PREFACE

INTENDED AUDIENCE

The DOS-73 Technical Reference is written for technicians doing component-level troubleshooting of the DOS-73 Coprocessor board.

ORGANIZATION OF THIS MANUAL

System Features and Functions

Briefly describes the physical features and functional capabilities of the DOS-73 Coprocessor board.

DOS-73: Theory Of Operation

Describes the DOS-73 hardware and the functions performed by it.

Diagnostics

Describes DOS-73 Coprocessor diagnostic procedures.

CONTENTS

1: SYSTEM FEATURES AND FUNCTIONS

GENERAL DESCRIPTION

The DOS-73 coprocessor is a peripheral product designed to bring MS-DOS capabilities to the AT&T UNIX PC. The DOS-73 hardware is essentially a PC "on a board," with several enhancements. The hardware enhancements include: Hercules monochrome graphics emulation (first page only), PC COM2: port emulation, Microsoft Mouse emulation using the mouse, and increased performance due to a full 16 bit bus and increased clock speed. This document describes the DOS-73 hardware and outlines the DOS-73/7300/UNIX interface standards as ALLOY has defined them.

DOS-73 Architecture

The DOS-73 hardware was designed to parallel the PC architecture. The hardware also includes enhancements which allow it to be integrated into a multi-processor environment, and improvements. The system includes the following:

- 8086 CPU
- 512K Bytes Dynamic RAM
- Memory Timing & Control Circuitry
- INS8250 Asynchronous Communications Element
- Interface Circuitry to the 7300 bus
- I8259 Programmable Interrupt Controller
- I8253 Programmable Timer
- Video Data Handling Circuitry
- I/O Intercept Circuitry
- 8087 Numeric Coprocessor Hooks

A block diagram of the board is found on the following page.

DOS-73 BLOCK DIAGRAM

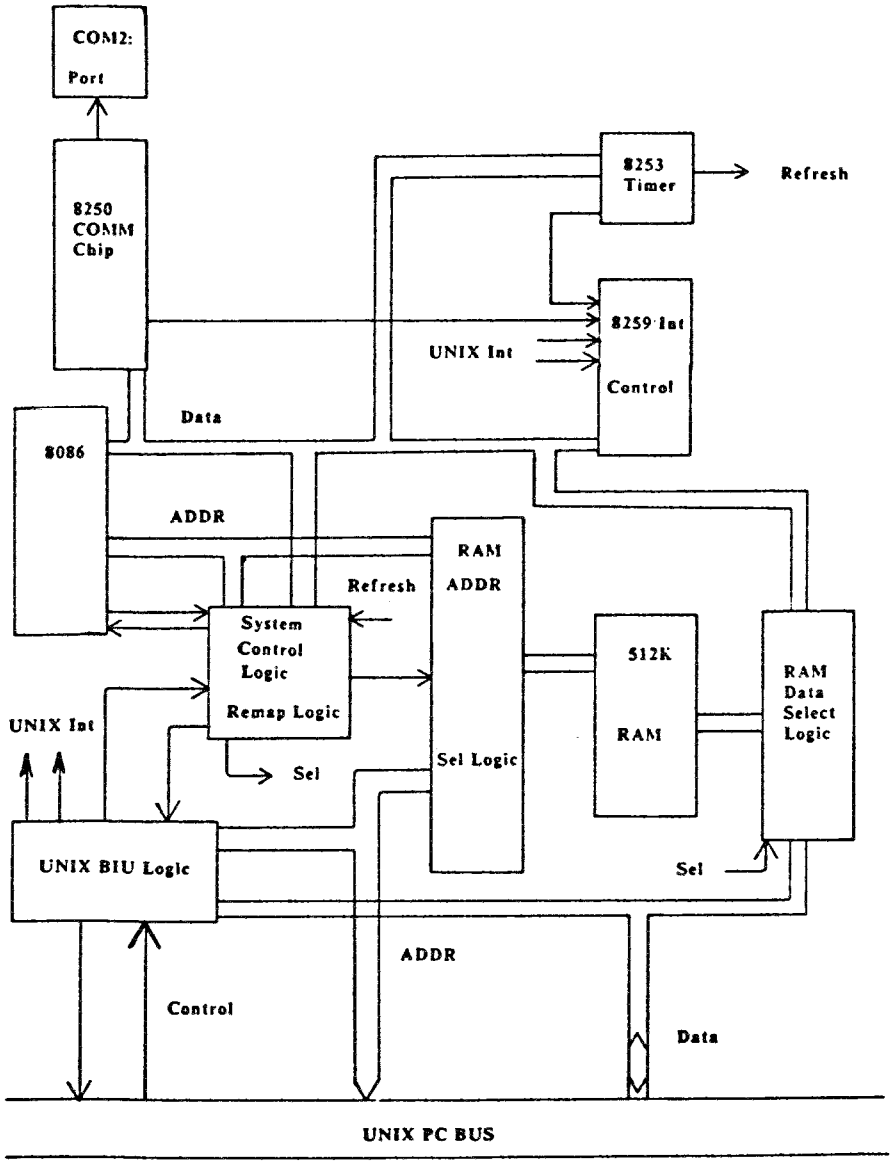

Figure 1-1. DOS-73 Coprocessor Block Diagram

2: DOS-73: Theory Of Operation

CPU

The DOS-73 utilizes an INTEL 8086 CPU. The CPU is run at a master clock frequency of 8 Mhz. This effectively doubles the speed of the DOS-73 board when compared to the IBM PC. The 8086's 16 bit data bus further increases the overall system throughput.

Minor problems occur when switching from an 8088 to an 8086 and continuing to use 8 bit VLSI peripherals. One problem is that the 8086 expects data for 8 bit I/O cycles, to odd addresses, on the upper half of the data bus. This is a problem because the 8 bit peripherals usually reside on the lower half of the bus. Thus, a data path is needed to move data from the lower half of the bus to the upper half on I/O operations to odd addresses. This process is known as byte swapping.

The logic to control this is incorporated in M33 (PA20L10) and in the bus transceivers M24, M25, and M49. M33 monitors the bus control signals generated by the 8086 during its execution of instructions. Based upon the type of operation being executed, M33 sends the appropriate device enable control signals to M24, M25, M49, according to the following table:

OPERATION	CHIP ENABLED
word wrt to mem	M24,M49
word rd from mem	M24,M49
byte wrt to mem (even addr)	M24
byte wrt to mem (odd addr)	M49
byte rd from mem (even addr)	M24
byte rd from mem (odd addr)	M49
byte wrt to I/O (even addr)	M24
byte wrt to I/O (odd addr)	M25
byte rd from I/O (even addr)	M24
Byte rd from I/O (odd addr)	M25

The CPU is supported by two other chips, the I8284 clock generator, and the I8288 bus controller. These chips are M51 and M50, respectively. The 8284 is used to divide the master system clock into a 33% duty cycle, 8Mhz clock. This chip also produces a 4mhz peripheral clock, and serves to synchronize the 8086's reset. The 8288 is used to decode the 8086's status signals and generate the bus control signals traveling to the rest of the board. This chip is also responsible for generating the signal ALE. This signal strobes the 8086 addresses into the address latches. The address latches are M30, M37, and M43. The addresses must be saved because the 8086 time multiplexes its addresses and data signals. Thus, the addresses that the 8086 sets up are only valid for a short time at the beginning of a bus cycle, and you must save them if you want to use them through the whole cycle.

Memory Organization

The DOS-73 utilizes 512k bytes of random access memory. The memory is implemented using sixteen, standard technology, 256k x 1 bit dynamic RAM chips. They are M56 through M71. The RAM system is dual-ported. This means that the 8086 may access the RAM or the UNIX system may access the RAM. In this section we will only discuss how the 8086 gets access to the RAM. The UNIX interface to the RAM will be discussed later.

The basic RAM timing is derived from a simple delay line circuit. Any memory read or memory write by the 8086 will start a memory cycle. The 8086 MRD and MWT signals are or'ed by the three input or gate M1. The output from this gate clocks a flip-flop (M12A), which generates the RAS to the RAM banks, and the master cycle length input to the delay (M2) line. The delay line delays the input signal, by some pre-specified length of time, before passing it to the output pins. The delay line we use has 40ns increments. Thus, the delay from input to the output on T1 is 40ns, to T2 is 80ns, and so on. T1 from the delay line is used as the MUX signal to the address multiplexors. Thus, 40ns after RAS becomes active the select on the address multiplexors changes to select the CAS addresses. Tap T2 is used to feed the clock on the CAS flipflop. So, 80ns after RAS becomes active the CAS flipflop (M12B) gets clocked and the master CAS signal becomes asserted. The actual CAS signals to the RAMS are decoded in M3.

M3 takes the output from the CAS flipflop and gates it with four other signals, to form HICAS and LOCAS. HICAS and LOCAS allow byte or word operations to the memory. The bank decode signals are XHI, XLO, HIDEN, and LODEN. XHI and XLO are generated by M33 and apply only to memory transfers by the UNIX system. XLI and XLO will not become active unless UNIX has control of the DOS-73 buses. HIDEN and LODEN are generated in M38 and M33, respectively. HIDEN becomes active when the UNIX system does NOT have control of the bus and the 8086 wants to transfer data on the upper half of the data bus (D8-D15). LODEN becomes active when the UNIX system does NOT have control of the bus and the 8086 wants to transfer data on the lower half of the bus (D0-D7). HICAS and LOCAS are generated according to the following table:

OPERATION ----->	HICAS	LOCAS
I/O (any width)	false	false
MEM (word even aligned)	true	true
MEM (byte even addr)	false	true
MEM (byte odd addr)	true	false

The last bit of memory to talk about is the refresh. The DRAMS we use require a 256 cycle 4ms refresh. Thus, if we space refresh cycles evenly, we must do a refresh every 12.5us to be safe. The way the system is structured, there are two types of refresh. One is a refresh when the UNIX system has control of the DOS-73 bus, and the other is a refresh when the 8086 is running.

When the 8086 is running, the refresh timing is generated on the board. To generate these refresh cycles we use one channel of the onboard programmable timer to generate a pulse train with a rising edge every 12.5us. This wave form is fed into the clock of the refresh request flip flop (M9b). Under normal conditions, this will cause the flipflop to be clocked and a refresh request to be generated. There are two conditions when this will not happen. One, if the 7300 has control of the local bus, then the onboard refresh requests will be inhibited. Two, if the 7300 has requested access to the local memory, then refresh requests will be cleared so as not to confuse the bus arbitration state machine.

COM2: Emulation

The DOS-73 board provides for I/O communication with the outside world by having hardware onboard that emulates the PC COM2: port. This is accomplished by using the same type of VLSI communications chip that IBM uses. The chip is the NATIONAL 8250 (m18). We have preserved compatability with IBM by installing the chip at the same I/O address as the PC's COM2: port. The actual address decoding is done by the MASTER_IO_DEC pal, a 1618 (m14). The COM port is further supported by RS-232 line drivers and receivers (m4 and m5) to translate the TTL signal levels to RS-232 levels. Physical interface compatability is achieved by terminating the signals with a DB-25 connector that is the same sex as the PC's COM port, and has the same pinouts.

Timer Emulation

The DOS-73 board has an onboard 8253 VLSI counter/timer chip. This is used for two things. One, to provide a clock to the RAM refresh circuit, and two, to provide a pulse train to the interrupt controller which emulates the IBM PC timer tick interrupt. The 8253 happens to be the same part that IBM uses to generate its internal timing, thus more PC compatability is achieved here. The I/O decoding for this chip is done by M14, the master I/O decoder PAL. Finally, the 8253 reqires a clock input from which it derives all of its timing. We provide it with a 1Mhz clock input. This is obtained by dividing the processor 8Mhz clock by two, three times. The clock division is done by flip-flops in M45 and M7.

Interrupt System

The DOS-73 coprocessor has an interrupt system very similar to that on the IBM PC. It uses an 8259 programmable interrupt controller (m32). The channel assignments are as follows:

CHN #	FUNCTION
0	Reserved for future use
1	Reserved for future use
2	Timer Tick interrupt from 8253
3	Comm interrupt from 8250
4	Interrupt 0 from UNIX interface
5	Interrupt 1 from UNIX interface
6	Interrupt 2 from UNIX interface
7	Interrupt 3 from UNIX interface

The 8086 is structured such that during an interrupt acknowledge cycle it wants to see valid vector data on the lower half of its data bus. The problem that arises here is that during an intak cycle the 8086 does not set up its address lines, thus the addresses on the system bus during an intak cycle are the ones left from the last instruction that was executed. This causes a problem if the last bus operation was a memory or I/O operation to an odd location. If this scenario occurs, A0 is left in the wrong state for proper data transfer during the interrupt acknowledge cycle. To avoid this type of problem we fix A0 to a zero during intak cycles. The address generation is done by negative logic or'ing (M27-c) A0 with INTAK, to form WILDA0. WILDA0 is then used by the system as its A0.

I/O Intercept System

The DOS-73 supports many onboard I/O mapped devices that the IBM PC does, however, we can't support them all due to space and cost limitations. Some important ones that aren't supported in hardware are the COM1: port, LPT: port, and the floppy controller. Since many PC compatible programs talk directly to these devices, we need to know when code is attempting to access them, so we may then emulate these devices and others in software. Hence the creation of the I/O intercept system.

The I/O intercept system is a simple concept. In hardware we keep a map of the addresses of I/O mapped devices that we do and don't support. As each instruction is executed hardware decodes what instruction is being executed for use later. More hardware compares the address of the instruction being executed to the address map of devices. If the instruction being executed is an I/O instruction and the address accessed by that instruction is in the map as valid, then processing continues normally. However, if the address was shown as invalid in the map then the following happens. First, we generate an NMI to the onboard CPU. This will be recognized prior to executing the next instruction. At the same time the lower fourteen bits of the address bus and two bits indicating the type of instruction being executed at the time of the NMI are stored into two eight bit latches. At this point the NMI service routine has enough information to tell what type of operation happened to cause the NMI and where the instruction was trying to access.

The address map is maintained in M14 (PAL1618). This PAL does all the I/O address decoding for the 8086, thus by default contains the map. The output labeled ILLADD(L) indicates that an address not on the board is being accessed if it is true. This signal is fed into M20 (PAL2018), and is used to generate the NMI to the CPU. In this PAL is logic which may enable and disable this feature. This logic is seen as the term ENIO. ENIO is essentially a one bit latch. When an I/O write occurs to port A2h then the latch is set true; when a write to A3h is done then the latch is cleared false. To generate the NMI to the cpu the ENIO bit is anded with ILLADD and IOW or IOR (see terms 2 and 3 of NMI equation in M20). Thus, if ENIO and ILLADD and IOW or ENIO and ILLADD and IOR are true, then the PAL will generate an NMI. The NMI signal is used to freeze the state machine tracking the instruction being executed (M38), and store its information as well as the bus address information into the NMI latches (M13 and M17).

Video Intercept System

The video intercept system on the DOS-73 is quite similar to the I/O intercept system. The video intercept system gives the software a way of knowing when the screen RAM has been accessed, and also revectors memory operations from the screen RAM addresses to the top part of our memory. The DOS-73 has only 512k of RAM, thus the IBM video RAM page falls outside of our memory. Therefore, we need a way to make video memory operations access our memory.

In a manner similar to the I/O system, the video NMIs may be enabled and disabled. This is done by the ENVID term in M20. An I/O write to A0h will set this true and enable NMI's to be generated when memory writes are done to the video RAM page, and a write to A1h will disable this feature. The term in the NMI equation which does this is term 1. To generate a video NMI it requires that ENVID and MEMW be true and that the bus address is equal to the B0000h memory page.

The remapping is similar to the NMI generation in that it may be enabled and disabled. This is done by an I/O write to the STH register and by setting the ENBMAP bit. This bit is set false after a board reset and is generated in M40 (PAL16R4). The ENBMAP signal is then fed in to M20, where the address comparison for remap addresses is done. The equation MAPNOW is what causes a video remap. This equation will be true any time that the bus address is equal to the B000h page and ENBMAP is true. This signal is fed into M21 (PAL16L8) which will do the actual address translation for the remap. M21 is used to generate the upper five address bits that will go to the system RAM (SYA14 - SYA18). In addition to the remap, it multiplexes the addresses from the 8086 and the UNIX system that go to the RAM. A video remap occurs as follows: if the 8086 is writing to the RAM and the MAPNOW signal goes true, the PAL fixes SYA15 thru SYA18 to a high state. These addresses are latched in on the falling edge of CAS, thus the remap cycles are forced into the top 32k of our RAM.

Math Coprocessor Emulation

The DOS board has the circuitry onboard to support the 8087 numeric coprocessor in an PC compatible fashion. The only support issue to contend with here is what happens when there is an operation fault in the 8087. When this occurs, an NMI is generated to the 8086. Following this the application program running at the time will generally do an I/O read from port 62h to obtain information concerning the cause of the NMI. We emulate two bits in the port 62 register, D6 and D7. We fix d6 low, thus indicating that an I/O CHCHK is false, and we set D7 to reflect the state of the 8087 int signal. This has the effect of indicating no parity error if the 8087 causes the NMI, and a parity error if anything else sets the NMI. When application software checks this register and sees an NMI with no parity error it will go to its 8087 error handling routines. If, however, it sees a parity error, it will pass control to our routines, which will then deal with the NMI according to our needs (I/O NMI or VIDEO NMI).

UNIX Interface

The UNIX PC provides an independent memory and I/O space for each slot in its bus. The I/O space is 256k bytes in length. The DOS-73 resides only in the UNIX PC slot I/O space. In our mapping scheme, the upper 128k of the I/O space is used for the board ID registers and the DOS control and status registers. The lower 128k bytes is mapped into our RAM.

Each expansion board for the UNIX PC must have its own unique four byte sequence of identification bytes that the computer may read. To save space, we have hard coded the board ID into a PAL (M54). This PAL is accessed and its data is gated onto the UNIX PC's bus any time that a read operation is done to the ID addresses for the slot that the board is in. The slot address comparison is done by M22 (74ls85). This chip compares the upper three UNIX address bits to the hardwired slot id bits of the slot the board is in. If they are equal and the PC is doing an I/O cycle, as indicated by the signal XIOEN, then the comparator will set its output true. This is fed into a flip-flop (M9), where it is synchronized to the UNIX PC's peripheral clock, to form the signal BOARDIO(H). Whenever this signal is true it indicates that the UNIX PC is doing a valid I/O access to our I/O space. The board id PAL uses BOARDIO(H) and the signal IOSPACE(H) and the lower four address bits to decode whether the board id is being accessed or something else is being accessed.

The other two registers in the upper half of the slot's I/O space are the UNIX_TO_DOS control register and the DOS_TO_UNIX status register. The address decoding for these two is done in M33 (PAL20L10). The DTU register may be read by the UNIX system and written by the 8086. It is primarily used to pass interrupt information to the UNIX system from the DOS-73 environment. It is also used to generate the physical interrupt to the UNIX PC. The 8086 interrupts the UNIX system by writing to the STH with D7 high. This causes two things in M41 (PAL20X8): First, the intbit (xd15 to unix) is set, indicating that an interrupt is set, and second, the HIRQ(L) line is set true to interrupt the UNIX system.

The UNIX system sends high level commands to the DOS-73 board through the UTD register. The UNIX system writes to this port and the address decoding is done by M33. Through the use of this register the UNIX system may Reset the DOS processor, request its memory, interrupt it, or clear an interrupt generated by the 8086 to UNIX. The bit definitions are detailed in the Software Driver Specifications.

The last bit of the UNIX interface to discuss is the memory window interface. The UNIX system may communicate with the DOS-73 board by windowing any one of the four 128k byte local memory segments into the lower 128k bytes of the slot id space. Once this is done, the UNIX system may read and write to this memory as though it were its own.

The windowing works as follows. When the DOS-73 board is in the reset state then the memory belongs to the UNIX system. When reset is de-asserted, then the local 8086 starts up and the RAM belongs to it. At this point, if the UNIX system wishes access to the RAM, it must follow a request/acknowledge arbitration process. First, the UNIX system must write to the UTD register and set bits 4 and 5. These bits will be decoded later to address one of the four 128k segments in local RAM. Next, the UNIX system must set the MEMORY_REQUEST bit. This line designates that the UNIX system is requesting access to the local memory. This bit becomes the TAKEMEM(L) signal and is fed into M26 (PAL16R8). This PAL is actually a state machine which translates the TAKEMEM(L) signal to a REQUEST/GRANT sequence that the 8086 will accept (see intel Microprocessor Components Handbook). When the PAL receives the grant pulse from the 8086, then it sends a BUSAK(L) signal to the rest of the board. This signal degates the 8086 address and data drivers from the local bus and asserts the drivers from the UNIX PC bus. This signal is also sent to the DTU PAL where the UNIX system may monitor its status. When the UNIX system sees this signal become true then it may read and write to our RAM. During the time that the UNIX system has access to our memory we feed the XRFBG signal into M26 to generate refresh cycles to our RAM. When the UNIX system has finished with our memory, it resets the TAKEMEM(L) bit and the M26 state machine executes the 8086 bus release sequence.

Software Driver Specifications

This section describes the relationship between the UNIX operating system and the DOS-73 device. This section includes two parts:

1) A description of the DOS-73 hardware.
2) A description of the user-level interface supplied by the DOS-73 device driver.

DOS-73 Hardware

Before describing any part of the UNIX/DOS-73 relationship, some basic information must be known about the DOS-73 hardware.

Just a glance at the DOS-73 board shows the basic configuration. The device is equipped with an 8086 microprocessor with 512K bytes of random access memory. There is support for the device to receive and transmit interrupts from/to the UNIX PC. The hardware is configured to interrupt the Unix PC at level one. The DOS-73 device also has a local RS-232 port and 8087 math processor, but neither are directly accessible from the user level.

Since there is only 256K bytes of address space per slot on the UNIX PC bus, the 512K bytes RAM is divided up into four pages of 128K bytes each. The selection of different pages on the DOS-73 device is done through a control register, which will be explained shortly.

The DOS-73 board is identified by the contents of the last four odd addresses in the slot it occupies. These locations contain the identification bytes and their associated checksums. The following table shows those values in hexadecimal:

Description	Offset in Slot	Value
LSB of ID	0x3fff9	0x86
MSB of ID	0x3fffa	0x73
LSB of IDCK	0x3fffc	0x7a
MSB of IDCK	0x3fffe	0x8d

The DOS-73 board communicates with the UNIX PC through one register residing at offset 0x3ffee. This register is called the Host-To-DOS/Status register, but for simplicity it will be referred to as the HTD register. When this register is written to by the s4, commands are sent to the hardware. When this register is read by the UNIX PC, commands are received from the hardware. Since there are no means of reading the register to get the last value written to it, the DOS-73 device driver maintains an image of the HTD register. Details will be explained later.

The HTD register control bits are as follows:

(Bit 0 always refers to the least significant bit.)

Bit 0 - RESET

This bit controls the main state of the DOS-73 device. When this bit is 1, the DOS-73 device is active and the 8086 is running. When this bit is 0, the DOS-73 device is reset and nonactive. The 8086 microprocessor begins executing code at address 0X1FFF0 when enabled.

Bit 1 - BRQ

This bit sends a bus request to the DOS-73 device. When a bus request is granted, the DOS-73 device's memory is in a tri-stated mode, and can be written to by the UNIX PC. When this bit is 0, such a request is made. When this bit is 1, no request is made.

Bit 2 - Clear Interrupt.

This bit is toggled to clear an interrupt that was received from the DOS-73 device. Toggling in this sense means set to 1, then back to 0.

Bit 3 - Interrupt/Bank Select

This bit has two functions which go hand-in-hand with bits 4-7 inclusive. When this bit is 0, the DOS-73 board is interrupted with the value (bit) selected in bits 4 through bit 7, (only one of these bits can be 1) producing four distinct interrupts. When this bit is 1, no interrupt is produced but it indicates that bits 4 and 5 are used to select which page bank of memory is to be addressed.

Bit 4 - I1/LSB of page bank

Bit 5 - I2/MSB of page bank

Bit 6 - I3

Bit 7 - I4

When the HTD register is read, it is a status register, with the following bit assignments:

Bit 0 - Bit 4 Inclusive - Request "type"

These four bits are set by the program running on the DOS-73 device to signal a request type to the UNIX PC. On an interrupt, this value is used to interpret what kind of service is requested.

Bit 5 - Bus ACK/Grant

This bit is 0 when bus request transmitted by the UNIX PC has been granted, 1 otherwise.

Bit 6 - RFU

Ignore this.

Bit 7 - Request

If this bit is 1, the DOS-73 device has transmitted an interrupt.

These registers are manipulated by the driver to control the device. There is some user-control of the HTD register, and any program running on the DOS-73 device should understand how these registers work, specifically the request numbers, since these affect user processes dealing with the device.

User Level Interface

The DOS-73 device is represented to the user as the character special file "/dev/dc73". This file is set up with mode 666, but can be changed by the system administrator.

There are only three system calls that are supplied to the device: open, close and ioctl. All I/O to the device is done through ioctl calls. Any attempts to read/write to the device in any other manner will produce an error.

The concept of an "owner" process is associated with the DOS-73 device. Only one process can have the device open at one time, and that process is the owner. Also, signals are sent to the process to indicate certain states of the DOS-73 board, and the opening process should set up the environment to act accordingly. Details will be explained later.

All control of the DOS-73 device is done through ioctl calls. The user level ioctl structure is contained in the include file "dc73.h". This file should be included in every C program that deals with the DOS-73 device. Upon installation of the DOS-73 System Software, this file is placed in /usr/bin/DOS. It should be moved into /usr/include/sys.

A portion of that file is displayed here for the purposes of this discussion:

```
/*
 * dc73ioctl structure.
 *
 * The following ioctl commands exist:
 *
 * SETBASE - Sets the default mask for the HTD register.
 *
 * INTDOS - Sends an interrupt to the DOS board.
 *
 * DOSREAD - Read data from DOS board.
 *
 * DOSWRITE - Write data to the DOS board.
 *
 * RMREAD - Read data from the board with remapping and swapping. This call will *
                    read 32K bytes. The user must have this space allocated.
 *
 *
 */

#define SETBASE 0
#define INTDOS 1
#define DOSREAD 2
#define DOSWRITE 3
#define RMREAD 4

union dc73io {
        struct setbase {
                unsigned char base;  /* New default base          */
        }b;
        struct dosint {
                unsigned char dint;  /* Interrupt level           */
        }d;
        struct dosrw {
                char *dosaddr;                  /* Dos address                  */
                char *hostaddr;                 /* Host address                 */
                int count;                              /* Xfer bytes                   */
        }rw;
        struct rmread {
                char *dosaddr;                  /* Dos address                  */
                char *hostaddr;                 /* Host address                 */
        }rm;
};
```

As you can see, there are five commands supplied by the DOS-73 device driver. Each command uses a different structure in the union. All ioctl calls return -1 on error and errno is set to reflect the error. The following is an explanation of the usage and actions of each of the commands:

1) SETBASE:

The DOS-73 device driver uses the HTD register for many things, such as giving bus requests, interrupts, and the like. However, since there is no way of reading the HTD register to get the last command value written, the driver maintains a "base" value of the register. This value can be set by a user process using the SETBASE ioctl call. This call uses the union b in the ioctl structure. All writes to the register use this base value, then is modified by the driver. Using this scheme, the user can put any value in the HTD register, but use of this command is recommended only for setting initialization values and/or resetting the processor. The driver does all the necessary commands for doing writes and interrupts automatically through other ioctl calls.

2) INTDOS:

Using this ioctl call, the device driver will send an interrupt to the DOS-73 device. This ioctl call uses the structure d in the union. The value in the dint field must be 0, 1, 2, or 3, giving the user access to all interrupts.

3) DOSREAD and 4) DOSWRITE:

These two ioctl calls support all transfers of data from user memory onto the DOS-73 device. The user supplies in the union rw:

i) A pointer to the start of the space to take/put information to/from the DOS-73 device. This is placed in the field hostaddr. All register manipulations are automatic.

ii) An address to place/take data to/from the DOS-73 device. This value is placed in the field dosaddr. This field should really be defined as an int, but . . .

All page mapping is done automatically by the driver. DOS-73 addresses range from 0-512K. Bus requests are also handled by the driver.

iii) A count of the number of bytes to transfer. This is placed in the field count.

The user must have count bytes allocated in the space. The driver allocates no memory. If EIO is ever set as an errno, the DOS-73 device denied a bus request. This is usually a signal that the hardware is going south.

5) RMREAD:

This is a special read that is used for graphics transfers. Some additional conversions must be done when sending the graphics RAM over to the s4. This is done automatically in the driver. All graphics transfers are 32K long. The user supplies the dosaddr and hostaddr as before in the structure rm.

Also associated with the device are signals that are sent to the owner process according to certain values held in the status register after an interrupt. When an interrupt occurs, the driver looks at the value in the status register and, if it recognizes it, sends a signal to the owner process. If the value is unrecognizable, the driver prints an error to the console.

The following bit patterns generate the following signals:

Bit Pattern	Signal
0x01	SIGUSR2
0x02	SIGUSR1
0x0f	SIGQUIT
0x04	SIGINT
0x08	SIGTERM

The user should set up the process to catch these signals and to dictate appropriate actions. The 8086 program running on the DOS-73 device should ONLY use those bit patterns, since any else produce errors.

It is clear that the 8086 program and the UNIX user process should have strict interfaces to avoid confusion.

3: Diagnostics

To run the Diagnostics, insert the DOS-73 Diagnostics Diskette into the floppy disk drive and invoke the UNIX Shutdown command. When prompted, press the <Return> key to load the diagnostics.

Testing The COM2: Port

A COM2: LOOPBACK connector has been included with your DOS-73 package. It connects the following RS-232 pins together:

Pin #	Function	Pin #	Function
2	Transmit	3	Receive
4	RTS	5	CTS
20	DTR	6, 8	DSR-RLSD

This connector must be inserted in the DOS-73 COM2: port before testing begins. If this connector is not inserted in the COM2: port a Failed message will appear after 'Testing COM2.'

Number of Test Passes

The user selectable option for the DOS-73 diagnostics is the number of test passes. Each pass tests all of the components of the DOS-73 Hardware. The Diagnostics can be aborted at any point by hitting the <Break> key.

The following screen is displayed when the DOS-73 Diagnostics are loaded:

```
DOS-73 Diagnostics                    Rev. x.xx

Enter the number of passes (0= Test until Break is hit)  <0> 1

Insert the COM2: LOOPBACK connector to the rear of the
DOS-73 board.

     Hit return key to continue...

Testing MEMORY                  (Passed)
Testing Video NMI               (Passed)
Testing I/O NMI                 (Passed)
Testing Video Remap             (Passed)
Testing COM2:                   (Passed)
Testing Re-fresh. Timer         (Passed)
Testing Interrupt Controller    (Passed)
Testing address trap            (Passed)
Testing 8087                    (Absent)

                Pass =1  (Successful)

     Press return key to continue...
```

Test Descriptions

1) ALL THE TESTS

Self-explanatory: prompts for the number of passes desired and executes all tests round robin.

2) TEST MEMORY

First writes hex data 00, 0FF, 0AA, and 055 to entire 512K of DOS-73 memory and reads it back; repeats above with hex data 055

2a) TEST MEMORY: ADDRESS LINES

Tests each of the memory address lines by writing a unique value out on each line, then reading it back.

3) TEST NMI SPEED

Sets up interrupt vector for NMI service routine (which clears register ax to 0).
Enables video NMI.
Writes data to video RAM (455th video RAM address) which should cause an NMI.
Checks if ax was cleared to 0.

4) TEST VIDEO NMI

Sets up interrupt vector for NMI service routine that reads lsb of NMI latch (port 24h) into register al and masks lower 2 bits; and enables port and video NMI.

bit 0 of latch = 0 for VIDEO RAM WRITE
1 for I/O read or write
bit 1 of latch = 0 conveys no information for VIDEO NMI
1 indicates I/O write if I/O NMI
checks if al is 0

5) TEST PORT OUT NMI

Same as above, except checks if al is 1.

6) TEST PORT IN NMI

Same as above, except checks if al is 3.

7) TEST VIDEO REMAP

Disables video NMI, writes hex data 0a55a to hex addresses 0, 401, & 802 of PC screen RAM (page 0b000); reads data back data at same addresses of DOS-73 screenram (page 7800)

8) TEST COM2: DATA BIT

Sets up interrupt vector for NMI service routine which reads both lsb and msb of NMI latches (latch information is not used for anything), initializes 8250 chip (9600 Baud, 8 Data Bits, 1 Stop Bit, no parity), loads data into register ah (data is 0) repeats 9X: rotates-data left 1x, transmits data to 8250, receives data from 8250, and checks to see if data matches.

9) TEST COM2: DTR LINE

Initializes 8250 chip as above, clears modem status register, turns DTR on, reads modem status register (bit 5 should be clear), turns DTR off, and reads modem status register (bit 5 should be set).

10) TEST COM2: RTS LINE

Initializes 8250 chip as above, turns RTS on, reads modem status register (bit 4 should be clear), turns RTS off, reads modem status register (bit 4 should be set).

11) TEST RE-FRESH/TIMER

Reads counter 0 (timer tick) of CTC (8253) -lsb the msb, kills some time, reads counter 0 again -lsb then msb; checks if tick count elapsed is within acceptable range.

12) TEST 8087

Part 1: hand coded portion checks if 8087 is present; it divides 5 by 2 (integer division) and expects the result to be 2. If part 1 succeeds then, part 2: does floating point arithmetic. It adds 3.75 to 3.75 and expects the result to be 7.5. Then it multiplies 7.5 by 7.5 and expects the result to be 56.25; then it divides 56.25 by 7.5 and expects the result to be 7.5; then it subtracts 3.75 from 7.5 and expects the result to be 3.75.

13) TEST INTERRUPT CONTROLLER

Tests interrupts 4 thru 7. Host sends interrupt request (and level number) to DOS-73 board; board sends level number back to Host (via Slave Status Register) in interrupt service routine.

14) TEST ADDRESS TRAP

Selects incorrect card select addresses (i.e., slots excluding the slot that the DOS-73 board presently resides in) on the PC 7300 and tests if the DOS-73 board responds.

In Case Of Trouble

If any errors are detected, the DOS-73 Diagnostics will report a 'Failed' message after the appropriate test and an 'Unsuccessful' message after the PASS # count.

At this point you should do the following:

1. Power off the UNIX PC, using the "Shutdown" command.
2. Remove the DOS-73 Board.
3. Press down on all of the socketed I.C. chips to assure that they are properly seated.
4. Replace the DOS-73 board.
5. Power up the UNIX PC.
6. Insert the DOS-73 Diagnostics diskette and repeat the test.

If problems persist, select another slot in the UNIX PC and repeat the test. If you are still unable to successfully test the DOS-73 system, it must be returned for repair.