Hardware_Manual

COLLABORATORS			
	TITLE : Hardware_Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 18, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Hardware_Manual

1.1 Amiga® Hardware Reference Manual: 4 Sprite Hardware

This chapter discusses sprites which are special graphic objects that are easy to define and easy to animate. The following sprite topics are covered:

* Defining the size, shape, color, and screen position of sprites.
* Displaying and moving sprites.
* Combining sprites for more complex images, additional width, or additional colors.
* Reusing a sprite DMA channel multiple times within a display field to create more than eight sprites on the screen at one time.

1.2 4 Sprite Hardware / What are Sprites?

Sprites are graphic objects that are created and moved independently of the playfield display and independently of each other. Together with playfields, sprites form the graphics display of the Amiga. You can create more complex animation effects by using the blitter, which is described in the chapter called "Blitter Hardware." Sprites are produced on-screen by eight special-purpose sprite DMA channels . Basic sprites are 16 pixels wide and any number of lines high. You can choose from three colors for a sprite's pixels, and a pixel may also be transparent, showing any object behind the sprite. For larger or more complex objects, or for more color choices, you can combine sprites .

Sprite DMA channels can be reused several times within the same display

field. Thus, you are not limited to having only eight sprites on the screen at the same time.

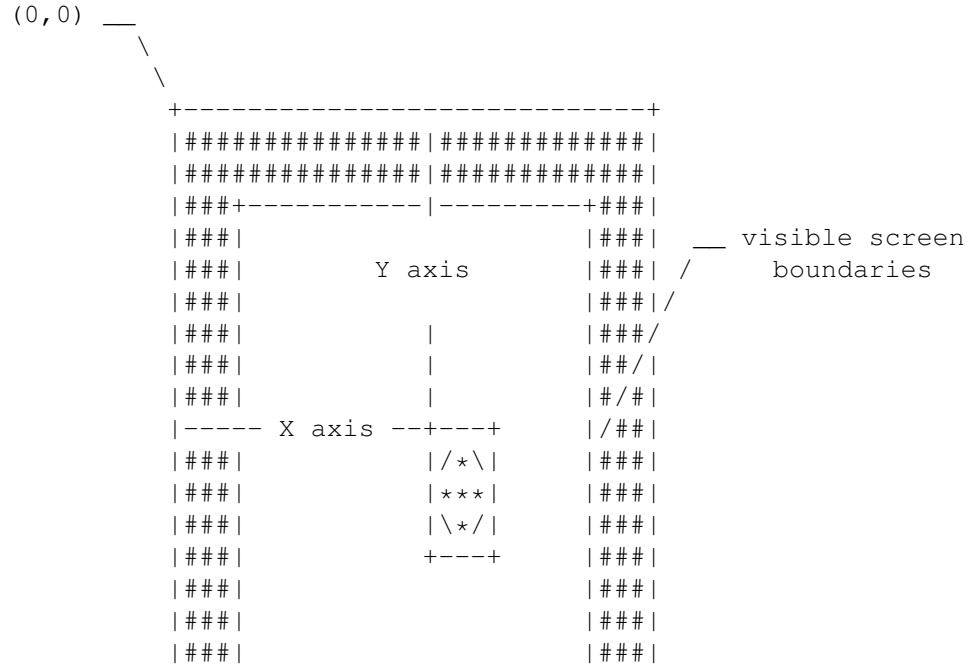
1.3 4 Sprite Hardware / Forming a Sprite

To form a sprite, you must first define it and then create a formal data structure in memory. You define a sprite by specifying its characteristics:

* On-screen width of up to 16 pixels.
* Unlimited height.
* Any shape.
* A combination of three colors, plus transparent.
* Any position on the screen.

1.4 4 / Forming a Sprite / Screen Position

A sprite's screen position is defined as a set of X,Y coordinates. Position (0,0), where X = 0 and Y = 0, is the upper left-hand corner of the display. You define a sprite's location by specifying the coordinates of its upper left-hand pixel. Sprite position is always defined as though the display modes were low resolution and non-interlaced. The X,Y coordinate system and definition of a sprite's position are graphically represented in Figure 4-1. Notice that because of display overscan, position (0,0) (that is, X = 0, Y = 0) is not normally in a viewable region of the screen.



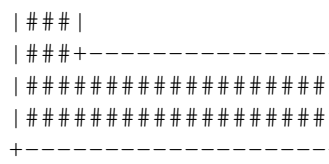


Figure 4-1: Defining Sprite On-screen Position

The amount of viewable area is also affected by the size of the playfield display window (defined by the values in DDFSTRT , DDFSTOP , DIWSTRT , DIWSTOP , etc.). See the "Playfield Hardware" chapter for more information about overscan and display windows .

Horizontal Position
Vertical Position
Clipped Sprites

1.5 4 / / Screen Position / Horizontal Position

A sprite's horizontal position (X value) can be at any pixel on the screen from 0 to 447. To be visible, however, an object must be within the boundaries of the playfield display window . In the examples in this chapter, a window with horizontal positions from pixel 64 to pixel 383 is used (that is, each line is 320 pixels long). Larger or smaller windows can be defined as required, but it is recommended that you read the "Playfield Hardware chapter" before attempting to do so. A larger area is actually scanned by the video beam but is not usually visible on the screen.

If you specify an X value for a sprite that takes it outside the display window, then part or all of the sprite may not appear on the screen. This is sometimes desirable; such a sprite is said to be " clipped ."

To make a sprite appear in its correct on-screen horizontal position in the display window, simply add its left offset to the desired X value. In the example given above, this would involve adding 64 to the X value. For example, to make the upper leftmost pixel of a sprite appear at a position 94 pixels from the left edge of the screen, you would perform this calculation:

Desired X position + horizontal-offset of display window ~= 94 + 64 = 158

Thus, 158 becomes the X value, which will be written into the data structure .

Counting Pixels.

The X position represents the location of the very first (leftmost) pixel in the full 16-bit wide sprite. This is always the case, even if the leftmost pixels are specified as transparent and do not appear on the screen.

If the sprite shown in Figure 4-2 were located at an X value of 158, the

actual image would begin on-screen four pixels later at 162. The first four pixels in this sprite are transparent and allow the background to show through.

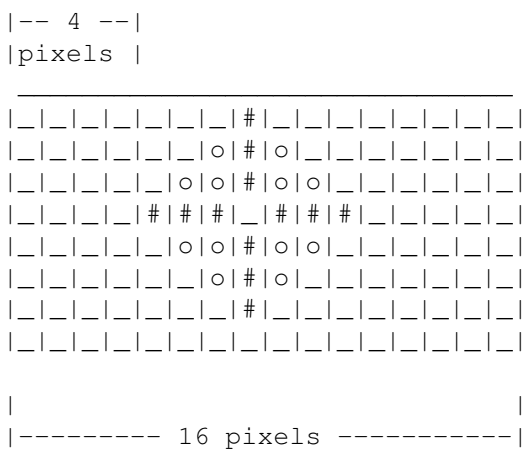


Figure 4-2: Position of Sprites

1.6 4 / / Screen Position / Vertical Position

You can select any position from line 0 to line 262 for the topmost edge of the sprite. In the examples in this chapter, an NTSC window with vertical positions from line 44 to line 243 is used. This allows the normal display height of 200 lines in non-interlaced mode. If you specify a vertical position (Y value) of less than 44 (i.e., above the top of the display window) the top edge of the sprite may not appear on screen.

To make a sprite appear in its correct on-screen vertical position, add the Y value to the desired position. Using the above numbers, add 44 to the desired Y position. For example, to make the upper leftmost pixel appear 25 lines below the top edge of the screen, perform this calculation:

Desired Y position + vertical-offset of the display window = 25 + 44 = 69

Thus, 69 is the Y value you will write into the data structure .

1.7 4 / / Screen Position / Clipped Sprites

As noted above, sprites will be partially or totally clipped if they pass across or beyond the boundaries of the display window. The values of 64 (horizontal) and 44 (vertical) are "normal" for a centered display on a standard NTSC video monitor. See Chapter 3, "Playfield Hardware," for more information on display offsets . Information on PAL displays will be found there. If you choose other values to establish your display window, your sprites will be clipped accordingly.

1.8 4 / Forming a Sprite / Size of Sprites

Sprites are 16 pixels wide and can be almost any height you wish -- as short as one line or taller than the screen. You would probably move a very tall sprite vertically to display a portion of it at a time.

Sprite size is based on a pixel that is 1/320th of a screen's width, 1/200th of a NTSC screen's height, or 1/256 of a PAL screen's height. This pixel size corresponds to the low resolution and non-interlaced modes of the normal full-size playfield. Sprites, however, are independent of playfield modes of display, so changing the resolution or interlace mode of the playfield has no effect on the size or resolution of a sprite.

1.9 4 / Forming a Sprite / Shape of Sprites

A sprite can have any shape that will fit within the 16-pixel width. You define a sprite's shape by specifying which pixels actually appear in each of the sprite's locations. For example, Figures 4-3 and 4-4 show a spaceship whose shape is marked by Xs. The first figure shows only the spaceship as you might sketch it out on graph paper. The second figure shows the spaceship within the 16-pixel width. The 0s around the spaceship mark the part of the sprite not covered by the spaceship and transparent when displayed.

```
  XX
 XXXXXX
XXXXXXXXXX
XXXXXXXXXX
 XXXXXX
  XX
```

Figure 4-3: Shape of Spaceship

```
0000XX0000000000
00XXXXXX00000000
XXXXXXXXXX000000
XXXXXXXXXX000000
00XXXXXX00000000
0000XX0000000000
```

Figure 4-4: Sprite with Spaceship Shape Defined

In this example, the widest part of the shape is ten pixels and the shape is shifted to the left of the sprite. Whenever the shape is narrower than the sprite, you can control which part of the sprite is used to define the shape. This particular shape could also start at any of the pixels from 2-7 instead of pixel 1.

1.10 4 / Forming a Sprite / Sprite Color

When sprites are used individually (that is, not attached as described in the Attached Sprites section), each pixel can be one of three colors or transparent. Color selection in similar to the method used for playfield colors. Figure 4-5 shows how the color of each pixel in a sprite is determined.

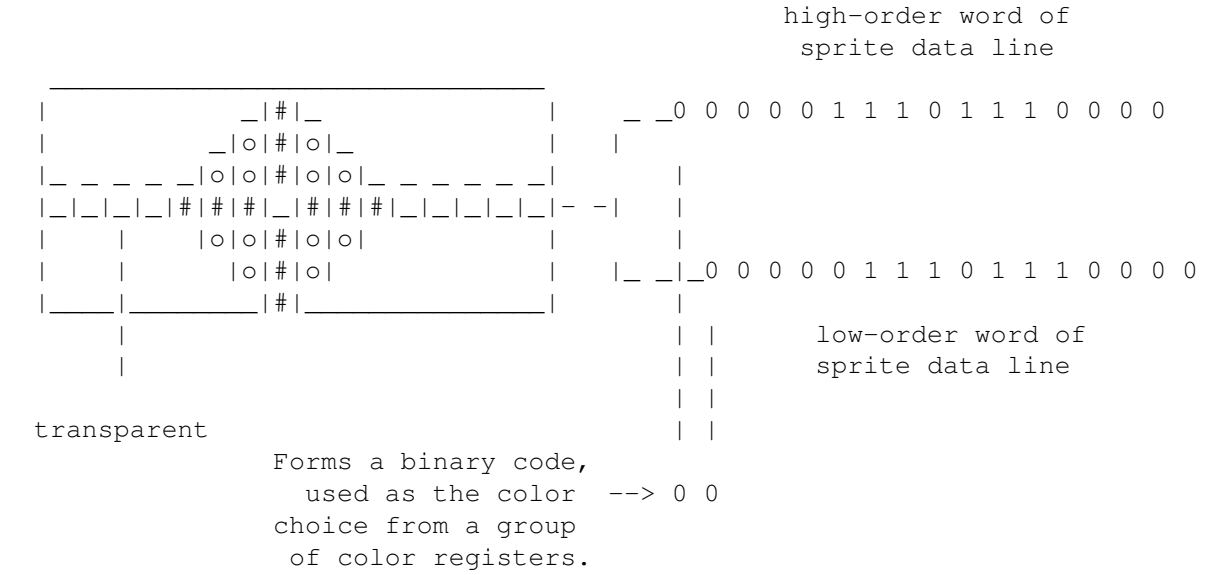


Figure 4-5: Sprite Color Definition

The 0s and 1s in the two data words that define each line of a sprite in the data structure form a binary number. This binary number points to one of the four color registers assigned to that particular sprite DMA channel. The eight sprites use system color registers 16 - 31. For purposes of color selection, the eight sprites are organized into pairs and each pair uses four of the color registers as shown in Figure 4-6.

About sprite color registers .

The color value of the first register in each group of four registers is ignored by sprites. When the sprite bits select this register, the "transparent" value is used.

```
00  Unused
01  Unused
      .
      .
      .
16  Unused   00  |
17  Color 1  01  |
18  Color 2  10  |-- Sprites 0 and 1
19  Color 3  11 _|
20  Unused   00  |
21  Color 1  01  |
22  Color 2  10  |-- Sprites 2 and 3
23  Color 3  11 _|
```

```
24  Unused   00  |
25  Color 1  01  |
26  Color 2  10  |-- Sprites 4 and 5
27  Color 3  11 _|
28  Unused   00  |
29  Color 1  01  |
30  Color 2  10  |-- Sprites 6 and 7
31  Color 3  11 _|
```

Figure 4-6: Color Register Assignments

If you require certain colors in a sprite, you will want to load the sprite's color registers with those colors. The "Playfield Hardware" chapter contains instructions on loading color registers .

The binary number 00 is special in this color scheme. A pixel whose value is 00 becomes transparent and shows the color of any other sprite or playfield that has lower video priority. An object with low priority appears "behind" an object with higher priority. Each sprite has a fixed video priority with respect to all the other sprites. You can vary the priority between sprites and playfields. (See Chapter 7, "System Control Hardware," for more information about sprite priority .)

1.11 4 / Forming a Sprite / Designing a Sprite

For design purposes, it is convenient to lay out the sprite on paper first. You can show the desired colors as numbers from 0 to 3. For example, the spaceship shown above might look like this:

```
0000122332210000
0001223333221000
0012223333222100
0001223333221000
0000122332210000
```

The next step is to convert the numbers 0-3 into binary numbers, which will be used to build the color descriptor words of the sprite data structure . The next section shows how to do this.

1.12 4 / Forming a Sprite / Building the Data Structure

After defining the sprite, you need to build its data structure, which is a series of 16-bit words in a contiguous memory area. Some of the words contain position and control information and some contain color descriptions. To create a sprite's data structure, you need to:

* Write the horizontal and vertical position of the sprite into the first control word.

* Write the vertical stopping position into the second control word.

* Translate the decimal color numbers 0 - 3 in your sprite grid picture into binary color numbers. Use the binary values to build color descriptor (data) words and write these words into the data structure.

* Write the control words that indicate the end of the sprite data structure.

Warning:

Sprite data, like all other data accessed by the custom chips, must be loaded into Chip RAM. Be sure all of your sprite data structures are word aligned in Chip Memory.

Table 4-1 shows a sprite data structure with the memory location and function of each word:

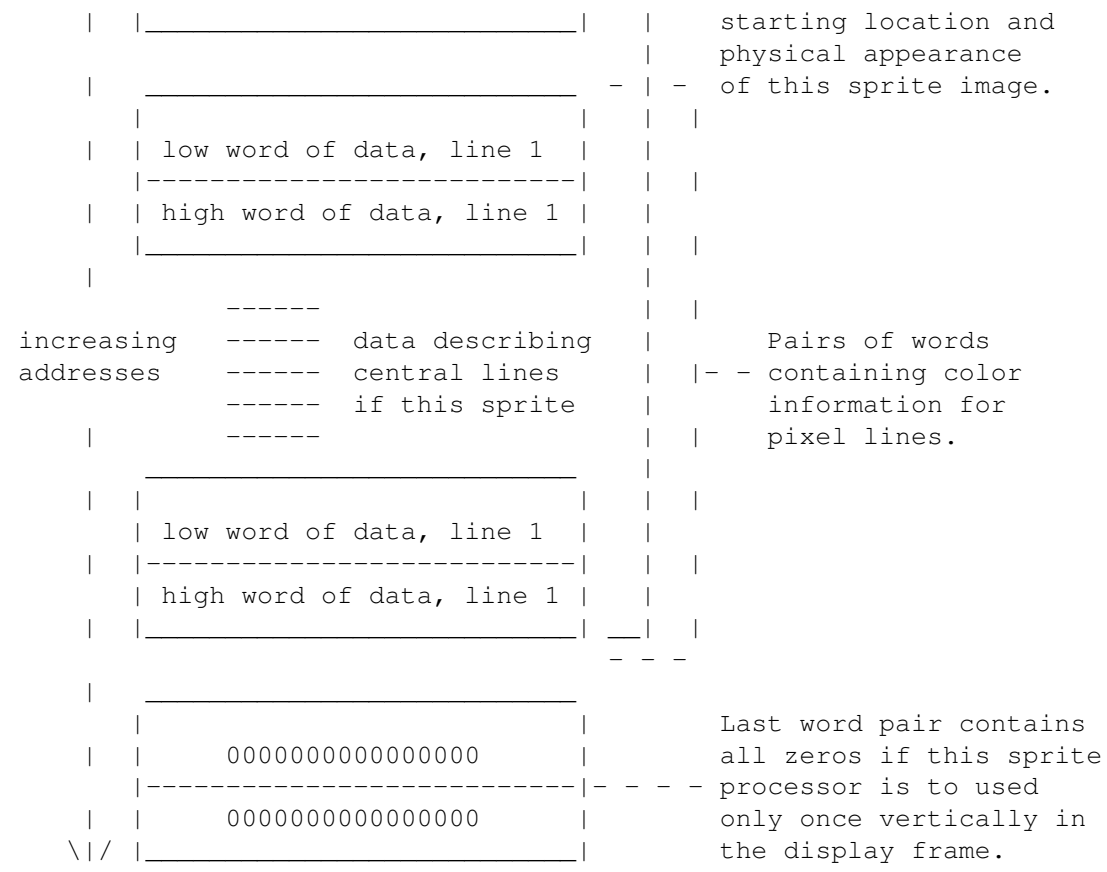
Memory Location	16-bit Word	Function
N	Sprite control word 1	Vertical and horizontal start position
N+1	Sprite control word 2	Vertical stop position
N+2	Color descriptor low word	Color bits for line 1
N+3	Color descriptor high word	Color bits for line 1
N+4	Color descriptor low word	Color bits for line 2
N+5	Color descriptor high word	Color bits for line 2
	-	
	-	
	-	
	End-of-data words	Two words indicating the next usage of this sprite

Table 4-1: Sprite Data Structure

All memory addresses for sprites are word addresses. You will need enough contiguous memory to provide room for two words for the control information, two words for each horizontal line in the sprite, and two end-of-data words.

Because this data structure must be accessible by the special-purpose chips, you must ensure that this data is located within chip memory.

Figure 4-7 shows how the data structure relates to the sprite.



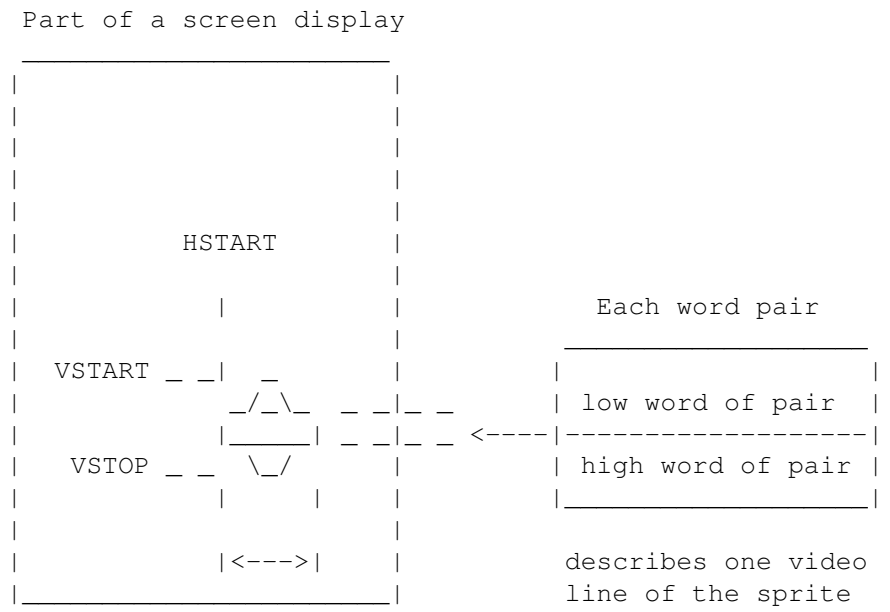


Figure 4-7: Data Structure Layout

Sprite Control Word 1 : SPRxPOS Sprite Color Descriptor Words
Sprite Control Word 2 : SPRxCTL End-of-data Words

1.13 4 / / Building the Data Structure / Sprite Control Word 1 : SPRxPOS

This word (SPR?POS) contains the vertical (VSTART) and horizontal (HSTART) starting position for the sprite. This is where the topmost line of the sprite will be positioned.

Bits 15-8 contain the low 8 bits of VSTART
Bits 7-0 contain the high 8 bits of HSTART

1.14 4 / / Building the Data Structure / Sprite Control Word 2 : SPRxCTL

This word contains the vertical stopping position of the sprite on the screen (i.e., the line AFTER the last displayed row of the sprite). It also contains some data having to do with sprite attachment , which is described later on.

SPRxCTL

Bits 15-8	The low eight bits of VSTOP
Bit 7	(Used in attachment)
Bits 6-3	Unused (make zero)
Bit 2	The VSTART high bit
Bit 1	The VSTOP high bit
Bit 0	The HSTART low bit

The value (VSTOP - VSTART) defines how many scan lines high the sprite will be when it is displayed.

1.15 4 / / Building the Data Structure / Sprite Color Descriptor Words

It takes two color descriptor (data) words to describe each horizontal line of a sprite; the high order word and the low order word. To calculate how many color descriptor words you need, multiply the height of the sprite in lines by 2. The bits in the high order color descriptor word contribute the leftmost digit of the binary color selector number for each pixel; the low order word contributes the rightmost digit.

To form the color descriptor words, you first need to form a picture of the sprite, showing the color of each pixel as a number from 0 - 3. Each number represents one of the colors in the sprite's color registers . For example, here is the spaceship sprite again:

0000122332210000
0001223333221000
0012223333222100
0001223333221000
0000122332210000

Next, you translate each of the numbers in this picture into a binary

number. The first line in binary is shown below. The binary numbers are represented vertically with the low digit in the top line and the high digit right below it. This is how the two color descriptor words for each sprite line are written in memory.

```
          0 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0    <- Low Sprite Word

          0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0    <- High Sprite Word
```

The first line above becomes the color descriptor low word for line 1 of the sprite. The second line becomes the color descriptor high word. In this fashion, you translate each line in the sprite into binary 0s and 1s. See Figure 4-7 .

Each of the binary numbers formed by the combination of the two data words for each line refers to a specific color register in that particular sprite channel's segment of the color table. Sprite channel 0, for example, takes its colors from registers 17 - 19. The binary numbers corresponding to the color registers for sprite DMA channel 0 are shown in Table 4-2.

Binary Number	Color Register Number
00	Transparent
01	17
10	18
11	19

Table 4-2: Sprite Color Registers

Recall that binary 00 always means transparent and never refers to a color except background.

1.16 4 / / Building the Data Structure / End-of-data Words

When the vertical position of the beam counter is equal to the VSTOP value in the sprite control words , the next two words fetched from the sprite data structure are written into the sprite control registers instead of being sent to the color registers . These two words are interpreted by the hardware in the same manner as the original words that were first loaded into the control registers . If the VSTART value contained in these words is lower than the current beam position, this sprite will not be reused in this display field. For consistency, the value 0 should be used for both words when ending the usage of a sprite. Sprite reuse is discussed later.

The following data structure is for the spaceship sprite. It will be located at V = 65 and H = 128 on the normally visible part of the screen.

```
SPRITE:
        DC.W    $6D60,$7200     ;VSTART, HSTART, VSTOP
```

```
        DC.W    $0990,$07E0     ;First pair of descriptor words
        DC.W    $13C8,$0FF0
        DC.W    $23C4,$1FF8
        DC.W    $13C8,$0FF0
        DC.W    $0990,$07E0
        DC.W    $0000,$0000     ;End of sprite data
```

1.17 4 Sprite Hardware / Displaying a Sprite

After building the data structure , you need to tell the system to display it. This section describes the display of sprites in automatic mode . In this mode, once the sprite DMA channel begins to retrieve and display the data, the display continues until the VSTOP position is reached. Manual mode is described later on in this chapter.

The following steps are used in displaying the sprite:

1. Decide which of the eight sprite DMA channels to use (making certain that the chosen channel is available).

2. Set the sprite pointers to tell the system where to find the sprite data.

3. Turn on sprite direct memory access if it is not already on.

4. For each subsequent display field, during the vertical blanking interval, rewrite the sprite pointers.

About sprite DMA.

If sprite DMA is turned off while a sprite is being displayed (that is, after VSTART but before VSTOP), the system will continue to display the line of sprite data that was most recently fetched. This causes a vertical bar to appear on the screen. It is recommended that sprite DMA be turned off only during vertical blanking or during some portion of the display where you are sure that no sprite is being displayed.

1.18 4 / Displaying Sprite / Selecting DMA Channel and Setting Pointers

In deciding which DMA channel to use, you should take into consideration the colors assigned to the sprite and the sprites video priority .

The sprite DMA channel uses two pointers to read in sprite data and control words . During the vertical blanking interval before the first display of the sprite, you need to write the sprite's memory address into these pointers. The pointers for each sprite are called SPRxPTH and SPRxPTL, where "x" is the number of the sprite DMA channel. SPRxPTH

contains the high three bits of the memory address of the first word in the sprite and SPRxPTL contains the low sixteen bits. The least significant bit of SPRxPTL is ignored, as sprite data must be word aligned. Thus, only fifteen bits of SPRxPTL are used. As usual, you can write a long word into SPRxPTH.

In the following example the processor initializes the data pointers for sprite 0. Normally, this is done by the Copper. The sprite is at address $20000.

```
     MOVE.L  #$20000,SPR0PTH+CUSTOM  ;Write $20000 to sprite 0 pointer...
```

These pointers are dynamic; they are incremented by the sprite DMA channel to point first to the control words , then to the data words , and finally to the end-of-data words . After reading in the sprite control information and storing it in other registers, they proceed to read in the color descriptor words . The color descriptor words are stored in sprite data registers, which are used by the sprite DMA channel to display the data on screen. For more information about how the sprite DMA channels handle the display, see the Hardware Details section below.

1.19 4 / Displaying a Sprite / Resetting the Address Pointers

For one single display field, the system will automatically read the data structure and produce the sprite on-screen in the colors that are specified in the sprite's color registers . If you want the sprite to be displayed in subsequent display fields, you must rewrite the contents of the sprite pointers during each vertical blanking interval. This is necessary because during the display field, the pointers are incremented to point to the data which is being fetched as the screen display progresses.

The rewrite becomes part of the vertical blanking routine, which can be handled by instructions in the Copper lists.

1.20 4 Sprite Hardware / Moving a Sprite

A sprite generated in automatic mode can be moved by specifying a different position in the data structure . For each display field, the data is reread and the sprite redrawn. Therefore, if you change the position data before the sprite is redrawn, it will appear in a new position and will seem to be moving.

You must take care that you are not moving the sprite (that is, changing control word data) at the same time that the system is using that data to find out where to display the object. If you do so, the system might find the start position for one field and the stop position for the following field as it retrieves data for display. This would cause a "glitch" and would mess up the screen. Therefore, you should change the content of the control words only during a time when the system is not trying to read them. Usually, the vertical blanking period is a safe time, so moving the sprites becomes part of the vertical blanking tasks

and is handled by the Copper as shown in the example below.

As sprites move about on the screen, they can collide with each other or with either of the two playfields. You can use the hardware to detect these collisions and exploit this capability for special effects. In addition, you can use collision detection to keep a moving object within specified on-screen boundaries. Collision Detection is described in Chapter 7, "System Control Hardware."

sprite_move.asm

1.21 4 Sprite Hardware / Creating Additional Sprites

To use additional sprites, you must create a data structure for each one and arrange the display as shown in the previous section, naming the pointers SPR1PTH and SPR1PTL for sprite DMA channel 1, SPR2PTH and SPR2PTL for sprite DMA channel 2, and so on.

About sprite DMA.

When you enable sprite DMA for one sprite, you enable DMA for all the sprites and place them all in automatic mode . Thus, you do not need to repeat this step when using additional sprite DMA channels.

Once the sprite DMA channels are enabled, all eight sprite pointers must be initialized to either a real sprite or a safe null sprite. An uninitialized sprite could cause spurious sprite video to appear.

Remember that some sprites can become unusable when additional DMA cycles are allocated to displaying the screen, for example when an extra wide display or horizontal scrolling is enabled (see Figure 6-9: DMA Time Slot Allocation).

Also, recall that each pair of sprites takes its color from different color registers, as shown in Table 4-3.

Table 4-3: Color Registers for Sprite Pairs

Sprite Numbers	Color Registers
0 and 1	17 - 19
2 and 3	21 - 23
4 and 5	25 - 27
6 and 7	29 - 31

Warning:

Some sprites become unusable when additional DMA cycles are allocated to displaying the screen, e.g. when enabling an extra wide display or horizontal scrolling. (See Figure 6-9: DMA Time Slot Allocation .)

Sprite Priority

1.22 4 / Creating Additional Sprites / Sprite Priority

When you have more than one sprite on the screen, you may need to take into consideration their relative video priority, that is, which sprite appears in front of or behind another. Each sprite has a fixed video priority with respect to all the others. The lowest numbered sprite has the highest priority and appears in front of all other sprites; the highest numbered sprite has the lowest priority. This is illustrated in Figure 4-8.

More about priorities.

See Chapter 7, "System Control Hardware," for more information on sprite priorities .

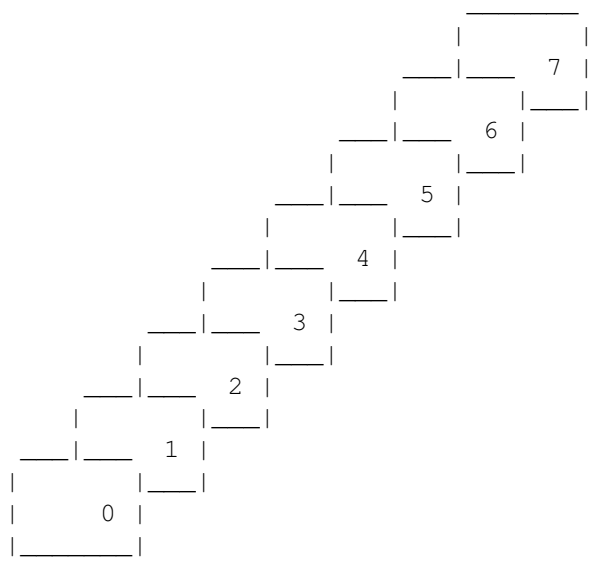
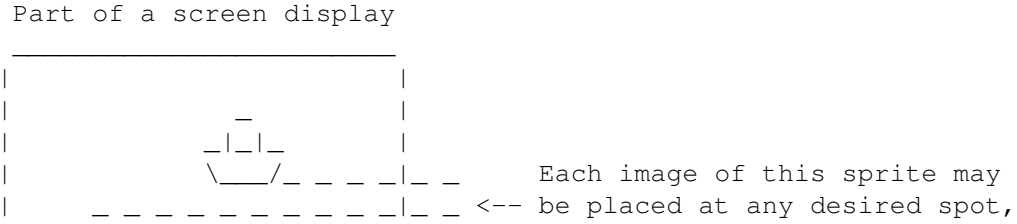


Figure 4-8: Sprite Priority

1.23 4 Sprite Hardware / Reusing Sprite DMA Channels

Each of the eight sprite DMA channels can produce more than one independently controllable image. There may be times when you want more than eight objects, or you may be left with fewer than eight objects because you have attached some of the sprites to produce more colors or larger objects or overlapped some to produce more complex images. You can reuse each sprite DMA channel several times within the same display field, as shown in Figure 4-9.



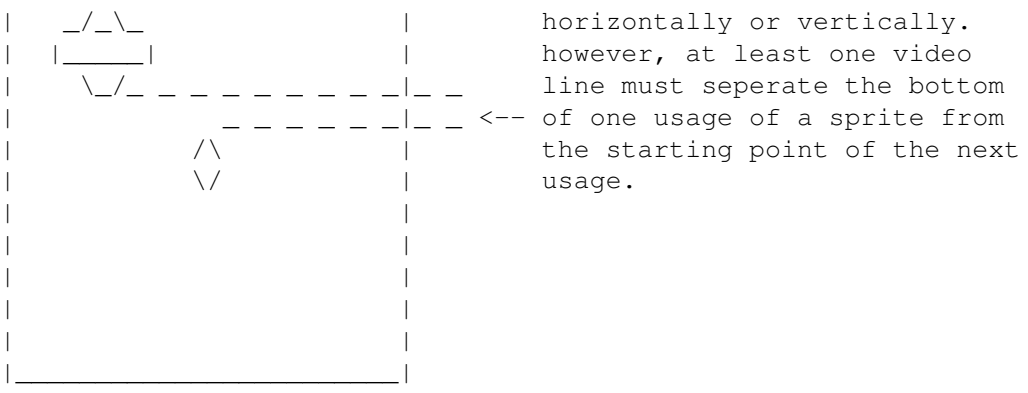
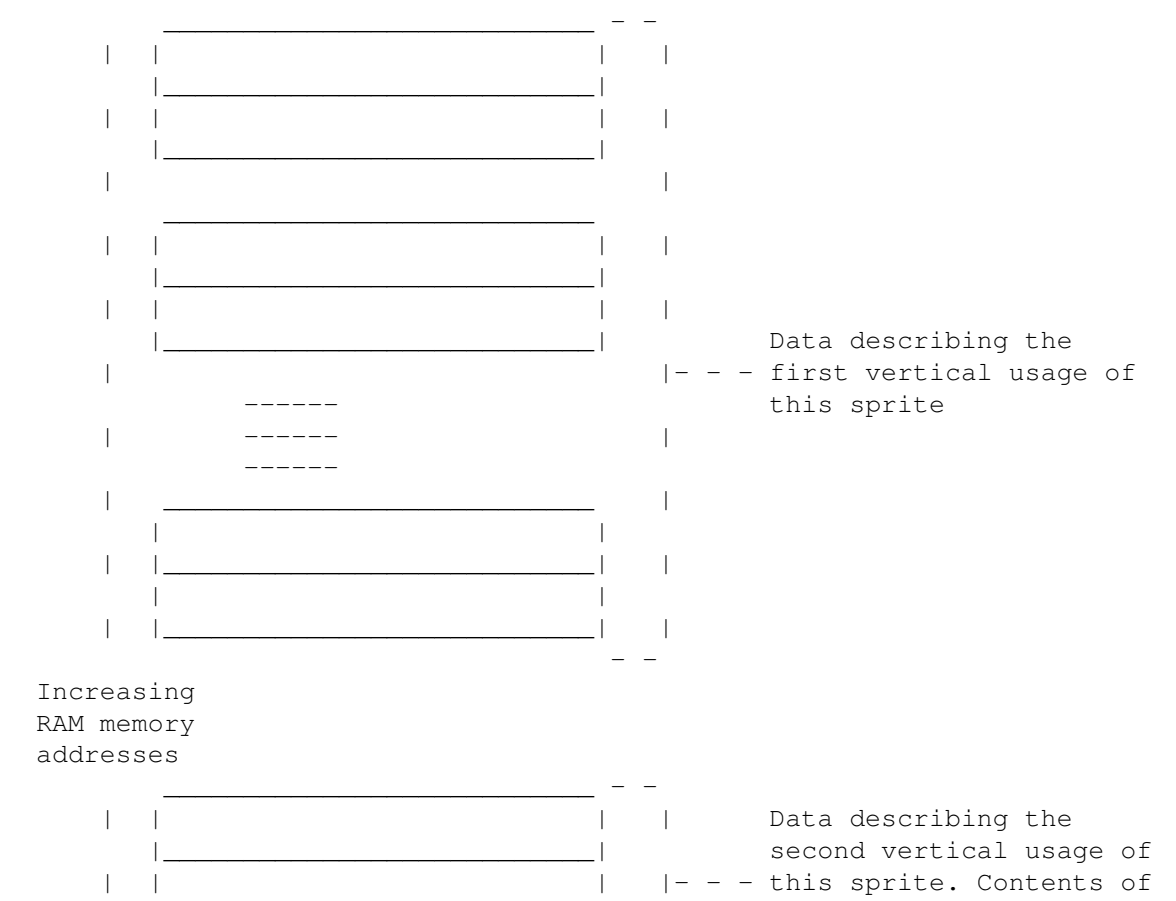


Figure 4-9: Typical Example of Sprite Reuse

In single-sprite usage, two all-zero words are placed at the end of the data structure to stop the DMA channel from retrieving any more data for that particular sprite during that display field. To reuse a DMA channel, you replace this pair of zero words with another complete sprite data structure , which describes the reuse of the DMA channel at a position lower on the screen than the first use. You place the two all-zero words at the end of the data structure that contains the information for all usages of the DMA channel. For example, Figure 4-10 shows the data structure that describes the picture above.



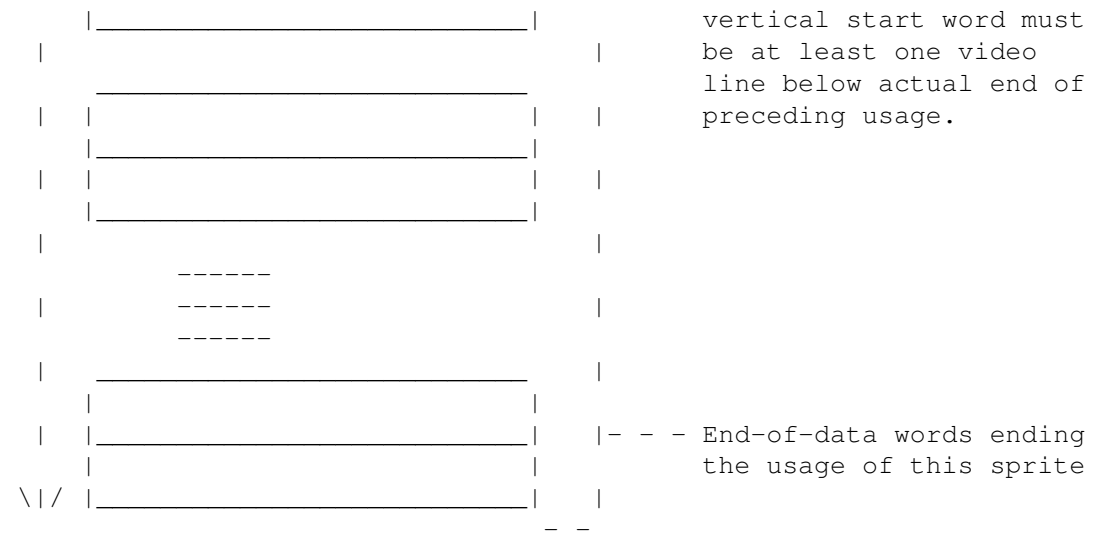


Figure 4-10: Typical Data Structure for Sprite Re-use

The only restrictions on the reuse of sprites during a single display field is that the bottom line of one usage of a sprite must be separated from the top line of the next usage by at least one horizontal scan line. This restriction is necessary because only two DMA cycles per horizontal scan line are allotted to each of the eight channels. The sprite channel needs the time during the blank line to fetch the control word describing the next usage of the sprite.

The following example displays the spaceship sprite and then redisplays it as a different object. Only the sprite data list is affected, so only the data list is shown here. However, the sprite looks best with the color registers set as shown in the example.

```
        LEA     CUSTOM,a0
        MOVE.W  #$0F00,COLOR17(a0)      ;Color 17 = red
        MOVE.W  #$0FF0,COLOR18(a0)      ;Color 18 = yellow
        MOVE.W  #$0FFF,COLOR19(a0)      ;Color 19 = white
SPRITE:
        DC.W    $6D60,$7200
        DC.W    $0990,$07E0
        DC.W    $13C8,$0FF0
        DC.W    $23C4,$1FF8
        DC.W    $13C8,$0FF0
        DC.W    $0990,$07E0
        DC.W    $8080,$8D00     ;VSTART, HSTART, VSTOP for new sprite
        DC.W    $1818,$0000
        DC.W    $7E7E,$0000
        DC.W    $7FFE,$0000
        DC.W    $FFFF,$2000
        DC.W    $FFFF,$2000
        DC.W    $FFFF,$3000
        DC.W    $FFFF,$3000
        DC.W    $7FFE,$1800
        DC.W    $7FFE,$0C00
        DC.W    $3FFC,$0000
```

```
        DC.W    $0FF0,$0000
        DC.W    $03C0,$0000
        DC.W    $0180,$0000
        DC.W    $0000,$0000     ;End of sprite data
```

1.24 4 Sprite Hardware / Overlapped Sprites

For more complex or larger moving objects, you can overlap sprites. Overlapping simply means that the sprites have the same or relatively close screen positions. A relatively close screen position can result in an object that is wider than 16 pixels.

The built-in sprite video priority ensures that one sprite appears to be behind the other when sprites are overlapped. The priority circuitry gives the lowest-numbered sprite the highest priority and the highest numbered sprite the lowest priority. Therefore, when designing displays with overlapped sprites, make sure the "foreground" sprite has a lower number than the "background" sprite. In Figure 4-11, for example, the cage should be generated by a lower-numbered sprite DMA channel than the monkey.

Figure 4-11: Overlapping Sprites (Not Attached)

You can create a wider sprite display by placing two sprites next to each other. For instance, Figure 4-12 shows the spaceship sprite and how it can be made twice as large by using two sprites placed next to each other.

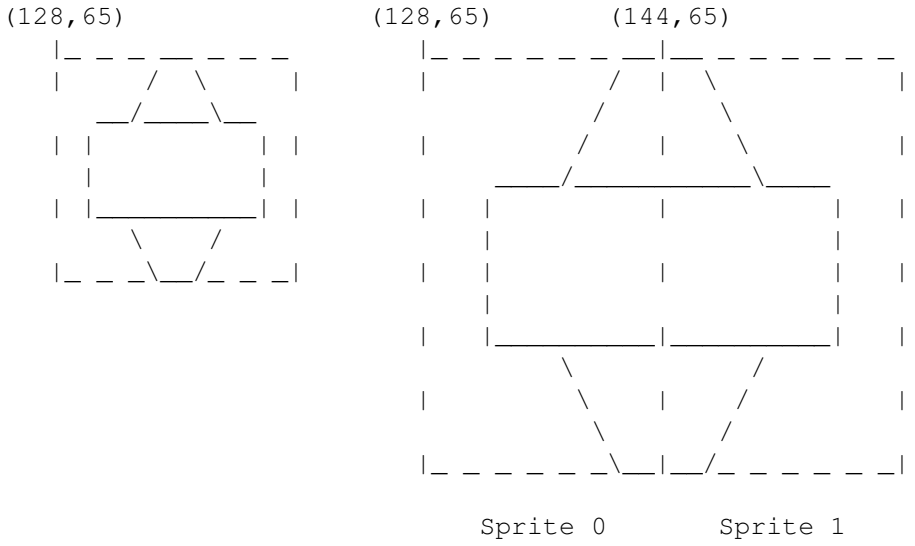


Figure 4-12: Placing Sprites Next to Each Other

1.25 4 Sprite Hardware / Attached Sprites

You can create sprites that have fifteen possible color choices (plus transparent) instead of three (plus transparent), by "attaching" two

sprites. To create attached sprites, you must:

* Use two channels per sprite, creating two sprites of the same size and located at the same position.

* Set a bit called ATTACH in the second sprite control word .

The fifteen colors are selected from the full range of color registers available to sprites -- registers 17 through 31. The extra color choices are possible because each pixel contains four bits instead of only two as in the normal, unattached sprite. Each sprite in the attached pair contributes two bits to the binary color selector number. For example, if you are using sprite DMA channels 0 and 1, the high and low order color descriptor words for line 1 in both data structures are combined into line 1 of the attached object.

Sprites can be attached in the following combinations:

Sprite 1 to sprite 0
Sprite 3 to sprite 2
Sprite 5 to sprite 4
Sprite 7 to sprite 6

Any or all of these attachments can be active during the same display field. As an example, assume that you wish to have more colors in the spaceship sprite and you are using sprite DMA channels 0 and 1. There are five colors plus transparent in this sprite.

0000154444510000
0001564444651000
0015676446765100
0001564444651000
0000154444510000

The first line in this sprite requires the four data words shown in Table 4-4 to form the correct binary color selector numbers.

Table 4-4: Data Words for First Line of Spaceship Sprite

	Pixel Number															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Line 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Line 2	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0
Line 3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Line 4	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0

The highest numbered sprite (number 1, in this example) contributes the highest order bits (leftmost) in the binary number. The high order data word in each sprite contributes the leftmost digit. Therefore, the

lines above are written to the sprite data structures as follows:

Line 1	Sprite 1 high order word for sprite line 1
Line 2	Sprite 1 low order word for sprite line 1
Line 3	Sprite 0 high order word for sprite line 1
Line 4	Sprite 0 low order word for sprite line 1

See Figure 4-7 for the order these words are stored in memory. Remember that this data is contained in two sprite structures.

The binary numbers 0 through 15 select registers 17 through 31 as shown in Table 4-5.

Table 4-5: Color Registers in Attached Sprites

Decimal Number	Binary Number	Color Register Number
0	0000	16 *
1	0001	17
2	0010	18
3	0011	19
4	0100	20
5	0101	21
6	0110	22
7	0111	23
8	1000	24
9	1001	25
10	1010	26
11	1011	27
12	1100	28
13	1101	29
14	1110	30
15	1111	31

* Unused; yields transparent pixel.

Attachment is in effect only when the ATTACH bit, bit 7 in sprite control word 2 , is set to 1 in the data structure for the odd-numbered sprite. So, in this example, you set bit 7 in sprite control word 2 in the data structure for sprite 1.

When the sprites are moved, the Copper list must keep them both at exactly the same position relative to each other. If they are not kept together on the screen, their pixels will change color. Each sprite will revert to three colors plus transparent, but the colors may be different than if they were ordinary, unattached sprites. The color selection for the lower numbered sprite will be from color registers 17-19. The color selection for the higher numbered sprite will be from color registers 20, 24, and 28.

The following data structure is for the six-color spaceship made with two attached sprites.

```
SPRITE0:
        DC.W    $6D60,$7200     ;VSTART = 65, HSTART = 128
        DC.W    $0C30,$0000     ;First  color descriptor word
        DC.W    $1818,$0420
        DC.W    $342C,$0E70
        DC.W    $1818,$0420
        DC.W    $0C30,$0000
        DC.W    $0000,$0000     ;End of sprite 0
SPRITE1:
        DC.W    $6D60,$7280     ;Same as sprite 0 except  attach  bit on
        DC.W    $07E0,$0000     ;First descriptor word for sprite 1
        DC.W    $0FF0,$0000
        DC.W    $1FF8,$0000
        DC.W    $0FF0,$0000
        DC.W    $07E0,$0000
        DC.W    $0000,$0000     ;End of sprite 1
```

1.26 4 Sprite Hardware / Manual Mode

It is almost always best to load sprites using the automatic DMA channels. Sometimes, however, it is useful to load these registers directly from one of the microprocessors. Sprites may be activated "manually" whenever they are not being used by a DMA channel. The same sprite that is showing a DMA-controlled icon near the top of the screen can also be reloaded manually to show a vertical colored bar near the bottom of the screen. Sprites can be activated manually even when the sprite DMA is turned off.

You display sprites manually by writing to the sprite data registers SPRxDATB and SPRxDATA , in that order. You write to SPRxDATA last because that address "arms" the sprite to be output at the next horizontal comparison. The data written will then be displayed on every line, at the horizontal position given in the "H" portion of the position registers SPRxPOS and SPRxCTL . If the data is unchanged, the result will be a vertical bar. If the data is reloaded for every line, a complex sprite can be produced.

The sprite can be terminated ("disarmed") by writing to the SPRxCTL register. If you write to the SPRxPOS register, you can manually move the sprite horizontally at any time, even during normal sprite usage.

1.27 4 Sprite Hardware / Sprite Hardware Details

Sprites are produced by the circuitry shown in Figure 4-13. This figure shows in block form how a pair of data words becomes a set of pixels displayed on the screen.

Figure 4-13: Sprite Control Circuitry

The circuitry elements for sprite display are explained below.

* Sprite data registers.

The registers SPRxDATA and SPRxDATB hold the bit patterns that describe one horizontal line of a sprite for each of the eight sprites. A line is 16 pixels wide, and each line is defined by two words to provide selection of three colors and transparent.

* Parallel-to-serial converters.

Each of the 16 bits of the sprite data bit pattern is individually sent to the color select circuitry at the time that the pixel associated with that bit is being displayed on-screen.

Immediately after the data is transferred from the sprite data registers, each parallel-to-serial converter begins shifting the bits out of the converter, most significant (leftmost) bit first. The shift occurs once during each low resolution pixel time and continues until all 16 bits have been transferred to the display circuitry. The shifting and data output does not begin again until the next time this converter is loaded from the data registers.

Because the video image is produced by an electron beam that is being swept from left to right on the screen, the bit image of the data corresponds exactly to the image that actually appears on the screen (most significant data on the left).

* Sprite serial video data.

Sprite data goes to the priority circuit to establish the priority between sprites and playfields.

* Sprite position registers.

These registers, called SPRxPOS , contain the horizontal position value (X value) and vertical position value (Y value) for each of the eight sprites.

* Sprite control registers.

These registers, called SPRxCTL , contain the stopping position for each of the eight sprites and whether or not a sprite is attached.

* Beam counter.

The beam counter tells the system the current location of the video beam that is producing the picture.

* Comparator.

This device compares the value of the beam counter to the Y value in the position register SPRxPOS . If the beam has reached the position at which the leftmost upper pixel of the sprite is to appear, the comparator issues a load signal to the serial-to-parallel converter and the sprite display begins.

Figure 4-13 shows the following:

* Writing to the sprite control registers disables the horizontal

comparator circuitry. This prevents the system from sending any output from the data registers to the serial converter or to the screen.

* Writing to the sprite A data register enables the horizontal comparator. This enables output to the screen when the horizontal position of the video beam equals the horizontal value in the position register.

* If the comparator is enabled, the sprite data will be sent to the display, with the leftmost pixel of the sprite data placed at the position defined in the horizontal part of SPRxPOS .

* As long as the comparator remains enabled, the current contents of the sprite data register will be output at the selected horizontal position on a video line.

* The data in the sprite data registers does not change. It is either rewritten by the user or modified under DMA control.

The components described above produce the automatic DMA display as follows: When the sprites are in DMA mode, the 18-bit sprite pointer register (composed of SPRxPTH and SPRxPTL) is used to read the first two words from the sprite data structure . These words contain the starting and stopping position of the sprite. Next, the pointers write these words into SPRxPOS and SPRxCTL . After this write, the value in the pointers points to the address of the first data word (low word of data for line 1 of the sprite.)

Writing into the SPRxCTL register disabled the sprite. Now the sprite DMA channel will wait until the vertical beam counter value is the same as the data in the VSTART (Y value) part of SPRxPOS . When these values match, the system enables the sprite data access.

The sprite DMA channel examines the contents of VSTOP (from SPRxCTL , which is the location of the line after the last line of the sprite) and VSTART (from SPRxPOS) to see how many lines of sprite data are to be fetched. Two words are fetched per line of sprite height, and these words are written into the sprite data registers. The first word is stored in SPRxDATA and the second word in SPRxDATB .

The fetch and store for each horizontal scan line occurs during a horizontal blanking interval, far to the left of the start of the screen display. This arms the sprite horizontal comparators and allows them to start the output of the sprite data to the screen when the horizontal beam count value matches the value stored in the HSTART (X value) part of SPRxPOS .

If the count of VSTOP - VSTART equals zero, no sprite output occurs. The next data word pair will be fetched, but it will not be stored into the sprite data registers. It will instead become the next pair of data words for SPRxPOS and SPRxCTL .

When a sprite is used only once within a single display field, the final pair of data words, which follow the sprite color descriptor words , is loaded automatically as the next contents of the SPRxPOS and SPRxCTL registers. To stop the sprite after that first data set, the pair of words

should contain all zeros.

Thus, if you have formed a sprite pattern in memory, this same pattern will be produced as pixels automatically under DMA control one line at a time.

1.28 4 Sprite Hardware / Summary of Sprite Registers

There are eight complete sets of registers used to describe the sprites. Each set consists of five registers. Only the registers for sprite 0 are described here. All of the others are the same, except for the name of the register, which includes the appropriate number.

Pointers
Control Registers
Data Registers

1.29 4 / Summary of Sprite Registers / Pointers

Pointers are registers that are used by the system to point to the current data being used. During a screen display, the registers are incremented to point to the data being used as the screen display progresses. Therefore, pointer registers must be freshly written during the start of the vertical blanking period.

SPR0PTH and SPR0PTL

1.30 4 / / Pointers / SPR0PTH and SPR0PTL

This pair of registers contains the 32-bit word address of Sprite 0 DMA data.

Pointer register names for the other sprites are:

SPR1PTH	SPR1PTL
SPR2PTH	SPR2PTL
SPR3PTH	SPR3PTL
SPR4PTH	SPR4PTL
SPR5PTH	SPR5PTL
SPR6PTH	SPR6PTL
SPR7PTH	SPR7PTL

1.31 4 / Summary of Sprite Registers / Control Registers

SPR0POS
SPR0CTL

1.32 4 / / Control Registers / SPR0POS

This is the sprite 0 position register. The word written into this register controls the position on the screen at which the upper left-hand corner of the sprite is to be placed. The most significant bit of the first data word will be placed in this position on the screen.

Sprite placement resolution.

The sprites have a placement resolution on a full screen of 320 by 200 NTSC (320 by 256 PAL). The sprite resolution is independent of the bitplane resolution.

Bit positions:

* Bits 15-8 specify the vertical start position, bits V7 - V0.
* Bits 7-0 specify the horizontal start position, bits H8 - H1.

Warning:

This register is normally only written by the sprite DMA channel itself. See the details above regarding the organization of the sprite data. This register is usually updated directly by DMA.

1.33 4 / / Control Registers / SPR0CTL

This register is normally used only by the sprite DMA channel . It contains control information that is used to control the sprite data-fetch process. Bit positions:

* Bits 15-8 specify vertical stop position for a sprite image, bits V7 - V0.

* Bit 7 is the attach bit. This bit is valid only for odd-numbered sprites. It indicates that sprites 0, 1 (or 2,3 or 4,5 or 6,7) will, for color interpretation, be considered as paired, and as such will be called four bits deep. The odd-numbered (higher number) sprite contains bits with the higher binary significance.

  During attach mode, the attached sprites are normally moved horizontally and vertically together under processor control. This allows a greater selection of colors within the boundaries of the sprite itself. The sprites, although attached, remain capable of independent motion, however, and they will assume this larger color set only when their edges overlay one another.

* Bits 6-3 are reserved for future use (make zero).
* Bit 2 is bit V8 of vertical start.
* Bit 1 is bit V8 of vertical stop.
* Bit 0 is bit H0 of horizontal start.

Position and control registers for the other sprites work the same way as described above for sprite 0. The register names for the other sprites are:

```
SPR1POS    SPR1CTL
SPR2POS    SPR2CTL
SPR3POS    SPR3CTL
SPR4POS    SPR4CTL
SPR5POS    SPR5CTL
SPR6POS    SPR6CTL
SPR7POS    SPR7CTL
```

1.34 4 / Summary of Sprite Registers / Data Registers

The following registers (SPR0DATA and SPR0DATB), although defined in the address space of the main processor, are normally used only by the display processor. They are the holding registers for the data obtained by DMA cycles.

```
SPR0DATA, SPR0DATB      data registers for Sprite 0
SPR1DATA, SPR1DATB      data registers for Sprite 1
SPR2DATA, SPR2DATB      data registers for Sprite 2
SPR3DATA, SPR3DATB      data registers for Sprite 3
SPR4DATA, SPR4DATB      data registers for Sprite 4
SPR5DATA, SPR5DATB      data registers for Sprite 5
SPR6DATA, SPR6DATB      data registers for Sprite 6
SPR7DATA, SPR7DATB      data registers for Sprite 7
```

1.35 4 Sprite Hardware / Summary of Sprite Color Registers

Sprite data words are used to select the color of the sprite pixels from the system color register set as indicated in the following tables.

If the bit combinations from single sprites are as shown in Table 4-6, then the colors will be taken from the registers shown.

Table 4-6: Color Registers for Single Sprites

	Single Sprites	
Sprite	Value	Color Register
0 or 1	00	Not used *
	01	17
	10	18
	11	19
2 or 3	00	Not used *
	01	21
	10	22
	11	23

4 or 5	00	Not used *
	01	25
	10	26
	11	27
6 or 7	00	Not used *
	01	29
	10	30
	11	31

* Selects transparent mode.

If the bit combinations from attached sprites are as shown in Table 4-7, then the colors will be taken from the registers shown.

Table 4-7: Color Registers for Attached Sprites

Attached Sprites	
Value	Color Register
0000	Selects transparent mode
0001	17
0010	18
0011	19
0100	20
0101	21
0110	22
0111	23
1000	24
1001	25
1010	26
1011	27
1100	28
1101	29
1110	30
1111	31

Interactions Among Sprites and Other Objects

1.36 4 / Color Registers / Interactions Among Sprites and Other Objects

Playfields share the display with sprites. Chapter 7, "System Control Hardware," shows how playfields can be given different video display priorities relative to the sprites and how playfields can collide with (overlap) the sprites or each other.

ECS Sprites.

For information relating to sprites in the Enhanced Chip Set (ECS),

such as SuperHires sprites and SuperHires sprite positioning ,
see Appendix C.