DOS 3.3

DOUBLE HI-RES GRAPHICS I

Owners of the Apple //e with an extended 80 column card have a whole new world of graphics available to them. Double Hi-Res Graphics has a resolution width of 560 dots and 16 Hi-Res colors. This is the first in a series of Graphics Workshop articles which will explore this new territory.

by Robert R. Devine
1415 W. 19th
El Dorado, AR 71730

Shortly after I traded in my old Apple II for a shiny new //e, I began hearing strange rumors about some new 560-dot "super Hi-Res" capabilities. Needless to say, due to my interest in Hi-Res graphics, this caught my immediate attention. I WANTED IT!!!

The problem, however, was that aside from the people passing the rumors, there didn't seem to be anyone (including my local dealers) who really knew anything about it. When I finally got the needed hardware, I spent a full week trying to follow the manual's instructions so that I could see what this new Double Hi-Res looked like. After almost a dozen calls to Apple in Cupertino, Dallas, Chicago, and Charlotte N.C., I hadn't been able to find anyone who even knew how to turn on the 560-dot display.

Finally, after two weeks I received a call from someone at Apple who was able to help. However, by that time I had managed, on my own, to access the Double Hi-Res screens. One of the many things I've always liked about Apple has been the excellent, easy-to-understand documentation that comes with their equipment; however, I really think that someone was "out to lunch" on this one.

What we're going to do in this series is explore this new world of **Double Hi-Res graphics**, see how it works, and look at the strengths and weaknesses in this new area. Despite my somewhat negative start, I think that Double Hi-Res is great, and once you understand how it works, I'm sure you'll like it too.

As we go along, we'll develop a special **DHR driver** program to use with the system, as well as special utilities that will be useful in accomplishing some of the things that we currently take for granted.

"Nibbling at the Double Hi-Res block shapes will require a greater understanding of the physical design of the Hi-Res screens."

Required Hardware

This series isn't for everyone; if you don't have the proper hardware, you might just as well turn back to the table of contents and spend your time pondering one of the many other interesting and informative topics in this issue.

The first thing you'll need is a **model //e Apple computer**; models II and II Plus need not apply.

The next thing you'll need is to be sure that your Apple has a **revision B or later motherboard**, as you can't generate Double Hi-Res graphics on the version A motherboard. To find out which version you have, open up your Apple and look at the numbers printed on the circuit board, just behind slot 3. The number you're looking for is 820-0064-(revision #).

If yours is a revision B or later, you're all set. If the motherboard in your Apple is an A, then you'll need to have it replaced by your Apple dealer. At present, Apple has an arrangement with qualified dealers to replace the motherboard with the revision B at **no charge to you**. The replacement is a simple 10-minute job, and what you'll get really amounts to a whole new computer, as they'll replace everything except the case, power supply, and keyboard. Not a bad deal if you ask me, especially if your Apple is out of warranty.

The final item you'll need is the **Apple extended 80-column card**, which can be installed in the auxiliary slot in your Apple. The card will need to have a jumper installed on the two Molex-type pins on the card. This card will add 80-column display capabilities to your Apple, as well as provide an additional 64K of usable RAM.

Benefits of a Hi-Res Monitor

One thing that you may soon consider purchasing is a Hi-Res monitor. If you're using a regular color TV with your Apple, you will find that the 80-column text display will be virtually unreadable. I now use two monitors — a Hi-Res green monitor for my text work (fantastic for 80-column display, and very helpful in graphics development since you can see all the individual dots rather than blocks of color), and a regular color TV for my graphics work. The next thing on my shopping list is a Hi-Res **color** monitor to replace my color TV.

The reason for my commenting on the type of monitor you use is this: The new 560-dot Double Hi-Res screen can be used for either full-screen graphics or graphics mixed with four lines of 80-column text. If you're writing or using a Double Hi-Res program with mixed text, and using a regular color TV, the graphics will be acceptable — but the 80-column text won't be readable. Therefore, your choices are: stick to full-screen graphics with simulated text, get a Hi-Res color monitor, or forget color by using a Hi-Res green monitor or black-and-white TV.

which range from $2000 through $3FFF. It is this range of memory which is used for the other 8192 bytes that are needed.

The result is that they've put 16,384 bytes of Hi-Res screen data into 8,192 memory addresses. Cute, guys...real cute.

Double Hi-Res Screen Dimensions

We now realize that the Double Hi-Res screen is really **80-bytes/columns wide**; however, it is only **40 addresses wide**. The horizontal **bytes/columns are numbered 0-79**; however, the horizontal **addresses are numbered 0-39**. The important concept to understand is that each address holds two bytes of data. One byte of screen data is held in a main memory address, and a different byte of screen data is held in the exact same address in auxiliary memory; therefore, each address is able to display two bytes of data or 14 adjacent dots on the screen.

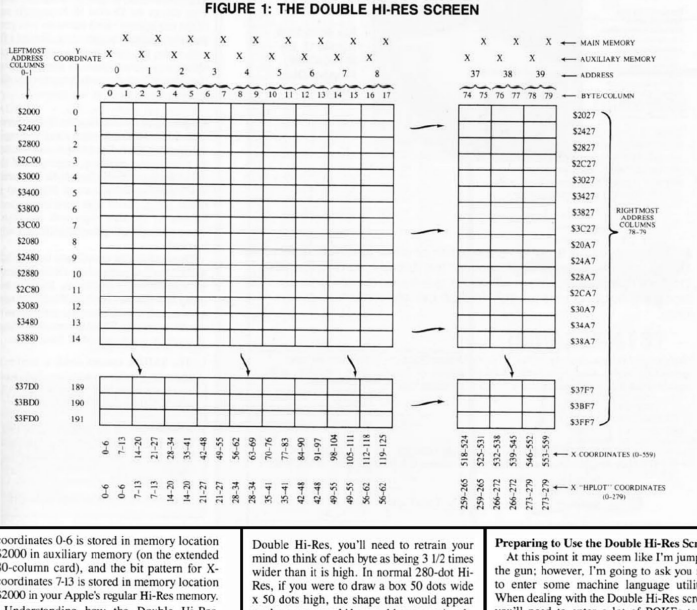
"...you'll need to retrain your mind to think of each byte as being 3 1/2 times wider than it is high."

As the graphics display circuitry inside your Apple builds the 560-dot screen display, it fetches bytes from these duplicate addresses simultaneously and displays them sequentially. The byte from auxiliary memory is first placed on the screen in the proper **even** byte/column (0,2,4...through 78). Then the byte from main memory is placed in the adjacent **odd** byte/column (1,3,5...through 79). Therefore, all even-numbered columns will contain data from auxiliary memory, and all odd-numbered columns will contain data from your Apple's normal Hi-Res memory.

To make things a bit easier to understand, let's refer to **Figure 1**, which is a representation of the Double Hi-Res screen. As you look at it you will notice that the screen addresses used are exactly the same as those used in regular 280-dot Hi-Res graphics. The only real difference is that while normal Hi-Res is 40 addresses **and** 40 data bytes wide, Double Hi-Res is 40 addresses and 80 data bytes/columns wide.

If you look at the top leftmost corner of the screen, you will note that memory location $2000 contains the screen information for **both** columns 0 and 1. The bit pattern for X-

FIGURE 1: THE DOUBLE HI-RES SCREEN



coordinates 0-6 is stored in memory location $2000 in auxiliary memory (on the extended 80-column card), and the bit pattern for X-coordinates 7-13 is stored in memory location $2000 in your Apple's regular Hi-Res memory.

Understanding how the Double Hi-Res screen is laid out is essential to being able to work this system, so I would suggest that you become familiar with **Figure 1** before proceeding further.

A Square Is a Rectangle... But a Rectangle Is Not a Square

In our drawing of the Double Hi-Res screen, we have shown each byte of memory as a square. As each of us learned to work with normal Hi-Res, we came to understand that each byte of data is really seven times wider than it is high. Now that you're getting into Double Hi-Res, you'll need to retrain your mind to think of each byte as being 3 1/2 times wider than it is high. In normal 280-dot Hi-Res, if you were to draw a box 50 dots wide x 50 dots high, the shape that would appear on the screen would be roughly square; in other words, the width and height would be about the same.

If you were to draw that same 50 dot x 50 dot shape on the Double Hi-Res screen, the resulting shape would be roughly twice as high as it is wide. This is because each Double Hi-Res dot is roughly twice as high as it is wide. This means that as you go about designing your shapes for Double Hi-Res, you'll need to use two squares (dots) across on your graph paper to equal one square (dot) of height. This will make for some rather strange looking shapes as you draw them on paper; however, they should look properly proportioned once you've placed them on the screen.

Preparing to Use the Double Hi-Res Screen

At this point it may seem like I'm jumping the gun; however, I'm going to ask you now to enter some machine language utilities. When dealing with the Double Hi-Res screen, you'll need to enter a lot of POKEs to get things going. You'll also need some special utilities to simulate Applesoft commands. These routines will prove quite useful as we proceed.

The routines that we will now begin to build will be the first parts of the DHR driver that we will develop in this series. If you've been going through the Graphics Workshop series with us, you'll remember that the first part of our driver was the YTABLE routines. This set of routines is comprised of three parts:

LISTING 1: YTABLE HEX DUMP

```
9464- A4 06 B1 CE
9468- 85 26 B1 EE 85 27 60 A9
9470- 80 85 CE A9 94 85 CF A9
9478- 40 85 EE A9 95 85 EF 60
9480- 00 00 00 00 00 00 00 00
9488- 80 80 80 80 80 80 80 80
9490- 00 00 00 00 00 00 00 00
9498- 80 80 80 80 80 80 80 80
94A0- 00 00 00 00 00 00 00 00
94A8- 80 80 80 80 80 80 80 80
94B0- 00 00 00 00 00 00 00 00
94B8- 80 80 80 80 80 80 80 80
94C0- 28 28 28 28 28 28 28 28
94C8- A8 A8 A8 A8 A8 A8 A8 A8
94D0- 28 28 28 28 28 28 28 28
94D8- A8 A8 A8 A8 A8 A8 A8 A8
94E0- 28 28 28 28 28 28 28 28
94E8- A8 A8 A8 A8 A8 A8 A8 A8
94F0- 28 28 28 28 28 28 28 28
94F8- A8 A8 A8 A8 A8 A8 A8 A8
9500- 50 50 50 50 50 50 50 50
9508- D0 D0 D0 D0 D0 D0 D0 D0
9510- 50 50 50 50 50 50 50 50
9518- D0 D0 D0 D0 D0 D0 D0 D0
9520- 50 50 50 50 50 50 50 50
9528- D0 D0 D0 D0 D0 D0 D0 D0
9530- 50 50 50 50 50 50 50 50
9538- D0 D0 D0 D0 D0 D0 D0 D0
9540- 20 24 28 2C 30 34 38 3C
9548- 20 24 28 2C 30 34 38 3C
9550- 21 25 29 2D 31 35 39 3D
9558- 21 25 29 2D 31 35 39 3D
9560- 22 26 2A 2E 32 36 3A 3E
9568- 22 26 2A 2E 32 36 3A 3E
9570- 23 27 2B 2F 33 37 3B 3F
9578- 23 27 2B 2F 33 37 3B 3F
9580- 20 24 28 2C 30 34 38 3C
9588- 20 24 28 2C 30 34 38 3C
9590- 21 25 29 2D 31 35 39 3D
9598- 21 25 29 2D 31 35 39 3D
95A0- 22 26 2A 2E 32 36 3A 3E
95A8- 22 26 2A 2E 32 36 3A 3E
95B0- 23 27 2B 2F 33 37 3B 3F
95B8- 23 27 2B 2F 33 37 3B 3F
95C0- 20 24 28 2C 30 34 38 3C
95C8- 20 24 28 2C 30 34 38 3C
95D0- 21 25 29 2D 31 35 39 3D
95D8- 21 25 29 2D 31 35 39 3D
95E0- 22 26 2A 2E 32 36 3A 3E
95E8- 22 26 2A 2E 32 36 3A 3E
95F0- 23 27 2B 2F 33 37 3B 3F
95F8- 23 27 2B 2F 33 37 3B 3F
```

```
KEY PERFECT 4.0
    RUN ON
    YTABLE

 CODE     ADDR# - ADDR#
 ----     -------------
 277A     8000 - 804F
 27AE     8050 - 809F
 28F0     80A0 - 80EF
 2461     80F0 - 813F
 27E7     8140 - 818F
 0630     8190 - 819B
TOTAL PROGRAM CHECK IS : 019C
```

LISTING 2:
DISASSEMBLY OF YADDR AND SETUP ROUTINES FROM YTABLE

```
9464-  A4 06     LDY   $06
9466-  B1 CE     LDA   ($CE),Y
9468-  85 26     STA   $26
946A-  B1 EE     LDA   ($EE),Y
946C-  85 27     STA   $27
946E-  60        RTS
946F-  A9 80     LDA   #$80
9471-  85 CE     STA   $CE
9473-  A9 94     LDA   #$94
9475-  85 CF     STA   $CF
9477-  A9 40     LDA   #$40
9479-  85 EE     STA   $EE
947B-  A9 95     LDA   #$95
947D-  85 EF     STA   $EF
947F-  60        RTS
```

```
CHECK CODE 3.0

ON: YTABLE
TYPE: B

LENGTH: 019C
CHECKSUM: EE
```

1. The YADDR routine which is used to retrieve Hi-Res screen addresses from a table and place them in memory addresses $26 and $27 (HBASL and HBASH).
2. The SETUP routine which must be CALLed to initialize the YTABLE pointers before any machine code drawing routines can be used.
3. The actual table of screen addresses.

We will use YTABLE rather than the regular screen address location routines (HPOSN), as table look-up is much faster with these.

Listing 1 is a complete hex dump of all three parts of YTABLE, and **Listing 2** is a disassembly of the YADDR and SETUP parts of the routine. If you already have the YTABLE routines on disk from the Graphics Workshop series, you're in luck. Simply BLOAD them into memory, then enter the new hex bytes for YADDR and SETUP from **Listing 2**, and you're all finished. The reason that this YTABLE is so much shorter is that we don't need the page 2 high bytes, since Double Hi-Res doesn't use any addresses on HGR page 2.

For more information about entering machine code and binary files into memory, refer to the Welcome to New *Nibble* Readers section in this issue. You should save YTABLE to disk with the command **BSAVE YTABLE, A$9464,L$19C**.

CALL 37999 to Set the YTABLE Pointers

To use YADDR to find screen addresses, you will **POKE** the value of the current Y-coordinate (0-191) that you are dealing with into memory location 6, then **CALL 37988** (YADDR). YADDR will place the memory address of the leftmost two screen columns (columns 0 and 1) into memory locations $26 and $27 (decimal 38 and 39). Our machine code drawing routines will look to these locations to find which screen address they are to use. To retrieve that address in decimal form (from Applesoft), enter:

PRINT PEEK (38)+PEEK(39)*256

Remember that while the addresses for columns 0 and 1 will be the same, the physical location of the address for column 0 will be in auxiliary memory, and the physical location of the address for column 1 will be in your Apple's main memory.

LISTING 3: THE HOME, HGR, INIT, AND KILL ROUTINES

```
:ASM

                1000 ****************************************
                1010 *
                1020 * DH.DRIVER
                1030 *
                1040 * BY ROBERT A DEVINE
                1050 *
                1060 * COPYRIGHT 1984 BY MICROSPARC, INC.
                1070 *
                1080 * S-C MACRO ASSEMBLER
                1090 *
                1100 ****************************************
                1110    .OR $941C
                1120    .TA $800
941C- A9 04     1130 HOME LDA #$04      ** CALL 37916
941E- 85 3D     1140    STA $3D
9420- 85 43     1150    STA $43
9422- A9 07     1160    LDA #$07
9424- 85 3F     1170    STA $3F
9426- D0 0A     1180    BNE J1
9428- A9 20     1190 HGR LDA #$20       ** CALL 37928
942A- 85 3D     1200    STA $3D
942C- 85 43     1210    STA $43
942E- A9 3F     1220    LDA #$3F
9430- 85 3F     1230    STA $3F
9432- A9 00     1240 J1 LDA #$00
9434- 85 3C     1250    STA $3C
9436- 85 42     1260    STA $42
9438- A9 FF     1270    LDA #$FF
943A- 85 3E     1280    STA $3E
943C- 38        1290    SEC
943D- 20 11 C3  1300    JSR $C311       ** MOVE MEMORY TO AUX
9440- 60        1310    RTS
9441- 8D 5E C0  1320 INIT STA $C05E     ** ANNUNCIATOR 3 OFF   CALL 37953
9444- 8D 0D C0  1330    STA $C00D       ** 80 COLUMN     ON
9447- 8D 50 C0  1340    STA $C050       ** GRAPHICS      OFF
944A- 8D 57 C0  1350    STA $C057       ** HI-RES        ON
944D- 60        1360    RTS
944E- 8D 5F C0  1370 KILL STA $C05F     ** ANNUNCIATOR 3 ON    CALL 37966
9451- 8D 0C C0  1380    STA $C00C       ** 40 COLUMN     OFF
9454- 8D 51 C0  1390    STA $C051       ** TEXT          ON
9457- 8D 56 C0  1400    STA $C056       ** LOW RES       OFF
945A- 8D 00 C0  1410    STA $C000       ** 80 STORE      OFF
945D- 8D 54 C0  1420    STA $C054       ** PAGE 2        OFF
9460- 20 58 FC  1430    JSR $FC58       ** HOME CURSOR
9463- 60        1440    RTS

SYMBOL TABLE

9428- HGR
941C- HOME
9441- INIT
932- J1
944E- KILL

0000 ERRORS IN ASSEMBLY
```

```
KEY PERFECT 4.0
    RUN ON
    DHR.DRIVER

 CODE     ADDR# - ADDR#
 ----     -------------
 289C     8000 - 804F
 1CBC     8050 - 809F
 212E     80A0 - 80EF
 26E9     80F0 - 813F
 258E     8140 - 818F
 2944     8190 - 81DF
 021F     81E0 - 81E3
TOTAL PROGRAM CHECK IS : 01E4

CHECK CODE 3.0

ON: DHR.DRIVER
TYPE: B

LENGTH: 01E4
CHECKSUM: EE
```

POKE 6,Y: CALL 37988 to Get Screen Addresses

The address stored in $26 and $27, plus the screen address offset (0-39), will be the final location in memory where the actual drawing will be done.

Special Utility Routines

Before we run our first tests, you'll need to enter one more short set of routines which will make housekeeping much easier. These routines are shown in **Listing 3** and are called HOME, HGR, INIT, and KILL. Enter these routines beginning at address $941C. Then save them to disk, after BLOADing YTABLE, with the command:

```
BSAVE DHR.DRIVER,A$941C,L$1E4
```

Immediately after BLOADing your DHR driver into memory, you should always protect it from strings by setting **HIMEM**, and **CALL 37999** to set the YTABLE pointers.

Now that we have a few utilities to work with, let's see what these routines do.

The INIT Routine — CALL 37953

This routine should always be used to get your Apple into 560-dot Double Hi-Res mode. It is the sequence of steps that is referred to on pages 11 and 12 of your extended 80-column card supplement. You will note that we have turned annunciator 3 **off** rather than **on** (as the manual says), as this is the only way to make things work.

The KILL Routine — CALL 37966

This routine is used to turn off your 560-dot display and return to normal Apple text mode, with the 80-column card **off**. Here we reset all the switches that were set by the INIT routine. The routine also ensures that the 80STORE and PAGE2 soft switches are turned **off** when leaving 560-dot mode. This routine does not affect any graphics which you may have drawn on the Hi-Res screens, so you may re-enter Double Hi-Res with the INIT routine and find your graphics intact. The final function of KILL is to use the normal HOME routine to clear the text screen and send the cursor HOME.

You should be aware that if you turn on the 80-column card using soft switches, you can **not** turn it off using the **<ESC><CONTROL-Q>** sequence described in the manual. The only way I've found to get out of the 80-column card is to either enter PR#3: **<ESC><CONTROL-Q><RESET>**, or use the KILL routine. The only time **<ESC><CONTROL-Q>** seems to work is when you have turned on the card with PR#3.

The HOME and HGR Routines — CALL 37916 and CALL 37928

You should immediately understand that these routines are intended to **simulate** the normal HOME and HGR commands from Applesoft; i.e., they really don't do the same things as their counterpart Applesoft commands. Perhaps you would have selected different names for these routines — but since I'm writing this, I get to pick the names!!

When you enter either the HOME or HGR Applesoft command while in Double Hi-Res mode, the only part of the screen display affected is that part which is stored in main memory. This means that entering either of these commands while in 560 mode will only clear one-half of the text or Hi-Res screen (the odd columns).

Our driver HOME and HGR routines **duplicate** the state of main memory in auxiliary memory by doing a memory move. In other words, to clear **both** areas of Double Hi-Res memory, you would enter **HGR** which would clear Hi-Res main memory, then **CALL 37928** (HGR), which would duplicate the contents of main memory in auxiliary memory, effectively clearing both screens. The same process would be used to clear the text screen.

This method could just as easily be used to turn both screens to a solid background color or some other pattern you might choose. If, for instance, you wanted to turn the entire screen white, you could first set normal Hi-Res to white, and then **CALL 37928** (HGR), which would also change the Hi-Res screen in auxiliary memory to white.

Special Notes About the HOME and HGR Routines

The first thing you must be sure of before using HOME or HGR is that the 80STORE soft switch is turned **off**. If 80STORE is **on** when you use these routines, they will not work. 80STORE can be turned off with **POKE 49152,0**. It can then be turned on again after using HOME or HGR with **POKE 49153,0**.

The second thing you should know is that both routines can also be made to duplicate (move) auxiliary memory to main memory. By entering a single **POKE**, you can change the direction of the memory move.

POKE 37948,24 will cause auxiliary memory to move to main memory.

POKE 37948,56 will cause main memory to move to auxiliary memory.

These POKEs simply change the instruction in **line 1360** to either SEC (SEt Carry) or CLC (CLear Carry).

Testing the Double Hi-Res Screen

Now that we've prepared a few utilities which will help us out, let's see what the Double Hi-Res screen looks like. You'll now need to have the DHR driver in memory.

The first thing to do is **CALL 37953** (INIT). At this point, you are in 560-dot mode with full-screen graphics. You will note that the INIT routine does not clear the screen; this is so that your program can enter and exit 560-dot mode without affecting anything that you might have previously stored in Hi-Res memory.

Next, enter HGR and you will note that some of the garbage will disappear from the screen; however, it will probably still contain some vertical white bars which represent the data bytes in auxiliary memory. The HGR command will also change your mode to mixed text and graphics. You may still use the normal POKEs/PEEKs at locations 49234 and 49235 to change to full-screen or mixed text and graphics.

The bars are still on the screen because HGR only clears the Hi-Res area in main memory. To clear the rest of the screen, **CALL 37928** (HGR), which will clear the Hi-Res memory area on the 80-column card. There are other ways to clear memory on the 80-column card, but I have chosen this method of memory duplication because it may have many other useful applications in your programs.

Now enter **HOME** and then **CALL 37916** (HOME), which will also clear the text window, again by the memory duplication method. If you now hold down the RETURN or left arrow key, your cursor will soon appear in the text area. At this point, you're ready to begin drawing on the screen.

A Matter of Terminology

In the *Extended 80-Column Text Card Manual*, the area of main memory (for either text or graphics) is called page 1, and the area of auxiliary memory (for either text or graphics) is called page 1X. For the balance of our discussion, we will refer to these two areas of memory as text pages 1 or 1X, and graphics pages 1 or 1X, to be consistent with the manual.

Now that we're in 560-dot mode, let's see how we can exit and return to normal text mode. If you try to exit with **<ESC><CONTROL-Q>** as recommended in the manual, you'll find that **nothing happens**. The TEXT command will get you out of 560-dot mode; however, you will still have 80-column text.

If all else fails, you could always resort to **<CONTROL><RESET>**, which is a reasonably sure way to recover from anything.

But in this case, I'd suggest **CALL 37966** (KILL), which returns you to normal text mode and clears the text screen. This routine also resets the soft switches to their proper positions. If by chance your program has changed RAMRD, RAMWRT, or ALTZP switches (refer to your manual for a discussion of these switches), you'd better be sure to have your program reset them as well.

When you **CALL 37953** (INIT) again, you will return to 560-dot mode with the screen nice and clear, just like you left it.

Next, enter:

HCOLOR=3: HPLOT 13,0 TO 13,159

As strange as it seems, you should have gotten a **red** line down the screen, even though you set **HCOLOR to white**, and you only drew one vertical line.

Now enter:

HPLOT 10,0 TO 10,159

EGADS!!! What's going on here? Your Apple doesn't have a Hi-Res pink, but — sure enough — you should be looking at a vertical pink line. In fact, we still have HCOLOR set to white.

The 16-Color Connection

One of the exciting "freebies" that comes along with Double Hi-Res (and doesn't seem to be mentioned in the manual) is the fact that you can now generate 16 colors on the Double Hi-Res screen.

From now on, when using Applesoft with Double Hi-Res, the only HCOLOR values that you need use are **HCOLOR=3** (white, turn a dot on) and **HCOLOR=0** (black, turn a dot off).

Let's take a moment to clarify what this 560-dot resolution is all about. If you are drawing on the Double Hi-Res screen in white or black, you will have 560-dot horizontal resolution, just as advertised. If you are drawing shapes or the background in color, you will still have 140-dot resolution, exactly the same as with normal Hi-Res; however, you will now have your choice of 16 different colors.

In normal 280-dot Hi-Res, white is generated by having two adjacent dots turned **on**, and black is the result of having two adjacent dots turned **off**. In 560-dot mode, white or black are created by having four adjacent dots turned **either on or off**.

In 560-dot mode, the status of bit 7 is totally ignored, which simply means that whether it happens to be on or off is unimportant, as it has no effect on the color that is displayed.

Since the screen dots in 560-mode are only half as wide as those of normal Hi-Res, this means that four dots will fit in the same space as two regular Hi-Res dots. As it happens, there are 2*2*2*2=16 different ways that you can arrange those four adjacent dots; thus the choice of 16 different colors.

When we drew our first vertical line at 13, we changed the pattern of one of those groups of four dots, causing **red** to appear. We then drew our next vertical line at 10, and changed the pattern of that same group of four dots, which changed the color to **pink**.

Now that we know there is a choice of 16 different colors, let's run a short program to see just what these 16 colors look like.

Listing 4 is a short program that will display all 16 colors. You will notice that our special utilities have made it very easy for us to get into 560-dot mode, clear the screen, and run the display. We could have drawn the bars with HPLOT statements (which in this case would have actually been a bit faster); however, this method will help you get used to working with the driver routines. At the end of the demo, simply press any key and the KILL routine will return you to normal text mode with the 80-column card off.

We've begun by drawing our colors on page 1X, and at the end (**line 150**), we've duplicated our display on page 1 by setting the proper POKEs and using the HGR routine. Each of the vertical bars that were drawn on the screen are four dots wide.

Special Note: You'll notice that we used a GET A$ statement in the last line of our demo. When we executed this statement, the last byte of the YTABLE Address Table was damaged, so you should always be sure to set HIMEM to protect your driver routines whenever you use them.

Now let's look at the Hi-Res colors (**Figure 2**) and find out what bit patterns make what colors. You should bear in mind that all of the bit patterns shown in **Figure 2** are really reversed from their normal direction. The bit patterns placed on the screen are always placed in reverse order. For example, a byte with the value 11 ($B) which has a bit pattern of 00001011 would appear on the screen as 1101000, with bit 7 ignored.

FIGURE 2: THE HI-RES COLORS

Nibble Value	Bit Pattern	Color Description
0	0 0 0 0	Black
1	1 0 0 0	Dark blue
2	0 1 0 0	Medium blue
3	1 1 0 0	Medium light blue
4	0 0 1 0	Dark green
5	1 0 1 0	Gray
6	0 1 1 0	Light green
7	1 1 1 0	Aqua
8	0 0 0 1	Red
9	1 0 0 1	Pink
10	0 1 0 1	Gray
11	1 1 0 1	Light blue
12	0 0 1 1	Orange
13	1 0 1 1	Flesh
14	0 1 1 1	Yellow
15	1 1 1 1	White

Now that we know what the colors look like, let's try out another test. Reload the driver, clear the Hi-Res screens and text window, then try this:

HCOLOR=3
FOR Y=0 TO 180 STEP 20: HPLOT 0,0 TO 279,Y: NEXT

This will draw 10 lines on the screen, with each succeeding line at a greater angle. You will note that the lines on the screen have holes in them (this is because you're only drawing on page 1), and that the lines become more and more jagged as they become less and less horizontal. This is one of the drawbacks to using HPLOT statements in Double Hi-Res and is the result of the dots being twice as high as they are wide.

Now enter **POKE 49153,0** which turns on the 80STORE soft switch, allowing you to select the page on which you want to draw.

Then enter **POKE 49237,0**, which sets the page 2 switch to select page 1X. Again enter:

FOR Y=0 TO 180 STEP 20: HPLOT 0,0 TO 279,Y: NEXT

This time you have drawn the exact same pattern of lines in the even columns of page 1X. You did not, however, do anything to improve the appearance of the lines or fill in any of the spaces, except on the one horizontal line. If anything, the quality of the lines has gotten worse.

Take a bit of time now to try out your own experiments and when you're ready, we'll look a little deeper into a method of drawing on this new graphics screen.

If you've taken some time trying to HPLOT lines, shapes, or whatever on the screen, you've probably found that things don't always work out quite the way you expected. So let's take another more closely-detailed look at how the screen is laid out.

In this part of our discussion, we will be looking at the screen information shown in **Figure 3**.

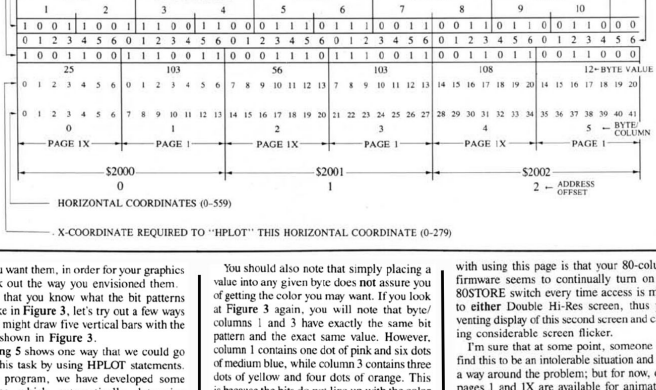
This figure represents only three leftmost addresses on the screen, $2000-$2002, which are located at the top of the screen at Y-coordinate 0. To help you get a clearer picture of how the colors work on the screen, these bytes will contain five blocks of color (pink, medium blue, yellow, orange, and light green), with each color area eight dots wide.

If you tried when you conducted your own tests to enter some HPLOT statements with X-coordinates greater than 279, you quickly found that your Apple quite firmly rejects all such attempts with the familiar ILLEGAL QUANTITY ERROR. So how do you HPLOT all 560 Double Hi-Res dots?

Just as you've already discovered that there are duplicate memory addresses where screen data is stored, you now need to understand that there are also duplicate HPLOT coordinates on the screen. When drawing on page 1X, an **HPLOT 0,0** will plot a point at the very left-most dot on the screen; however, when on page 1, an **HPLOT 0,0** will plot a point at position 7 in the 0-559 range.

So even though **Figure 3** shows the status of the leftmost 42 screen dots, if you tried an **HPLOT 40,Y** (regardless of whether you're on page 1 or page 1X), the resulting point would be somewhere off in never-never land — nowhere near the area of the screen that our figure describes. It should be quite obvious from looking at the information in **Figure 3** that you must be a lot more careful when using the Double Hi-Res screen. You'll need to do more advance planning when it comes to shape animation to be sure that your colors stay the

FIGURE 3



way you want them, in order for your graphics to work out the way you envisioned them.

Now that you know what the bit patterns look like in **Figure 3**, let's try out a few ways that we might draw five vertical bars with the colors shown in **Figure 3**.

Listing 5 shows one way that we could go about this task by using HPLOT statements. In this program, we have developed some formulas which automatically determine whether we want to draw on page 1 or page 1X, based on the value of the X-coordinate (0-559) that we have selected.

Line 600 determines which column (0-79) we're in and flips the proper soft switch. **Lines 610** and **620** take the X-coordinate that we've selected and translate it into the proper HPLOT X-coordinate that we really want (0-279).

You will probably want to use **lines 600-620** as a subroutine in your programs that use HPLOTs. You should note, however, that this will only work with vertical lines. Horizontal or diagonal HPLOT lines will tend to cross over from page to page, and will require their own type of special handling.

Finally, **Listing 6** shows how we can create the same five-color bars by directly POKEing the proper bit patterns into memory on pages 1 and 1X. You should remember that the value you POKE will have a bit pattern that is the reverse of the bit pattern you want to see on the screen.

Let's take the bit pattern from page 1X, address 0, as an example. You will note that the pattern appearing in this byte is 1001100. The value that we want to store here will have a bit pattern of (high bit 0 or 1) + 0011001, which in this case is 25 with the high bit set to 0.

You should also note that simply placing a value into any given byte does **not** assure you of getting the color you may want. If you look at **Figure 3** again, you will note that byte/columns 1 and 3 have exactly the same bit pattern and the exact same value. However, column 1 contains one dot of pink and six dots of medium blue, while column 3 contains three dots of yellow and four dots of orange. This is because the bits do not line up with the color blocks which are shown above each byte.

If you look closely at **Figure 3**, you will notice that the left edge of the color block and the left edge of the byte line up evenly on column 0, and do not line up evenly again until you reach the left edge of column 4. Therefore, if you wanted to fill the screen, or part of the screen, with some solid color, you would need to deal with four different values (bit patterns) which would repeat every four columns.

Another Double Hi-Res Page

So far we've only talked about Double Hi-Res pages 1 and 1X; however, there is another 560-dot graphics page available to you. This second Double Hi-Res page (which is not mentioned in the manual) is analogous to HGR2 and resides in memory addresses $4000 through $5FFF. These Hi-Res pages are called pages 2 and 2X, and they are designed exactly the same way as pages 1 and 1X, the only difference being in the addresses used.

To access this second Double Hi-Res screen, you must turn off the 80STORE switch with **POKE 49152,0** and turn on the page 2 switch with **POKE 49237,0**.

This second 560-dot screen can be very useful for nonanimated title pages or other information; however, it does not appear to be useful for animated graphics. The problem with using this page is that your 80-column firmware seems to continually turn on the 80STORE switch every time access is made to **either** Double Hi-Res screen, thus preventing display of this second screen and causing considerable screen flicker.

I'm sure that at some point, someone will find this to be an intolerable situation and find a way around the problem; but for now, only pages 1 and 1X are available for animation.

Loading Title Pages onto Pages 2 and 2X

If you want to build display graphics for pages 2 and 2X, you'll need to create two separate disk files: one to contain the data for the even columns, and the second to contain the screen data for the odd columns. Your best bet will be to create your graphics on pages 1/1X so that you can see your work.

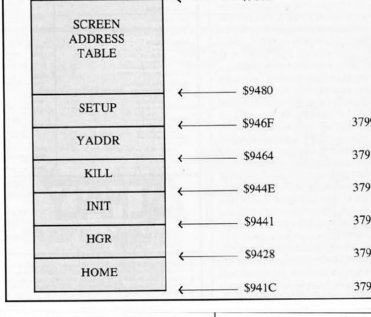
Once your graphics are completed, follow these steps to save the title page to disk:

1. **BSAVE TITLE.ODD,A$2000,L$2000**, which will save the odd columns.
2. Enter a line similar to **line 150** (**Listing 4**) which will move the screen data from auxiliary to main memory.
3. **BSAVE TITLE.EVEN,A$2000,L$2000**, which will save the even columns.

Now, to reload the title onto pages 2/2X for display, follow these steps:

1. **BLOAD TITLE.EVEN,A$4000**, which will load the even columns.
2. Enter **POKE 61,64: POKE 63,95: CALL 37938**. This uses part of our HGR routine (we enter at **line 1310**) to move the contents of page 2 onto page 2X.

FIGURE 4: MAP OF THE DHR DRIVER



3. BLOAD TITLE.ODD,A$4000, which will load the odd columns of your display.

4. Make sure to **POKE 49152,0**: **POKE 49237**, to display the graphics.

Since your graphics page will total 16384 bytes, it would be helpful to use some sort of fast DOS patch so that it doesn't take too long to get your graphics on the screen.

Turning the 80-Column Card On and Off

I think it might be worthwhile to spend a moment looking again at how we enter and exit the 80-column card. In our demonstrations, we have used the INIT and KILL routines to enter and exit Double Hi-Res, and we used the Applesoft HOME and our own HOME routine to clear the text area. As it happens, we have not yet actually turned the card on. The text that we generated had 80-column-sized letters, but we were only able to print in the odd columns. This meant that we could only place 40 characters on the screen.

If you're not going to need 80 columns of text, what we've done so far is enough; however, if you need true 80-column text, you'll need to actually turn the card on. This can be tricky, and if done incorrectly can result in the loss of the DOS hooks, and even your Applesoft program pointers. I learned both of these lessons through experience.

Proper Syntax for Double Hi-Res With 80-Column Text

- To enter Double Hi-Res:

  CALL 37953 {The INIT routine}
  PRINT CHR$(4)"PR#3" {This can be used in place of **HOME**: CALL **37916** (HOME)}

- To exit:

  PRINT CHR$(12);CHR$(21) {This acts like **<ESC><CONTROL-Q>** and retains DOS}
  CALL **37966** {The KILL routine}

Mixing up the order in which you enter or exit can cause strange results. BEWARE!!!!

Conclusion

When I sat down to write this series, I was planning to get into some of the Double Hi-Res block routines, with some animation ideas, in Part 1. As it turned out, the process of laying the groundwork for Double Hi-Res took longer than I had expected, so we'll begin working with animation techniques next month. See you then!!!!

The Double Hi-Res Manual

The first thing you'll need to know about the *Extended 80-Column Text Card Supplement* (mine has the number 030-0496-A on the back) is that this manual **has lots of bugs!!!** So far I've found six separate errors/typos in the manual. If you try to follow the instructions in the manual, you won't be able to make the new Double Hi-Res work!!

At this point, I'd suggest that you turn to pages 11 and 12 in the manual and change the instructions "Turn on the annunciator 3 soft switch.." to "Turn **off** the annunciator 3 soft switch..."

Annunciator 3 is turned **off** with either **POKE 49246,0** or **STA $C05E.**

Suggested Reading

Due to the way that Double Hi-Res is designed, the most efficient way to work with it is by using machine language routines to store specific data bytes on the screen. In other words, **block shapes** seem like the best way to go. (This does not mean that you can't use the HPLOT statement from Applesoft; we will also use this statement in many of our tests.) The driver that we'll develop, as well as most of our experiments, will deal with block shapes.

For a complete discussion of normal 280-dot block shape usage, see "Graphics Workshop: Block Shapes Part 1" (*Nibble* Vol. 4/No. 3). I think that once you've gotten the block shape concepts down firmly, it will make your transition to **Double Hi-Res block shapes** a lot easier. Double Hi-Res block shapes will be a bit more complex and require a greater understanding of the physical design of the Hi-Res screens.

How 560-Dot Double Hi-Res Is Designed

When working with Double Hi-Res, you will **always** be working with Hi-Res graphics page 1, which begins at memory address $2000 (8192 decimal) and ends at memory address $3FFF (16383 decimal). This area of memory (as described on p. 34 of your *Apple //e Reference Manual*) is 192 bytes high and 40 bytes wide, for a total of 8192 bytes. Each of the 40 horizontal bytes is capable of displaying 7 dots on the screen; 40 bytes wide x 7 dots = 280 dots horizontally.

So, you ask, how do 280 dots make 560 dots? What they've done is to use the 8192 bytes from normal Hi-Res page 1, plus another 8192 bytes from the extended 80-column card to make up the 16384 bytes needed for the 560-dot display.

You should understand at this point that the 64K of auxiliary RAM on the extended 80-column card **is not additional RAM**; rather, it is **duplicate RAM**, with exactly the same range of memory addresses as you already had when you bought your Apple. This means that there is also a block of memory addresses on the extended 80-column card

LISTING 4: COLOR.DEMO

```
1  REM  ************************
2  REM  *    COLOR.DEMO        *
3  REM  * BY ROBERT R. DEVINE  *
4  REM  * COPYRIGHT (C) 1984   *
5  REM  * BY MICROSPARC, INC.  *
6  REM  * LINCOLN, MA  01773   *
7  REM  ************************
10 PRINT CHR$ (4)"BLOAD DHR.DRIVER"
15 CALL 37999: REM  SETUP YTABLE
20 CALL 37953: REM  INIT DOUBLE HI-RES
25 HGR : CALL 37928: REM  CLEAR DHR SCREENS
30 HOME : CALL 37916: REM  CLEAR TEXT WINDOW
40 YTAB 22: PRINT "0 1 2 3 4 5 6 7 8 9 1 1 1 1 1
   1 1"
50 PRINT "                    0 1 2 3 4 5"
60 PRINT "   DOUBLE HI-RES COLORS"
70 POKE 49153,0: POKE 49237,0: REM  SELECT PAG
   E 1X
100 FOR X = 0 TO 15
120 FOR Y = 0 TO 150: POKE 6,Y: CALL 37988: ADD
   RESS = PEEK (38) + PEEK (39) * 256
130 POKE ADDRESS + X * 2,X
140 NEXT Y,X
150 POKE 49236,0: POKE 49152,0: POKE 37948,24:
    CALL 37928: REM  DUPLICATE PAGE 1X ONTO P
   AGE 1
160 VTAB 10: GET A$: CALL 37966: REM  TURN OFF
    560 DOT GRAPHICS MODE
```

LISTING 5: BAR.DEMO1

```
1  REM  ************************
2  REM  *    BAR.DEMO1         *
3  REM  * BY ROBERT R. DEVINE  *
4  REM  * COPYRIGHT (C) 1984   *
5  REM  * BY MICROSPARC, INC.  *
6  REM  * LINCOLN, MA  01773   *
7  REM  ************************
10 PRINT CHR$ (4)"BLOAD DHR.DRIVER"
20 CALL 37953: REM  INIT DOUBLE HI-RES
25 HGR : CALL 37928: REM  CLEAR DHR SCREENS
30 HOME : CALL 37916: REM  CLEAR TEXT WINDOW
40 POKE 49153,0: REM  SET 80STORE FOR MAIN/AUX
   ILIARY MEMORY SWITCHING
50 POKE 49234,0: REM  SET FOR FULL SCREEN GRAP
   HICS
60 HCOLOR= 3
100 FOR X = 1 TO 22: READ XC0@RD: GOSUB 600: NEXT
   X
110 DATA  0,3,4,7,8,9,12,13,17,18,19,21,22,23,
   25,27,30,31,33,34,37,38
500 END
600 POKE 49236,0:COLUMN = INT (XC0@RD / 7): IF
    COLUMN / 2 = INT (COLUMN / 2) THEN  POKE
    49237,0
610 XC = INT (COLUMN / 2) + XC0@RD / 7 - COLUM
   N
620 XC = INT (XC - 7 + 5): HPLOT XC,0 TO XC,1
    50: RETURN
```

LISTING 6: BAR.DEMO2

```
1  REM  ************************
2  REM  *    BAR.DEMO2         *
3  REM  * BY ROBERT R. DEVINE  *
4  REM  * COPYRIGHT (C) 1984   *
5  REM  * BY MICROSPARC, INC.  *
6  REM  * LINCOLN, MA  01773   *
7  REM  ************************
10 PRINT CHR$ (4)"BLOAD DHR.DRIVER"
15 CALL 37999: REM  SETUP YTABLE
   PROTECT DRIVER
20 CALL 37953: REM  INIT DOUBLE HI-RES
25 HGR : CALL 37928: REM  CLEAR DHR SCREENS
30 HOME : CALL 37916: REM  CLEAR TEXT WINDOW
40 POKE 49153,0: REM  SET 80STORE FOR MAIN/AUX
   ILIARY MEMORY SWITCHING
50 POKE 49234,0: REM  SET FOR FULL SCREEN GRAP
   HICS
100 POKE 49237,0: REM  SELECT PAGE 1X
110 FOR ADDRESS = 0 TO 2: READ BYTE: GOSUB 500
    : NEXT ADDRESS
120 POKE 49236,0: REM  SELECT PAGE 1
130 FOR ADDRESS = 0 TO 2: READ BYTE: GOSUB 500
    : NEXT ADDRESS
140 DATA  25,56,108,103,103,12
200 END
500 FOR Y = 0 TO 150: POKE 6,Y: CALL 37988: POKE
    PEEK (38) + PEEK (39) * 256 + ADDRESS,BY
    TE: NEXT Y: RETURN
```

TABLE 1: SUMMARY OF DHR.DRIVER ROUTINES

Routine Name	Call Address	Hex Address	Routine Function
SETUP	37999	$946F	Set YTABLE pointers.
YADDR	37988	$9464	Store screen address in $26-$27.
KILL	37966	$944E	Exit Double Hi-Res mode.
INIT	37953	$9441	Enter Double Hi-Res mode
HGR	37928	$9428	Move graphics page to graphics page.
HOME	37916	$941C	Move text page to text page.

Special POKEs to use with the driver:

POKE 6,Y Place Y-coordinate which you want an address for into location 6 before using the YADDR routine.

POKE 37948,24 Set HGR and HOME routines to move memory from page 1X to page 1.

POKE 37948,56 Set HGR and HOME routines to move memory from page 1 to page 1X.

Double Hi-Res Graphics I (Cont.)

The Double Hi-Res Manual

The first thing you'll need to know about the *Extended 80-Column Text Card Supplement* (mine has the number 030-0496-A on the back) is that this manual **has lots of bugs!!!** So far I've found six separate errors/typos in the manual. If you try to follow the instructions in the manual, you won't be able to make the new Double Hi-Res work!!

At this point, I'd suggest that you turn to pages 11 and 12 in the manual and change the instructions "Turn **on** the annunciator 3 soft switch..." to "Turn **off** the annunciator 3 soft switch..."

Annunciator 3 is turned **off** with either **POKE 49246,0** or **STA $C05E.**

Suggested Reading

Due to the way that Double Hi-Res is designed, the most efficient way to work with it is by using machine language routines to store specific data bytes on the screen. In other words, **block shapes** seem like the best way to go. (This does not mean that you can't use the HPLOT statement from Applesoft; we will also use this statement in many of our tests.) The driver that we'll develop, as well as most of our experiments, will deal with block shapes.

For a complete discussion of normal 280-dot block shape usage, see "Graphics Workshop: Block Shapes Part 1" (*Nibble* Vol. 4/No. 3). I think that once you've gotten the block shape concepts down firmly, it will make your transition to **Double Hi-Res block shapes** a lot easier. Double Hi-Res block shapes will be a bit more complex and require a greater understanding of the physical design of the Hi-Res screens.

How 560-Dot Double Hi-Res Is Designed

When working with Double Hi-Res, you will **always** be working with Hi-Res graphics page 1, which begins at memory address $2000 (8192 decimal) and ends at memory address $3FFF (16383 decimal). This area of memory (as described on p. 34 of your *Apple //e Reference Manual*) is 192 bytes high and 40 bytes wide, for a total of 8192 bytes. Each of the 40 horizontal bytes is capable of displaying 7 dots on the screen; 40 bytes wide x 7 dots = 280 dots horizontally.

So, you ask, how do 280 dots make 560 dots? What they've done is to use the 8192 bytes from normal Hi-Res page 1, plus another 8192 bytes from the extended 80-column card to make up the 16384 bytes needed for the 560-dot display.

You should understand at this point that the 64K of auxiliary RAM on the extended 80-column card **is not additional RAM**; rather, it is **duplicate RAM**, with exactly the same range of memory addresses as you already had when you bought your Apple. This means that there is also a block of memory addresses on the extended 80-column card