = board base address
0022E
0022E                            ; get address
0022E   200A                                   MOVE.L     A2,D0
00230   0680 0000 000C                         ADD.L      #csAddr,D0
00236   2040                                   MOVEA.L    D0,A0
00238   2010                                   MOVE.L     (A0),D0               ; D0 = address on board
0023A   0280 00FF FFFF                         ANDI.L     #$00ffffff,D0         ; mask to 24 bits
00240   2209                                   MOVE.L     A1,D1                 ; get board base address
00242   D081                                   ADD.L      D1,D0                 ; add it to the requested address
00244   2040                                   MOVEA.L    D0,A0                 ; A0 = requested address on board
00246   616A            002B2                  BSR.S      NbRead                ; get the byte
00248   2480                                   MOVE.L     D0,csVar(A2)          ; Return to caller the requested byte.
0024A
0024A   7000                     ReadGood      MOVEQ      #noErr,D0             ; return no error
0024C G 426A 000A                              MOVE.W     #ctlNoErr,csError(A2)
00250 G 426A 0008                              MOVE.W     #stGood,csStatus(A2)  ; Default status
00254   006A 0100 0008                         ORI.W      #stCmplt,csStatus(A2) ; flag call complete
0025A
0025A   4CDF 0303                ReadDone      MOVEM.L    (SP)+,A0/A1/D0/D1     ; restore local registers
0025E   4CDF 1110                              MOVEM.L    (SP)+,A0/A4/D4        ; restore registers
00262   6082            001E6                  BRA.S      ExitDrvr
00264
00264
00264
00264
00264
    ***************************************************************************************
00264                            *   Write - write LONG to board memory
00264                            *
00264                            *   Entry:     A0 - param blk pointer
00264                            *              A1 - DCE pointer
00264                            *              A2 - cs parameters (ie. A2 <- csParam(A0)) (must be preserved)
00264                            *
00264
    ***************************************************************************************
00264                            Write
00264   48E7 C0C0                              MOVEM.L    A0/A1/D0/D1,-(SP) ; save local work registers
00268
00268                            ; get base address of board
00268   1029 0028                              MOVE.B     dCtlSlot(A1),D0       ; get the slot address
0026C   E188                                   LSL.L      #8,D0             ; shift the 4 slot bits into proper position
0026E   E188                                   LSL.L      #8,D0
00270   E188                                   LSL.L      #8,D0
00272   0080 F000 0000                         ORI.L      #$f0000000,D0         ; Slot space
00278   2240                                   MOVEA.L    D0,A1                 ; A1 = board base address
0027A
0027A                            ; get address
0027A   200A                                   MOVE.L     A2,D0
0027C   0680 0000 000C                         ADD.L      #csAddr,D0
00282   2040                                   MOVEA.L    D0,A0
00284   2010                                   MOVE.L     (A0),D0               ; D0 = address on board
00286   0280 00FF FFFF                         ANDI.L     #$00ffffff,D0         ; mask to 24 bits
0028C   2209                                   MOVE.L     A1,D1                 ; get board base address
0028E   D081                                   ADD.L      D1,D0                 ; add it to the requested address
00290   2040                                   MOVEA.L    D0,A0                 ; A0 = requested address on board
00292   2012                                   MOVE.L     csVar(A2),D0          ; D0 contains the LONG to be written
00294   6136            002CC                  BSR.S      NbWrite               ; write the byte
00296
00296   7000                     WriteGood     MOVEQ      #noErr,D0             ; return no error
00298 G 426A 000A                              MOVE.W     #ctlNoErr,csError(A2)
0029C G 426A 0008                              MOVE.W     #stGood,csStatus(A2)  ; Default status
```

```
Loc   F Object Code     Addr  M Source Statement

002A0   006A 0100 0008                         ORI.W      #stCmplt,csStatus(A2) ; flag call complete
002A6
002A6   4CDF 0303               WriteDone      MOVEM.L    (SP)+,A0/A1/D0/D1     ; restore local registers
002AA   4CDF 1110                              MOVEM.L    (SP)+,A0/A4/D4        ; restore registers
002AE   6000 FF36       001E6                  BRA        ExitDrvr
002B2
002B2
002B2
002B2
002B2
002B2
002B2
   ******************************************************************************************
002B2                           *   NbRead - reads a LONG from a NUBUS card
002B2                           *
002B2                           *   Enter:     A0 - pointer to address in 32-bit address space
002B2                           *
002B2                           *   Uses:      no other registers
002B2                           *
002B2                           *   Exit:      D0 - the long word read
002B2                           *
002B2
   ******************************************************************************************
002B2                           NbRead
002B2   48E7 4000                              MOVEM.L    D1,-(SP)              ; save work register
002B6
002B6   2038 0001                              MOVE.L     true32b,D0            ; set 32-bit mode
002BA   A05D                                   _SwapMMUMode
002BC
002BC   2210                                   MOVE.L     (A0),D1               ; Get value specified
002BE
002BE   2038 0000                              MOVE.L     false32b,D0
002C2   A05D                                   _SwapMMUMode                     ; back to 24-bit mode
002C4
002C4   2001                                   MOVE.L     D1,D0                 ; return value
002C6   4CDF 0002                              MOVEM.L    (SP)+,D1              ; restore work register
002CA   4E75                                   RTS
002CC
002CC
002CC
002CC
   ******************************************************************************************
002CC                           *   NbWrite - writes a LONG to a NUBUS card
002CC                           *
002CC                           *   Enter:     A0 - pointer to address in 32-bit address space
002CC                           *              D0 - the long word to write
002CC                           *
002CC                           *   Uses:      no other registers
002CC                           *
002CC
   ******************************************************************************************
002CC                           NbWrite
002CC   48E7 4000                              MOVEM.L    D1,-(SP)              ; save work registers
002D0
002D0   2200                                   MOVE.L     D0,D1                 ; save value in D1
002D2
002D2   2038 0001                              MOVE.L     true32b,D0            ; set 32-bit mode
002D6   A05D                                   _SwapMMUMode
002D8
002D8   2081                                   MOVE.L     D1,(A0)               ; write value specified
002DA
002DA   2038 0000                              MOVE.L     false32b,D0
002DE   A05D                                   _SwapMMUMode                     ; back to 24-bit mode
002E0
002E0   2001                                   MOVE.L     D1,D0                 ; restore entry value
002E2   4CDF 0002                              MOVEM.L    (SP)+,D1              ; restore work register
002E6   4E75                                   RTS
002E8
002E8
002E8
002E8
002E8   0000 02E8               _End020Drvr    EQU           *                  ; the end of the driver
002E8                                          STRING        C
002E8
002E8
002E8
002E8
002E8
```

```
Loc   F Object Code     Addr  M  Source Statement

002E8
002E8   0000 7FEC                              ORG            ROMSize-fhBlockSize
07FEC
07FEC
    ********************************************************************************************
07FEC                            *             format/header block
07FEC
    ********************************************************************************************
07FEC   00FF 8014                              DC.L           (_sRsrcDir-*)**$00ffffff ; offset to sResource directory
07FF0   0000 8000                              DC.L           ROMSize           ; length of declaration data
07FF4   0000 0000                              DC.L           0                 ; CRC (Patched by crcPatch, an MPW tool
07FF8   01                                     DC.B           Rev1              ; revision level
07FF9   01                                     DC.B           AppleFormat       ; format
07FFA   5A93 2BC7                              DC.L           TestPattern       ; test pattern
07FFE   00                                     DC.B           0                 ; Reserved byte (must be zero)
07FFF   78                                     DC.B           $78               ; Byte lanes: 0111 1000 (bytelane 3)
08000
08000
08000                                          ENDMAIN

                                               END

Elapsed time: 3.06 seconds.

Assembly complete - no errors found.  8142 lines.
```

Appendix E - Schematic