SCSI Your Mac

by John Bass

I got tired of waiting on those sluggish Mac floppy drives, tired of waiting for an affordable hard disk for the Mac. I got so tired, finally, that I went to work. My efforts paid off, and I was pleased enough with the result to think that others might appreciate my MacSCSI® interface, too.

One of the most obvious advantages to doing your own hard disk interface for the Mac is cost. I think MacSCSI is the hard-disk equivalent of *DDJ*'s January 1985 "Fatten Your Mac" 512K upgrade: the hacker's hard disk upgrade at budget prices, if you're willing to shop around for disks and SASI/SCSI controllers. Here in Silicon Valley we see hard disks at the computer swaps for between $100.00 and $500.00, often either refurbished or new discontinued models that were surplus. Likewise, SASI/SCSI controllers are seen at between $50 and $150. Thus, with the MacSCSI interface and some bargain hunting you can put a hard disk on your Mac for less than $500. New production drives and controllers are available between $900 and $3000, depending on the size and performance.

Another advantage is the experience you'll gain, both in writing your own bells and whistles into the supporting software, and in simply cracking open your Mac and getting to know its innards better. And in this regard, the MacSCSI interface is unlike the 512K upgrade: you don't have to take a soldering iron to a delicate, expensive logic board to install MacSCSI. You don't have to modify the Mac logic board at all.

The parts for this venture are few. The MacSCSI host adapter consists of four parts: an NCR5380 single chip SCSI interface, two 74LS10's to provide the address decodes, and a 50 pin header for the SASI/SCSI cable. The rest of a complete setup is a SASI/SCSI controller, hard disk drive, power supply, and cables for power and data. You can make your own case from a bookcase speaker for a few dollars, you can build a nice custom cabinet, or you can buy an off-the-shelf hard disk enclosure complete with power supply and cables.

Figure 1 (page 95) tells the story: it contains the schematic for the MacSCSI interface. The host adapter is memory mapped into the Mac's ROM address space above its last valid address. To accomplish this, I used selection equations for the NCR5380 of

$$NCRSEL = \overline{ROMSEL} * A20 * \overline{A11}$$
$$NCRRD = NCRSEL * A4$$
$$NCRWRITE = \overline{UDS}$$

By using address lines to enable the read and write enables on the NCR5380, I avoided needing to run any wires to chips or to cut away at the main circuit board; the only interface point to the Mac is its ROM socket. This scheme does, though, require that the software read an NCR5380 register at one address and write it at another—quite a reasonable tradeoff for not having to do any cuts or adds to the main logic board, I think.

While it's not clear whether Apple would honor a warranty or Apple Care contract on a Mac upgraded this way, if done carefully the upgrade will not physically alter or harm the Mac. If you have problems with your Mac (other than the MacSCSI upgrade), most helpful dealers should be willing to service your Mac if your boards are not altered or damaged. You may even find some helpful dealers and independent repair centers that will install the upgrade for you at their normal shop rates (about an hour of their time).

If you're up for a project, the interface can be assembled using off-the-shelf parts and point-to-point wiring. If you do it this way, you'll need to make a chip carrier from perf board with holes on 0.1 inch centers. You can salvage the socket pins for the chip carrier by cutting apart two Augat low profile 28 pin sockets with machined pins. Then attach the perf board to a wooden support block and drill out to 0.055-inch in the pin pattern for the Mac's ROMs. Keeping the wooden block attached, press the pins into the perf board using a vise or press (be careful not to crush the pins). The wooden block provides a die to protect the socket pins during the insertion process. You can then insert the other parts into the perf board and wire them using point-to-point soldering on the back side. Finally, and carefully, remove the ROMs from the board, insert them into the chip carrier, and plug the chip carrier with the ROMs back into the ROM sockets.

I'm skipping any discussion on how to take your Mac apart and reassemble it; for that, see the installation section on page 96.

Now for the hard part. I've come up with several options for getting the SCSI ribbon cable out of the case, none of them elegant. One is to cut a hole in the back of the case for an exit. Another is a panel-mounted connector installed in the case (my preferred solution). Or you can run a ribbon cable out between the bottom case seam in front, although you'll have to widen the seam slightly to do this. A fourth solution is not to bring the cable out at all, of course. General Computer has already demonstrated the feasibility of installing a hard disk inside the Mac case. I haven't

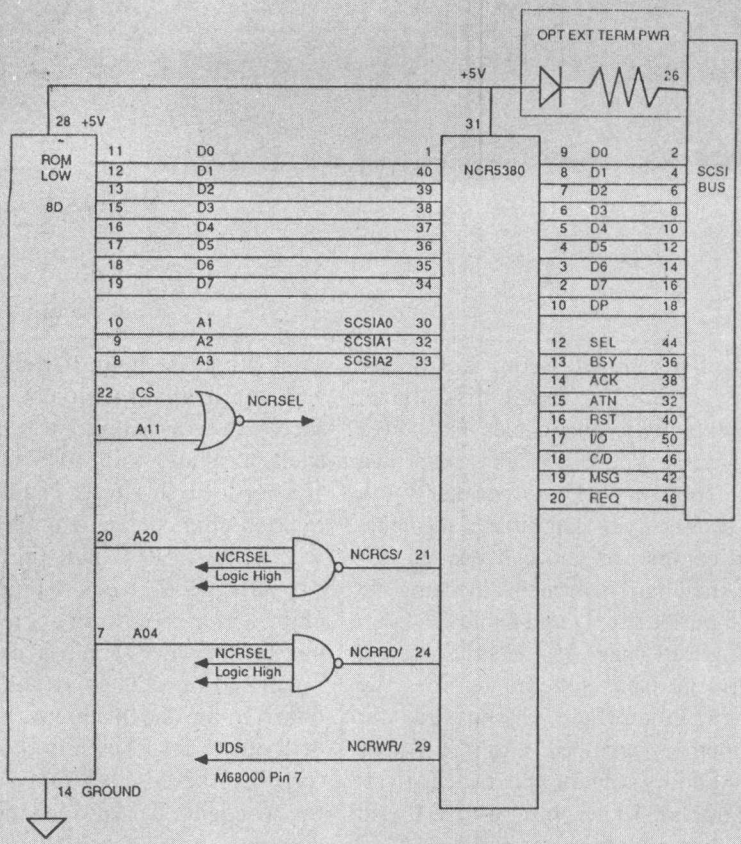

Figure 1

Mac SCSI host adaptor by DMS Design—Schematic Logic Drawing

Mac SCSI Selects

READ	WRITE	
RESET	INITIATOR START	0xE
DATA	TARGET START	0xC
STATUS	DMA START	0xA
SCSI STATUS	SELECT	0x8
TARGET CMD	TARGET CMD	0x6
MODE	MODE	0x4
INITIATOR CMD	INITIATOR CMD	0x2
SCSI DATA	SCSI DATA	0x0
0x50001X Odd Bytes	0x50002X Even Bytes	

Macintosh Memory Map

Bank	Address	Overlay	Overlay/
0	00xxxxx	Rom	Ram
1			
2	400xxxx	Rom	Rom
	50000xx	Mac SCSI	Mac SCSI
3	60xxxxx	Ram	
4	9FFFFx	SCC Read	SCC Read
5	BFFFFx	SCC Write	SCC Write
6	DFxxFF	IWM	IWM
7		VIA	VIA

Figure 2

Device Memory Maps

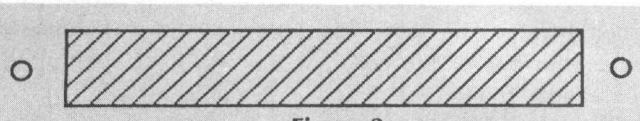

Figure 3

Template for Cut Out

Installation Outline for External Cabling

Materials Required

- Xcelite XTD-15 screwdriver with a 6-inch extension
- wooden ruler
- two cotton towels
- hand drill, ¼- and ⅛-inch drill bits
- 10-inch Mill Bastard flat file
- IC extractor or a small screwdriver to remove the 28 pin ROMs
- AMP connector 1-499970-0; AMP Ground Plane 102793-4
- paper template (see Figure 3, page 95)

Opening the Case

Disconnect the Mac from all external cables, power, mouse, keyboard etc. Place the Mac facedown on a towel to prevent marring the plastic case. Remove the five case screws with the Xcelite XTD-15 Torx screwdriver; two are located under the handle, one underneath the battery case plate, and two at the bottom of the case. Use the long edge of a wooden ruler to pry the case apart. Applying pressure to the power and I/O connectors, while lifting on the case, should release it from the faceplate. Set the foil EMI sheild aside until reassembly.

Cutting the connector slot

Using the paper template, mark the slot to be cut into the back of the Macintosh. Remove most of the plastic from the slot with a handdrill, and square off the edges with the flat file. With the slot cut, recenter the template, mark and drill the ⅛-inch holes for the 50 pin feed-thru header. Bolt the header to the inside of the case, with pin 1 on the bottom.

Removing the Motherboard ROMs

Disconnect the diskdrive and analog board cables from the motherboard; remove the motherboard from the chassis and place it on the second towel. The ROMs are in the IC sockets marked ROM HI and ROM LO. Using a chip extractor or small screwdriver, carefully remove the chip marked ROM HI. Place the ROM in the socket marked ROM HI on the Mac SCSI board, and repeat the procedure for ROM LO. Double check the orientation of the ROMs by noting that the notch on each chip points away from the 50 pin header.

Installing the Mac SCSI board

Orient the Mac SCSI board, with the 50 pin header toward the I/O connectors on the motherboard, and simply plug it into the vacated ROM sockets. Reinsert the motherboard into the chassis, taking care not to damage the 50 pin header. Reconnect the diskdrive and analog board cables, followed by the large Mac SCSI interface cable.

Closing the Mac back up

Set the Macintosh facedown and position the EMI sheild over the I/O connectors. Replace the back, making sure each side is completely seated, and secure the five holding screws.

tried this yet.

The software for MacSCSI is so far pretty simple. It consists of a routine to format the drive and a disk driver. Details of the operation of these two routines will vary a little depending upon the controller manufacturer and drive type. The Listing in this article (page 98) is for a XEBEC S1410 controller with rev D ROMs and a Seagate ST506 5Mb drive, both of which are common on the used market in Silicon Valley. As a starting point I used the RAMdisk supplied with the Aztec C Compiler release. The strategy was to use the basic RAMdisk driver as a local cache and use a simple LRU algorithm for replacement. Using the Aztec C example for the explorer desk accessory, we recoded the driver into C, for a slight loss in performance. On a 128k Mac the cache size should be set between 1 and 10. On a 512k Mac you should use some number between 30 and 300, optimally.

Other development environments and C compilers may be used as well with minor changes in the coding and installation procedures. For Aztec C the documentation on the RAMdisk and explorer desk accessory provide all the magic incantations necessary to convert the source files into an installed driver.

Questions, anyone? How large a disk, you ask? It depends in part on the software; you can reasonably handle up to about five Mb of storage on the Mac without partitioning; at ten Mb the number of files gets unwieldy. Then you'll just need more sophisticated software. Do you have to have a Fat Mac to support MacSCSI? No, but while this hard disk upgrade can work with many applications on a 128k Mac, a 512k Mac is highly recommended.

What if you don't want to take the time to do it yourself? For the lazy, the MacSCSI board is available from Fastime, P.O. Box 12508, San Luis Obispo, CA 93406 for $150.00 assembled and tested, along with machine-readable copies of the sources listed here. You can also get a complete kit, including drives from 10Mb to 110Mb, tape backup, and extended software drivers with multiple soft partitions to support the larger drives. Drivers for disk sharing and the Appletalk file server are planned or under development. Write for more information.

John L. Bass
DMS Design
P.O. Box 1456
Cupertino, CA 95014

DDJ

(Listings begin on page 98)

```
all: clean FormatSCSI ChkSCSI MountSCSI MacSCSI BootSCSI
        echo done

clean:
        rm FormatSCSI MountSCSI MacSCSI BootSCSI ChkSCSI
        rm TestSCSI MSCSI
        rm *.bak

print:
        cat makefile > .bout
        cat newpage > .bout
        cat sl41Ø.c > .bout
        cat newpage > .bout
        cat np_fmt.c > .bout
        cat newpage > .bout
        cat np_dvr.c > .bout
        cat newpage > .bout
        cat newpage > .bout

FormatSCSI:
        cc np_fmt.c
        cc sl41Ø.c
        ln -mo FormatSCSI np_fmt.o sl41Ø.o sys:lib/mixcroot.o -lc
        cprsrc DRVR 3Ø sys:system FormatSCSI
        rm np_fmt.o sl41Ø.o

ChkSCSI:
        cc np_chk.c
        cc sl41Ø.c
        ln -mo ChkSCSI np_chk.o sl41Ø.o sys:lib/mixcroot.o -lc
        cprsrc DRVR 3Ø sys:system ChkSCSI
        rm np_chk.o sl41Ø.o

MacSCSI:
        cc -bu np_dvr.c
        cc -bu sl41Ø.c
        ln -d -n MacSCSI -I 28 -R 4Ø np_dvr.o sl41Ø.o -lc -o MacSCSI
        cp -f MSCSI TestSCSI
        cprsrc DRVR 28 MacSCSI TestSCSI
        rm np_dvr.o sl41Ø.o

MountSCSI:
        cc mountscsi.c
        ln -mo MountSCSI mountscsi.o sys:lib/mixcroot.o -lc
        cprsrc DRVR 3Ø sys:system MountSCSI
        cp -f MountSCSI MSCSI
        cprsrc DRVR 28 MacSCSI MountSCSI
        rm mountscsi.o

BootSCSI:
        cc bootscsi.c
        ln -mo BootSCSI bootscsi.o sys:lib/mixcroot.o -lc
        cprsrc DRVR 3Ø sys:system BootSCSI
        rm bootscsi.o

/*
 * sl41Ø.c version 1.Ø, July 2Ø, 1985
 *
 * MacSCSI I/O routine for XEBEC S141Ø with ST5Ø6 drive and similar
 * SASI/SCSI controllers that are pre-SCSI standard. Other controller
```

```
 * drive combinations will require changes. This version was developed
 * on the Mac under Aztec C.
 *
 * Copyright 1985 by John L. Bass, DMS Design
 * PO Box 1456, Cupertino, CA 95014
 * Right to use, copy, and modify this code is granted for
 * personal non-comercial use, provided that this copyright
 * disclosure remains on ALL copies. Any other use, reproduction,
 * or distribution requires the written consent of the author.
 *
 * Sources are available on diskette from Fastime, PO Box 12508
 * San Luis Obispo, Ca 93406 -- (805) 546-9141. Write for ordering
 * information on this and other Mac products.
 */

/*
 * SASI/SCSI Command and Sense Blocks
 */
struct scsicmd {
        char    sc_cmd;                 /* command code */
        char    sc_adrH;                /* High Byte of address */
        char    sc_adrM;                /* Middle byte of address */
        char    sc_adrL;                /* Low byte of address */
        char    sc_arg;                 /* count or interleave value */
        char    sc_vendor;              /* vendor byte -- seek algorithm */
};

struct scsisense {
        char    ss_code;                /* status code */
        char    ss_adrH;                /* address when valid */
        char    ss_adrM;
        char    ss_adrL;
};

/*
 * NCR5380 registers on the MacSCSI host adapter found at
 * location 0x500000
 */
struct NCR5380 {
        char wr_data,   filla;          /* force scsi bus data */
```

(Continued on page 101)

```
        char wr_icmd,   fillb;          /* initiator command */
        char wr_mode,   fillc;          /* ncr5380 mode */
        char wr_tcmd,   filld;          /* target command */
        char wr_sele,   fille;          /* select enable */
        char wr_send,   fillf;          /* start send operation */
        char wr_trec,   fillg;          /* start target recieve */
        char wr_irec,   fillh;          /* start initiator recieve */
        char fill00,    rd_data;        /* current scsi bus data */
        char fill01,    rd_icmd;        /* initiator command status */
        char fill02,    rd_mode;        /* ncr 5380 mode */
        char fill03,    rd_tcmd;        /* target command status */
        char fill04,    rd_bstat;       /* current bus status */
        char fill05,    rd_stat;        /* chip bus and status info */
        char fill06,    rd_input;       /* input data */
        char fill07,    rd_reset;       /* reset strobe */
};

/*
 * Phase defines        TCMD                BSTAT
 */
 #define        P_DOUT  0x00            /* 0x60 data out */
 #define        P_DIN   0x01            /* 0x64 data in */
 #define        P_CMD   0x02            /* 0x68 command */
 #define        P_STAT  0x03            /* 0x6C status */
 #define        P_MOUT  0x06            /* 0x70 Message Out */
 #define        P_MIN   0x07            /* 0x74 Message In */

/*
 * Global for drive size for use by format routine
 */
long    ScsiDrvSize = 4L * 17L * 153L;

/*
 * ScsiReset -- assume this is the only host adapter in system and do
 * a hard reset of the buss and controllers.
 */

ScsiReset() {
        long cntr;
        register zero = 0;
        register struct NCR5380 *ncr = 0x500000;

        ncr->wr_data = zero;
        ncr->wr_mode = zero;
        ncr->wr_tcmd = zero;
        ncr->wr_icmd = 0x80;
        for(cntr=0x40000;cntr>0;cntr--);
        ncr->wr_icmd = zero;
}

/*
 * ScsiCmd - Select the target controller 0, build and transfer
 * the command block.
 */
ScsiCmd(opcode,lun,blk,len,ctl) {
        struct scsicmd cmd;
        long cntr;
        register zero = 0;
        register struct NCR5380 *ncr = 0x500000;
        register char *ptr;
```

(Continued on next page)

Listing One

```
        ncr->wr_tcmd = zero;                            /* select controller */
        ncr->wr_data = 1;
        ncr->wr_icmd = 0x05;
        for(cntr=0x40000;cntr>0 && (ncr->rd_bstat & 0x40) == zero;cntr--);
        ncr->wr_icmd = zero;
        cmd.sc_cmd  = opcode;           /* build scsi command block */
        cmd.sc_adrH = lun<<5;
        cmd.sc_adrM = blk>>8;
        cmd.sc_adrL = blk;
        cmd.sc_arg  = len;
        cmd.sc_vendor = ctl | 1;        /* force XEBEC ST506 Halfstep */
        ScsiOut(6,&cmd,P_CMD);          /* send cmd to ctlr */
}

/*
 * ScsiOut/ScsiIn - transfer bytes on data bus with req/ack handshake
 * In the interest of speed we ignore edge following, the controller
 * will respond within a microsecond or so during a particular phase.
 * The rest of the loop is unfolded and optimized for the Aztec C
 * compiler to generate 9 memory references per byte. Best case would
 * be 7 memory references.
 */
ScsiOut(len,ptr,phase)
register char *ptr;
{
        register high = 0x11;
        register long low = 0x01;
        register char *i,*d;
        register zero = 0;
        register struct NCR5380 *ncr = 0x500000;

        d = &ncr->wr_data;
        i = &ncr->wr_icmd;
        ncr->wr_tcmd = phase;
        ncr->wr_icmd = 0x01;
        do {
                while((ncr->rd_bstat & 0x20) == zero);    /* sync with req */
                if((ncr->rd_stat & 0x08) == zero) break;  /* if done */
                *d = *ptr; ptr+=low; *i = high; *i = low;
                *d = *ptr; ptr+=low; *i = high; *i = low;
                *d = *ptr; ptr+=low; *i = high; *i = low;
                *d = *ptr; ptr+=low; *i = high; *i = low;
                *d = *ptr; ptr+=low; *i = high; *i = low;
                *d = *ptr; ptr+=low; *i = high; *i = low;
                while((ncr->rd_bstat & 0x20) == zero);    /* sync with req */
                if((ncr->rd_stat & 0x08) == zero) break;  /* if a cmdblk */
                *d = *ptr; ptr+=low; *i = high; *i = low;
                *d = *ptr; ptr+=low; *i = high; *i = low;
        } while ((len -= 8) > 0);
        ncr->wr_icmd = zero;
        return(0);
}

ScsiIn(len,ptr,phase)
register char *ptr;
{
        register high = 0x10;
        register long one = 0x01;
        register char *i,*d;
        register zero = 0;
```

(Continued on page 104)

Listing One

```
        register struct NCR5380 *ncr = 0x500000;

        d = &ncr->rd_data;
        i = &ncr->wr_icmd;
        ncr->wr_tcmd = phase;
        do {
                while((ncr->rd_bstat & 0x20) == zero);
                if((ncr->rd_stat & 0x08) == zero) break;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
                *ptr = *d; ptr+=one; *i = high; *i = zero;
        } while ((len -= 8) > 0);
        return(0);
}

/*
 * ScsiStat - get the last two bytes to finish a command sequence
 */
ScsiStat() {
        register zero = 0;
        register struct NCR5380 *ncr = 0x500000;
        register char *ptr;
        short stat;

        ptr = &stat;
        ncr->wr_tcmd = P_STAT;
        while((ncr->rd_bstat & 0x20) == 0);
        *ptr++ = ncr->rd_data;
        ncr->wr_icmd = 0x10;
        ncr->wr_icmd = zero;
        ncr->wr_tcmd = P_MIN;
        while((ncr->rd_bstat & 0x20) == 0);
        *ptr++ = ncr->rd_data;
        ncr->wr_icmd = 0x10;
        ncr->wr_icmd = zero;
        ncr->wr_tcmd = zero;
        return(stat);
}

/*
 * ScsiFmt -- Do a default drive format (ST506)
 */
ScsiFmt() {
        ScsiCmd(0x04,0,0,12,0);         /* issue format command */
        return(ScsiStat());                     /* return status */
}

/*
 * ScsiRead and ScsiWrite -- do the I/O for a specified sector
 * and the related data.
 */
ScsiRead(sector,data)
char *data;
{
```

```
        ScsiCmd(0x08,0,sector,1,0);     /* issue read command */
        ScsiIn(512,data,P_DIN);         /* transfer in data */
        return(ScsiStat());                     /* return status */
}
ScsiWrite(sector,data)
char *data;
{
        ScsiCmd(0x0A,0,sector,1,0);     /* issue write command */
        ScsiOut(512,data,P_DOUT);       /* transfer out data */
        return(ScsiStat());                     /* return status */
}

/*
 * MacSCSI non-partitioned format routine.
 * This version was developed on the Mac under Aztec C.
 *
 * Copyright 1985 by John L. Bass, DMS Design
 * PO Box 1456, Cupertino, CA 95014
 * Right to use, copy, and modify this code is granted for
 * personal non-comercial use, provided that this copyright
 * disclosure remains on ALL copies. Any other use, reproduction,
 * or distribution requires the written consent of the author.
 *
 * Sources are available on diskette from Fastime, PO Box 12508
 * San Luis Obispo, Ca 93406 -- (805) 546-9141. Write for ordering
 * information on this and other Mac products.
 */

struct Volume {
                short   drSigWord;      /* should be $D2D7 */
                long    drCrDate;       /* date and time of initialization */
                long    drLsBkUp;       /* date and time of last backup */
                short   drAtrb;         /* volume attributes */
                short   drNumFls;       /* # of files in file directory */
                short   drDirSt;        /* first logical block of file dir */
                short   drBlLen;        /* # of logical blocks in file dir */
                short   drNmAlBlks;     /* # of allocation blocks on volume */
                long    drAlBlkSiz;     /* size of allocation blocks */
                long    drClpSiz;       /* # of bytes to allocate */
                short   drAlBlSt;       /* logical block number of first
                                                        allocation block */
                long    drNxtFNum;      /* next unused file number */
                short   drFreeBks;      /* # of unused allocation blocks */
                char    drVN;           /* length of volume name */
                char    drFill[512-37]; /* volume name & start of alloc. map */
};
struct  Volume  v;

char    zeros[512];                             /* block worth of nulls */

long    ScsiDrvSize;                    /* number of 512 byte sectors */

main() {
                char *p;
                int     i;

#ifdef hack
        printf("Scsi reset\n");
        ScsiReset();
        printf("Drive being formatted\n");
        if(ScsiFmt()) {
                printf("Format Failed\n");
                exit(1);
        }
```

Mac Toolbox Listing (Listing Continued, text begins on page 94)

```
#endif
        v.drSigWord = 0xd2d7;
        v.drCrDate  = v.drLsBkUp = 0;
        v.drAtrb    = 0;
        v.drNumFls  = 0;
        v.drNxtFNum = 1;
        v.drDirSt   = 4;
        printf("Directory Start:      %6.6d\n",v.drDirSt);
        v.drBlLen   = 28;
        printf("Directory BlkLen:     %6.6d\n",v.drBlLen);

        v.drAlBlSt = v.drDirSt + v.drBlLen;
        printf("First Allocation Blk: %6.6d\n",v.drAlBlSt);

        v.drAlBlkSiz = 512L * (ScsiDrvSize/640L + 1L);
        v.drClpSiz  = v.drAlBlkSiz;
        printf("Allocation BlockSize: %6.6ld\n",v.drAlBlkSiz);

        v.drNmAlBlks = (ScsiDrvSize-v.drAlBlSt)/(v.drAlBlkSiz>>9);
        v.drFreeBks = v.drNmAlBlks;
        printf("Allocation Blocks:    %6.6d\n",v.drNmAlBlks);

        for(v.drVN = 0,p="MacSCSI";*p;p++) v.drFill[v.drVN++] = *p;

        printf("Initializing Drive\n");
        ScsiWrite(0,zeros);
        ScsiWrite(1,zeros);

        if(ScsiWrite(2,&v)) {
                printf("Write of config block failed\n");
        }
        for(i=3;i<ScsiDrvSize;i++) if(ScsiWrite(i,zeros)) {
                printf("Write failed at block %d\n",i);
        }
        printf("Checking Drive\n");
        for(i=0;i<ScsiDrvSize;i++) if(ScsiRead(i,zeros)) {
                printf("Read failed at block %d\n",i);
        }
        printf("Format completed ok\n");
}
main()
{
        printf("Exit code %d\n",OpenDriver("\P.MacSCSI"));
}

main()
{
        *(int *)0x210 = 5;              /* boot drive to MacSCSI */
}
```

End Listing

Due to a lack of space in this issue the listing for the Mac SCSI disk driver (np_dvr.c) will be published in the October issue.

Last month (September 1985, #107 page 94) John Bass described how to put together your own hard-disk interface for the Macintosh for less than $500. The following listing, np_dvr.c, is the driver routine.

Mac Toolbox Listing

```
/*
 * np_dvr.c version 1.0, July 20, 1985
 * MacSCSI non-partitioned driver routine.
 * This version was based on the ramdisk.asm example found in
 * Aztec C 1.06 and was developed on the Mac under Aztec C.
 *
 * The changes are Copyright 1985 by John L. Bass, DMS Design
 * PO Box 1456, Cupertino, CA 95014
 * Right to use, copy, and modify this code is granted for
 * personal non-commercial use, provided that this copyright
 * disclosure remains on ALL copies. Any other use, reproduction,
 * or distribution requires the written consent of the author.
 *
 * Sources are available on diskette from Fastime, PO Box 12508
 * San Luis Obispo, Ca 93406 -- (805) 546-9141. Write for ordering
 * information on this and other Mac products.
 */

#asm
;:ts=8
;
DRVNUM  equ     5               ;change this if conflicts with others
;
Start
        dc.w    $4f00           ;locked, read, write, ctrl, status
        dc.w    0               ;no delay
        dc.w    0               ;no events
        dc.w    0               ;no menu
        dc.w    open-Start      ;open routine
        dc.w    rdwrt-Start     ;prime routine
        dc.w    control-Start   ;control routine
        dc.w    status-Start    ;status routine
        dc.w    tst-Start       ;close routine
        dc.b    8
        dc.b    ".MacSCSI"      ;name of driver
        ds      0               ;for alignment
;
MAXMAP          equ     640
;
drBlLen         equ     16
drNmAlBlks      equ     18
drAlBlkSiz      equ     20
drClpSiz        equ     24
drAlBlSt        equ     28
drFreeBks       equ     34
;
template
        dc.w    $d2d7           ;signature word
        dc.l    0               ;init time
        dc.l    0               ;backup time
        dc.w    0               ;attributes
        dc.w    0               ;# of files
        dc.w    4               ;first dir block
        dc.w    0               ;# of dir blocks
        dc.w    0               ;# of allocation blocks total
        dc.l    0               ;bytes/allocation block
        dc.l    0               ;bytes/allocation call
        dc.w    4               ;first data block
        dc.l    1               ;next free file #
        dc.w    0               ;# of unused allocation blocks
        dc.b    7               ;length of name
```

```
        dc.b    "MacSCSI"               ;name of drive
        ds      0                       ;for alignment
;
stabuf
        dc.w    0                       ;current track
        dc.b    0                       ;write protect in bit 7
        dc.b    1                       ;disk in place
        dc.b    1                       ;drive installed
        dc.b    0                       ;sidedness of drive
        dc.l    0                       ;next queue entry
        dc.w    0                       ;unused
        dc.w    DRVNUM                  ;drive number
drvref  dc.w    0                       ;driver ref num
        dc.w    0                       ;file system ID
        dc.b    0                       ;sidedness of disk
        dc.b    0                       ;needs flush flag
        dc.w    0                       ;error count
;
diskbase_
        dc.l    0                       ;base of disk memory
;
openflg
        dc.w    0                       ;once only open flag
;
open
        lea     openflg,a0              ;get once only flag
        tst.w   (a0)                    ;is it open already?
        bne     skip                    ;yes -- just exit
        st      (a0)                    ;set once only flag

        move.l  #14,d0                  ;size of drive queue entry
        dc.w    $a51e                   ;NewPtr - system heap
        clr.w   10(a0)                  ;local file system
        move.l  #DRVNUM,d0              ;drive number 5
        swap    d0
        move.w  24(a1),d0               ;driver reference number
        lea     drvref,a2               ;get address of driver reference number
        move.w  d0,(a2)                 ;set up status buffer
        dc.w    $a04e                   ;AddDriver

        move.l  (a1),-(sp)              ;push handle to resource
        dc.w    $a992                   ;DetachResource

        link    a6,#-50                 ;allocat the block on the stack
        move.l  sp,a0                   ;setup as arg to mountvol trap
        move.w  #DRVNUM,22(a0)          ;set our drive number in block
        dc.w    $a00f                   ;ask for it to be mounted
        unlk    a6                      ;clear block from stack

skip
        move.l  #0,d0                   ;set non-error return
        rts                             ;exit back to caller
;
rdwrt
        movem.l d0-d7/a0-a6,-(sp)       ;save regs
        move.l  a0,Pbp_
        move.l  a1,Dp_                          ;save arguments
        rts
restore_
        move.l  Dp_,a1                          ; pass DCEptr
        rts
#endasm

#include        <memory.h>
#include        <resource.h>
#include        <desk.h>
#include        <pb.h>
/*
 * varibles used by asm routines
```

```
 * Dp and Pbp are device driver arguments handled by save/restore
 * ptr and dvrarg needed by AddDriver call
 */

DCEPtr          Dp;
ParmBlkPtr      Pbp;

/*
 * Local Stuff
 */

int     DvrState;
        jsr     ScsiRdWr_               ;Call the C stuff
        movem.l (sp)+,d0-d7/a0-a6       ;restore regs
        bra     tst                     ;exit via IOdone

control
        cmp.w   #1,26(a0)       ;is it KillIO
        bne.s   tst             ;no, exit
        move.l  #0,d0
        rts                     ;just return
;
status
        cmp.w   #8,26(a0)       ;is it status request??
        bne.s   tst
        movem.l a0/a1,-(sp)     ;save regs
        lea     28(a0),a1       ;get dest address
        lea     stabuf,a0
        move.l  #22,d0          ;move 22 bytes
        dc.w    $a02e           ;BlockMove
        movem.l (sp)+,a0/a1     ;restore regs
;
tst
        move.l  #0,d0                   ;okay return
        movem.l d4-d7/a4-a6,-(sp)       ;save regs going into IOdone
        move.l  $8fc,a0                 ;get IOdone address
        jsr     (a0)                    ;call IOdone
        movem.l (sp)+,d4-d7/a4-a6       ;restore them after IOdone
        rts                             ;and jump to it

        public  _Uend_,_Dorg_,_Cend_

save_
        lea     Start+(_Uend_-_Dorg_)+(_Cend_-Start),a4 ; setup baseadr
/*
 * ScsiRdWr -- do a driver read/write function.
 */
ScsiRdWr() {
        register struct ioParam *ip;
        long    len,part;
        short   blkno;
        char    *addr;
        save();
        ip = & Pbp->u.iop;
        part = Dp->dCtlPosition & 0xFFFFFE00;
        len = (ip->ioReqCount + 511) & 0xFFFFFE00;
        addr = ip->ioBuffer;
        while(len >= 512) {
                blkno = part>>9;
                if((Pbp->ioTrap & 0xff) == 2) {
                        ScsiRead(blkno, addr);
                 } else {
                        ScsiWrite(blkno, addr);
                }
                len -= 512;
                addr += 512;
                part += 512;
        }
        ip->ioActCount = ip->ioReqCount;
        restore();                      /* exit to I/O Done */
}
```

End Listings