Pascal
#16: Driver to Have Two Volumes on One 3.5” Disk

Revised by: Guillermo Ortiz, Cheryl Ewy & Dan Strnad November 1988
Written by: Guillermo Ortiz October 1986

This Technical Note discusses how to install a driver to have more than one volume on a 3.5” 800K disk under Apple II Pascal.

For the sake of simplicity,.we will limit the discussion to the following case: we want to have two 400K volumes on the boot 3.5” disk. For such a scenario, Unit #4 occupies the first 800 blocks and Unit #20 uses blocks 800 to 1599 as shown here:

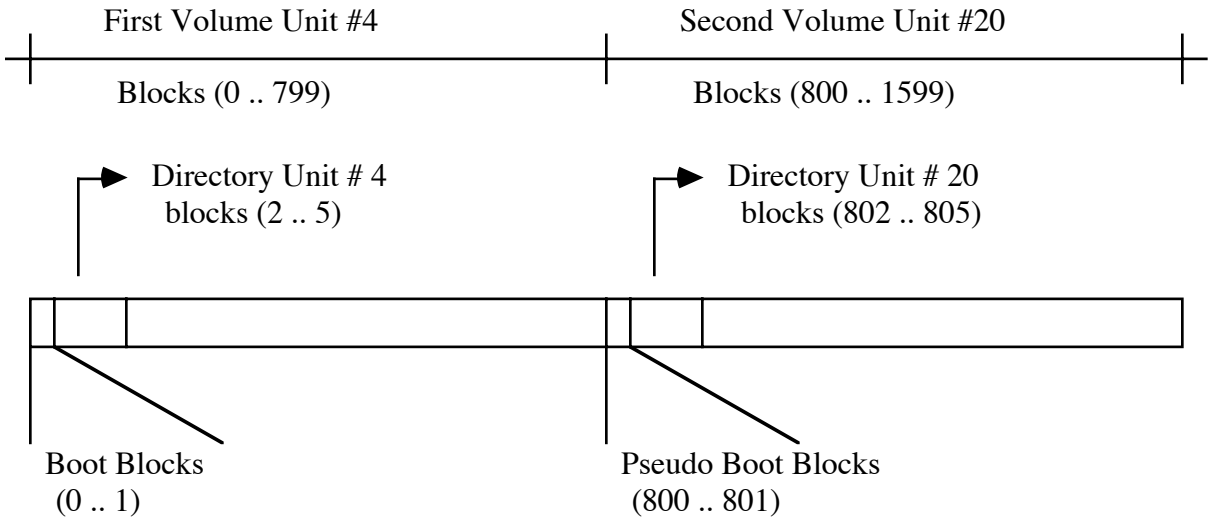

Figure 1–Block Diagram for 3.5” Disk

There are four calls a device driver has to handle, `UNITCLEAR`, `UNITSTATUS`, `UNITREAD`, and `UNITWRITE`. For the first one, our driver will only return since the device is already on-line. For a blocked device, `UNITSTATUS` returns the number of blocks available, in this case `UNITSTATUS (20) = 800`.

In the case of `UNITREAD` and `UNITWRITE`, all the driver has to do is add the offset of 800 to the number of the block requested then jump to the BIOS routine with the unit number set to four. Our driver is basically a dispatcher that directs the disk access to the proper blocks.

When this driver is present, the application must be very careful about making sure the right disk is in the drive when accessing the second volume; **any** access to Unit #20 could **damage** a normal volume present in the drive.

Once the driver is ready, it is necessary to format a disk with the special directories. With the listings for the driver we have included the source of a sample formatting program.

Once the disk is ready we proceed to transfer all system files to it including SYSTEM.ATTACH, ATTACH.DRIVERS (containing our driver), and ATTACH.DATA. This last file reflects the following information:

Driver Name - `FAKEDISK` - Not Aligned
Attached to #20 {Can change if desired}
Unit #s to be init at boot time - 20
This driver CAN be placed in the first HiRes screen {Change if needed}
This driver CAN be placed in the second HiRes screen {Change if needed}
This driver does not use interrupts
Driver does not have transient initialization code

The code has comments that explain it fairly well; for more information on drivers in general and how to use the attach tools please refer to *Apple II Pascal Device and Interrupt Support Tools*.

```
;
;       Disk Driver
;       by Guillermo Ortiz
;       03/25/86
;
;       This driver will allow splitting a 3.5 disk in two pieces of 400K
;       each, therefore permitting more than 77 files per disk. It
;       is required to "format" the disk with two directories, one at
;       block 0 .. 5 and the other at block 800 .. 805, each with a
;       length of 800 blocks. Names must be different!

;       The ancient admonition:
;
;       This is a sample!
;       No claims are made regarding the fitness of this code for
;       any particular purpose.

ROUTINE .EQU    02              ; For indirect jumping
RETURN  .EQU    04              ; Back to Pascal
BUFF    .EQU    06              ; Where to put stuff

        .PROC   FAKEDISK

;       At this level we could have some code to differentiate
;       between different pseudo volumes if we had more than
;       two pseudo-volumes per disk.
;       In this example we use Unit # 20 for the second part.
;       Using units 13 and up let us keep the "standard" drives available
;       In any UNIT call X Register contains the type of call
;       as follows:

        CPX     #04
        BEQ     STATUS          ; X = 4
        CPX     #02
        BEQ     INIT            ; X = 2
```

```
        STA     TEMP1
        STY     TEMP1+1         ; Saving A, Y and X
        STX     TEMP1+2         ;   for future use
```

```
;       We make the assumption that the disk split is the
;       System Volume, so we get the logical volume number for
;       Unit # 4 from the DISKNUM table;
;       see Apple // Pascal Device and Interrupt
;       Support Tools manual for details.

        TSX                     ; Gimmie the stack pointer
        LDA     0FEB6           ; Logical volume for boot disk
        STA     109,X           ;   so read from that disk

;       Our fiddling is complete now let's finish checking
;       the call in order to make the jump

        LDA     TEMP1+2         ; X contains the call code
        BEQ     READ            ; X = 0
        CMP     #01
        BEQ     WRITE           ; X = 1

;       Here we could have
;       instructions to report some undefined control code.
;       This driver will only CRASH!!!

        BRK                     ; Bumm!!!

;       Now the real stuff

READ    .EQU    *
        JSR     SETUP           ; Modify the stack
        LDY     #19.            ; Index for Reading from disk
        BNE     GET             ; Nice way of jumping

WRITE   .EQU    *
        JSR     SETUP           ; Modify the stack
        LDY     #16.            ; Index for WRITE to CONSOLE

GET     LDA     @0E2,Y          ; $E2 contains a pointer to the jump vector
        STA     ROUTINE         ; Set low byte of address
        INY
        LDA     @0E2,Y          ; Get high byte of address
        STA     ROUTINE+1       ;   and set it off

        LDX     TEMP1+2         ; Restore
        LDY     TEMP1+1         ;   all registers
        LDA     TEMP1           ;     before jump

        JMP     @ROUTINE        ; and Go!
```

```
;       INIT will only pass back the no_error IORESULT

INIT    .EQU    *
        LDX     #00             ; No error
        RTS                     ;   Go back

STATUS  PLA                     ; Get
        STA     RETURN          ;   return
        PLA                     ;     address
        STA     RETURN+1
        PLA                     ; Get
        STA     BUFF            ;   Pascal
        PLA                     ;     Buffer
        STA     BUFF+1          ;       address
        PLA                     ; Dump control
        PLA                     ;   word
        LDY     #00
        LDA     #20             ; Set
        STA     @BUFF,Y         ;   the number of blocks
        INY                     ;     to
        LDA     #03             ;       800
        STA     @BUFF,Y
        LDX     #00
        LDA     RETURN+1        ; and
        PHA
        LDA     RETURN
        PHA
        RTS                     ;   Return!


;       To any request for READ/WRITE we'll add 800 to the
;       number of the block needed.

SETUP   .EQU    *
        LDA     103,X           ; Get Block number low
        CLC                     ; Set up for addition
        ADC     #20             ;   Offset block count by 800
        STA     103,X           ;     and restore
        LDA     104,X           ; Get Block number high
        ADC     #03             ;   800 = $320
        STA     104,X           ;     and restore
        RTS                     ; Go back

TEMP1   .BLOCK  3               ; Temporary storage area


        .END
```

The driver requires that the disk be formatted in a special way. Run the following program to create your volume.

```
program REFORMAT;

{By Guillermo Ortiz
     03/27/86
}


{This program takes a newly formatted 3.5 disk and lays down two
 directories transforming the volume into two 400K pseudo-volumes to be
 used with the driver FAKEDISK which assigns Unit # 20 to the second
 part of the disk.
}
```

```
CONST   MAXDIR  = 77;   {Max number of files per volume}
        VIDLENGTH = 7;  {Max chars in volume name}
        TIDLENGTH = 15; {Max chars per file ID}
        FBLKSIZE = 512; {Number of bytes per block}
        DIRBLK = 2;     {We are reading the directory}

type    daterec = packed record
                    month:0..12;        {0 --> Meaningless date}
                    day:  0..31;        {Day of month}
                    year:0..100         {100 --> dated volume is temp}
                  end;

        vid = string [vidlength];       {Volume ID}
        dirrange = 0 .. maxdir;         {Number of files on disk}
        tid = string[tidlength];        {File ID}
        filekind = (untypedfile,xdskfile,codefile,textfile,infofile,
                    datafile,graffile,fotofile,securdir);

{Now the real directory layout}
        direntry =
          packed record
            dfirstblk:integer;          {1st physical disk address}
            dlastblock:integer;         {block after last used block}
            case dfkind:filekind of
              securdir,untypedfile:     {Volume info only in dir[0]}
               (filler1: 0..2048;       {Waste 13 bits}
                dvid:    vid;           {Name of volume}
                deovblk: integer;       {Last block in volume}
                dnumfiles:dirrange;     {Number of files in directory}
                dloadtime:integer;      {Time of last access}
                dlastboot:daterec);     {Most recent date setting}
              xdskfile,codefile,textfile,infofile,datafile,
              graffile,fotofile:        {Regular file info}
               (filler2: 0..1024;       {Waste 12 bits}
                status:  boolean;       {For filer wildcards}
                dtid:    tid;           {Name of file}
                dlastbyte:1..fblksize;  {Bytes in last block of file}
                daccess: daterec)       {Date of last modification}
          end;  {Of the whole directory record}

        directory = array [dirrange] of direntry;


var     dirinfo:directory;              {The directory goes here}
        UNITNUM:INTEGER;
        CH:CHAR;


PROCEDURE DOSTUFF;
{Function CHECK will read the directory from a freshly formatted
 3.5 disk, then DOSTUFF will make changes so it has only 800 blocks and
 a name HALFONE: and will write it back to block 2; then we will
 change the name to HALFTWO: and will write to block 802 as
 the directory for our second pseudo-volume.
}

BEGIN
  with dirinfo[0] do
    begin
      deovblk:=800;  {Cut it in half}
      dvid:='HALFONE';
    end;
  unitwrite(UNITNUM,dirinfo,sizeof(dirinfo),dirblk); {Put back main directory}
  DIRINFO[0].DVID:='HALFTWO';
  unitwrite(UNITNUM,dirinfo,sizeof(dirinfo),dirblk+800) {Write second dir.}
```

```
end; {Of DOSTUFF}


FUNCTION CHECK:BOOLEAN;

{Reads the directory from the target disk, if possible, warns the user
 of the certain destruction of the current directory and checks the
 size of the volume so that the program doesn't use other than 3.5
 disks.
 }


BEGIN
  CHECK:=FALSE;
  DIRINFO[0].DLASTBLOCK:=-999;  {Make sure we read from a disk}
  UNITREAD(UNITNUM,DIRINFO,SIZEOF(DIRINFO),DIRBLK);
  IF DIRINFO[0].DLASTBLOCK= 6 THEN  {IS THIS A PASCAL DISK?}
    BEGIN
      IF DIRINFO[0].DEOVBLK <> 1600 THEN
        BEGIN
          WRITELN('SORRY THIS PROGRAM IS INTENDED FOR 3.5 DISKS ONLY');
          EXIT(CHECK)
        END;
      WRITE('WE ARE ABOUT TO PERMANENTLY DESTROY    ');
      WRITELN(DIRINFO[0].DVID,':');
      WRITE('IS IT OK? --> ');
      REPEAT
        READ(KEYBOARD,CH)
      UNTIL CH IN ['Y','N','n','y'];
      WRITELN(CH);
      IF CH IN ['Y','y'] THEN
        CHECK:=TRUE
    END
  ELSE
    BEGIN
      WRITELN;
      WRITELN;
      WRITELN('CAN NOT READ DIRECTORY')
    END
END {OF CHECK};


PROCEDURE GETNUM;

{Prompts the user for the Unit Number of the target disk,
 checks the validity of the input and returns when provided with
 a reasonable value.
 }
```

```
VAR     I:INTEGER;

BEGIN
  WRITELN;
  WRITELN('PLEASE ENTER THE NUMBER OF THE UNIT CONTAINING THE DISK');
  WRITE('TO BE REFORMATTED (PRESS <ESCAPE> TO EXIT) --> ');
  UNITNUM:=0;
  REPEAT
    BEGIN
      WRITE(CHR(5));    {Cursor ON}
      READ(CH);         {For the prompt}
      WRITE(CHR(6));    {and then OFF for speed and elegance(?)}
      IF EOLN THEN
        IF (UNITNUM IN [4,5,9..12]) THEN
          EXIT(GETNUM)
        ELSE
          FOR I:= 1 TO 32 - UNITNUM DO  {Kind of crude but ...}
            WRITE(CHR(8));              {to go back to the same place}
      IF ORD(CH) = 27 THEN
        BEGIN
          WRITELN;
          WRITELN('YOU ASKED FOR IT!!!');
          WRITE(CHR(5));        {Turn cursor ON before we exit}
          EXIT(PROGRAM)
        END;
      IF (ORD(CH) = 8) AND (UNITNUM > 0) THEN
        BEGIN
          IF UNITNUM < 10 THEN
            UNITNUM:=0
          ELSE
            UNITNUM:=UNITNUM DIV 10;
          WRITE(CHR(8),' ',CHR(8))      {To delete previous entry}
        END
      ELSE
        BEGIN
          IF (UNITNUM = 0) AND (CH IN ['1','4','5','9']) THEN
            UNITNUM:=ORD(CH)-ORD('0')
          ELSE
            IF (UNITNUM=1) AND (CH IN ['0','1','2']) THEN
              UNITNUM:=10*UNITNUM+ORD(CH)-ORD('0')
            ELSE
              IF ORD(CH) > 31 THEN
                WRITE(CHR(8),' ',CHR(8))        {Unwanted stuff,so ...}
        END                                     {get rid of it.       }
    END
  UNTIL FALSE;                                  {No Exit here.}
  WRITELN
END {OF GETNUM};


BEGIN                                   {main}
  WRITELN;
  WRITELN;
  WRITELN('WE ARE ABOUT TO REFORMAT A VOLUME SO IT WILL CONTAIN TWO');
  WRITELN('400K PSEUDO-VOLUMES. MAKE SURE YOU MARK THE DISK CLEARLY');
  WRITELN('SO YOU DON''T FORGET');
  WRITELN;
  WRITELN;
  REPEAT
    GETNUM
  UNTIL CHECK;
  DOSTUFF;
  WRITE(CHR(5));                {Don't forget to turn cursor ON}
  writeln;
  WRITELN('AWAAAAAY!!!')
end.
```

If two volumes are not enough, you can modify this example to support more than two per disk; the key is to keep in mind that when the call comes to the driver, the accumulator contains the number of the Unit the for which the call is intended. After checking this number the driver could decide what offset it has to add to access the correct volume.

Of course the formatter program would have to change accordingly, laying down the directories for the new volumes with the appropriate names and sizes.

The same scheme can be applied to any device that Pascal can directly recognize (i.e., the Apple Memory Expansion Card, ProFile hard disk, etc.).

Further Reference

- *Apple II Pascal Device and Interrupt Support Tools*