Apollo Core
SAGA Technical Reference Manual

revision 1.0

chapter one
Sprite Hardware

What are Sprites?

Sprites are graphics objects that are created and moved independently from the rest of the display hardware and from each other.

Sprite Evolution

The original OCS/ECS chipset featured 8 special-purpose sprite DMA channels. In their basic form a sprite can be 16 pixels wide and any number of lines high. These basic sprites could select from a palette of 3 colours per pixel or have a pixel be transparent. Sprites may be "combined" to form larger objects on screen or increase the colours available for use with a sprite. To increase the width of a sprite would be a simple matter of using two sprites horizontally. A special feature was made available to "attach" sprites in specific combinations to increase the available colour palette to 15 colours + transparent. Attaching sprites will be discussed later.

Sprite DMA channels can be re-used during the active display, making it possible to create more than 8 sprites on-screen at any one time.

Sprites may become unavailable especially when using wider displays due to the limitations in DMA slots.

AGA Enhancements

- Sprites can select between ***LORES***, ***HIRES*** and ***SHIRES*** resolutions independent of bit-plane resolution.
- Sprites can be 16, 32- or 64-pixels wide.
- Attached sprites became available in all resolutions.
- Hardware scan-doubling for sprites became available allowing them to be displayed on 31khz monitors. Scan doubling can be used on 15khz displays causing sprites to be repeated per row.
- Sub-pixel sprite positioning in low resolution modes.
- Palette bank switching allowing the sprites to use colours independent from the bit-planes.

SAGA (AGA+) Enhancements

- 16 Sprite DMA channels, allowing 16 on-screen sprites and DMA re-use.
- No sprite channel loss due to DMA bandwidth loss found in AGA.
- Sprite data can reside in Fast RAM.
- Sprites can be horizontally repeated without resorting to copper tricks.
- Sprites support an indirect data mode allowing for fast sprite-data switching.
- Sprites use their own palette bank of 256 colours.
- Every sprite can use its own 16 colour set from this bank.
- No need to attach sprites to obtain 16 colours.
- Palette entries use 24bit *Red*, *Green* and *Blue* components for enhanced colour fidelity.
- A new palette register, and copper command allow moving the full 32bit palette entry in a single instruction.
- AGA+ sprite features are controlled by a new bit in the **FMODE** register to ensure backwards compatibility with OCS/ECS/AGA.

Screen Positioning

A sprite's position is defined as a set of X and Y coordinates which represent it's upper left corner. The X/Y value is always defined based on a low-resolution non-interlaced display regardless of the resolution of the rest of the display hardware. Position (0,0) is not normally in the viewable region of the screen due to display overscan, as described in the ***Playfield Hardware*** chapter. The amount of displayable area is also affected by the size of the playfield display window. This is controlled via the registers **DDFSTRT**, **DDFSTOP**, **DIWSTRT**, **DIWSTOP** etc.

Clipping is automatically applied to the bounds of the display and playfield display window.

The horizontal and vertical position (X/Y values) of the Sprite must take into account the display window offset. For example, to display a Sprite at 94 pixels from the left edge and 25 pixels from the top edge with a display window offset of 64 and 44 would require the calculations:

Desired X Position + Horizontal-Offset of Display Window = 94 + 64 = 158
Desired Y Position + Vertical-Offset of Display Window = 25 + 44 = 69

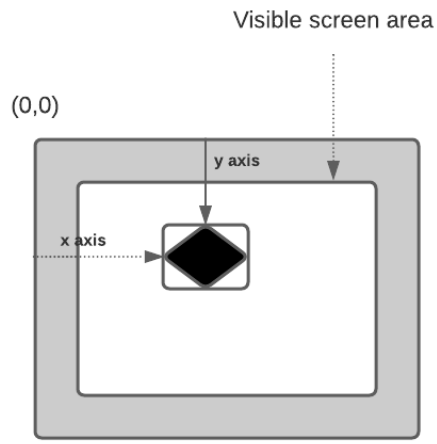


Figure 1-1: Defining Sprite On-Screen Position

- Any horizontal position between 0 and 447 may be used.
- Any vertical position between 0 and 262 may be used.

The values of (64 horizontal and 44 vertical) are normal for an **NTSC** display.

Sprite Sizes

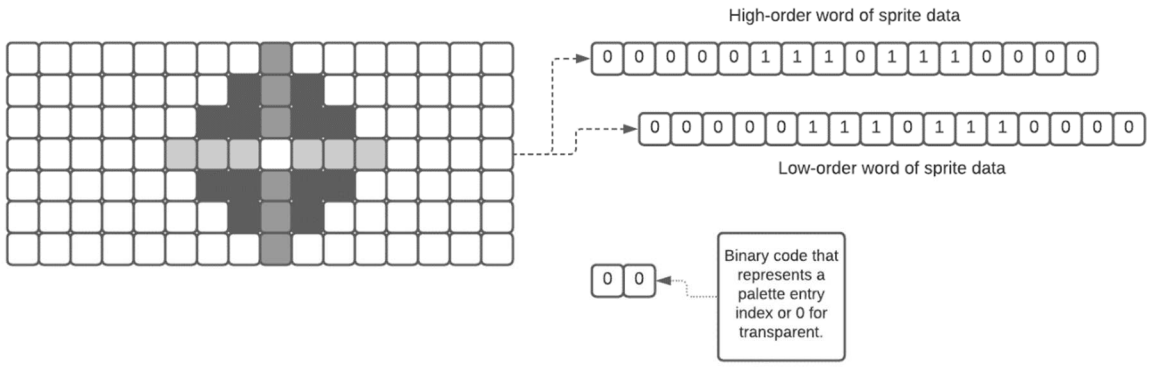
As mentioned previously sprites can be any height from a single row to taller than the screen. Under OCS/ECS sprites are limited to 16 pixels wide. AGA and AGA+ offer a choice of 16, 32- or 64-pixel wide sprites. Sprite size is based on a "*pixel*" that is 1/320th of the display's width and 1/256th of the display's height (or 1/200th for **NTSC**). This is independent of the resolution or interlace setting of the playfield hardware.

Sprite Image Data (Non-Attached)

Sprite image data is defined in a planar fashion, similar to how the playfield hardware operates. 16-bit Data Words are combined to form the index into the sprites colour palette. For OCS/ECS in non-attached mode the system colour registers 16-31 are used. The 8 sprites are organized into pairs and each pair share the same 4 colour entries.

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

Table 1 – Colour Registers for Classic Sprites



The binary value "00" is handled specially and indicates a transparent pixel. This will show the contents from behind that pixel depending on both the sprite priority and sprite to playfield priority.

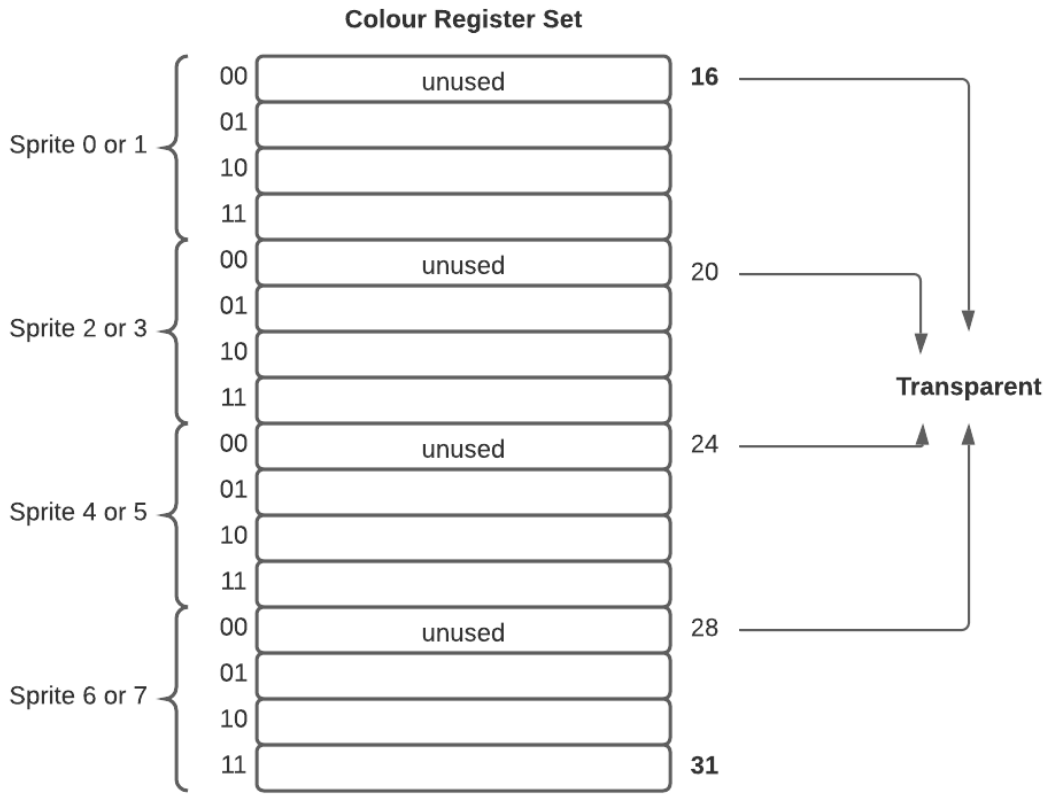


Figure 1-2: Colour Register Usage

Sprite Data Structure

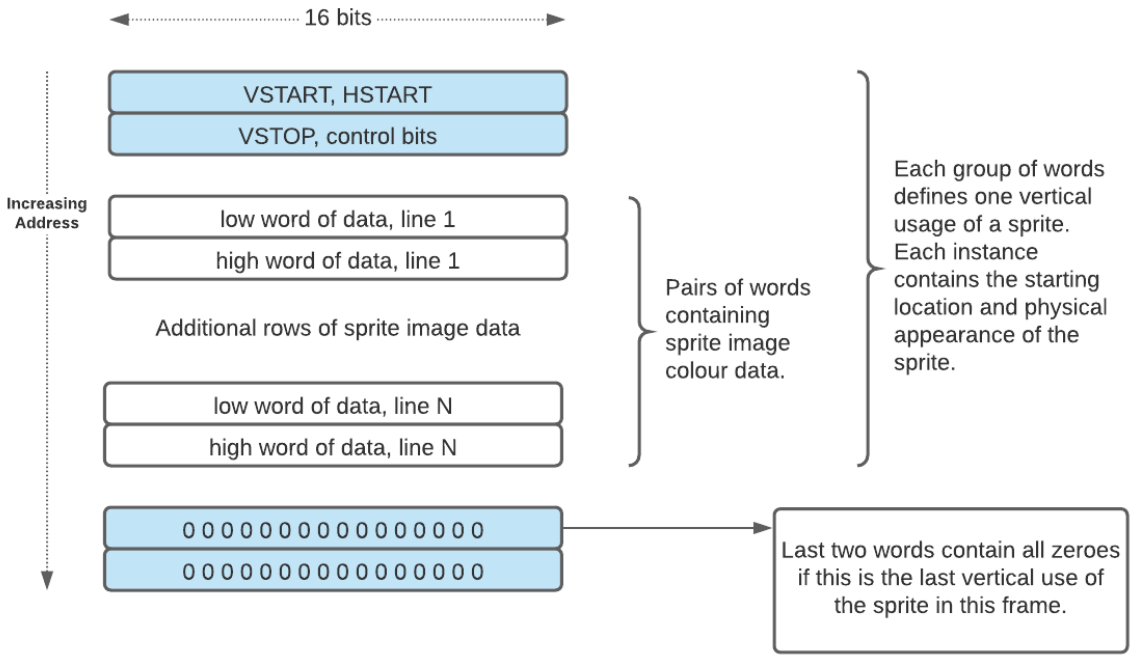
Sprite data is constructed in memory as a series of 16-bit words. Some of the words contain control information for the sprite, while the remainder contain the image data. The basic structure is as follows:

Memory Location	16-bit Word	Function
N	Sprite Control Word 1	Vertical and Horizontal Start Position
N+2	Sprite Control Word 2	Vertical Stop Position
N+4	Sprite Binary Image Data Low Word 1	Row 1 Colour Data
N+6	Sprite Binary Image Data High Word 1	
N+8	Sprite Binary Image Data Low Word 2	Row 2 Colour Data
N+10	Sprite Binary Image Data High Word 2	
N+..	End-Of-Data Words	Two words indicating the next usage of this sprite

Note

Sprite Data should be aligned on a 16-bit, 32-bit or 64-bit boundaries as determined by the size of sprite in use (16-pixel, 32-pixel, 64-pixel)

For original Amiga systems Sprite Data must reside in Chip Memory. This restriction is *not* present in the Apollo Core which allows Sprite Data to reside anywhere including Fast RAM.



Sprite Control Word 1: SPRxPOS

This word contains the vertical (VSTART) and horizontal (HSTART) starting position for the sprite.

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

Sprite Control Word 2: SPRxCTL

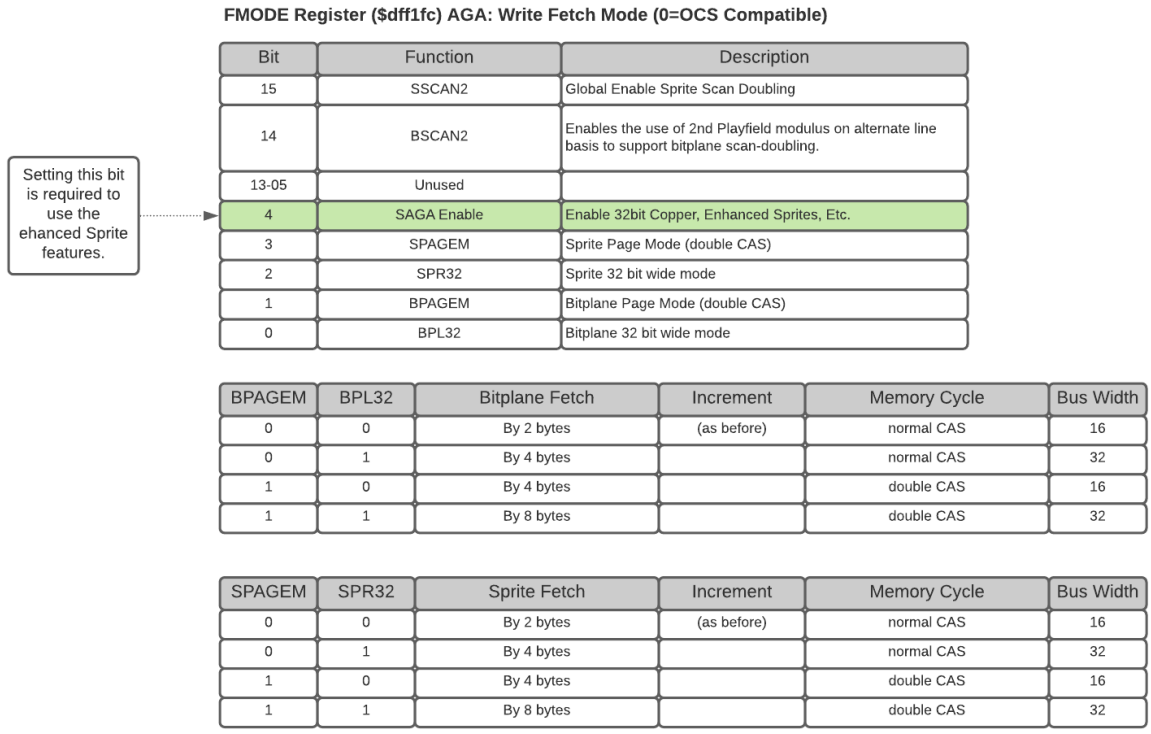
Bits 15-8	The low eight bits of **VSTOP**
Bit 7	**(Used in attachment)**
Bits 6-3	Unused (make zero)
Bit 2	**VSTART** high bit
Bit 1	**VSTOP** high bit
Bit 0	**HSTART** low bit

End-Of-Data Words

When the vertical position of the beam counter is equal to VSTOP value in the Sprite control words, t he next two words fetched from the data are written to the Sprite Control Registers instead of being sent via the colour registers for display. These two words are interpreted by the hardware in the same manner as the original words that were first loaded at the start of the Sprite data.

If the VSTART value of these end words is less than the current beam counter the Sprite will NOT be re-used during the display. For consistency the value 0 should be used for both words when ending the usage of the Sprite.

Enabling AGA+ Operation



FMODE Register ($dff1fc) AGA: Write Fetch Mode (0=OCS Compatible)

Bit	Function	Description
15	SSCAN2	Global Enable Sprite Scan Doubling
14	BSCAN2	Enables the use of 2nd Playfield modulus on alternate line basis to support bitplane scan-doubling.
13-05	Unused	
4	SAGA Enable	Enable 32bit Copper, Enhanced Sprites, Etc.
3	SPAGEM	Sprite Page Mode (double CAS)
2	SPR32	Sprite 32 bit wide mode
1	BPAGEM	Bitplane Page Mode (double CAS)
0	BPL32	Bitplane 32 bit wide mode

BPAGEM	BPL32	Bitplane Fetch	Increment	Memory Cycle	Bus Width
0	0	By 2 bytes	(as before)	normal CAS	16
0	1	By 4 bytes		normal CAS	32
1	0	By 4 bytes		double CAS	16
1	1	By 8 bytes		double CAS	32

SPAGEM	SPR32	Sprite Fetch	Increment	Memory Cycle	Bus Width
0	0	By 2 bytes	(as before)	normal CAS	16
0	1	By 4 bytes		normal CAS	32
1	0	By 4 bytes		double CAS	16
1	1	By 8 bytes		double CAS	32

Figure 1-3: FMODE Register Usage

To enable AGA+ Sprite operation **SAGA Enable**, **SPAGEM** and **SPR32** *must all* be enabled. All sprites in AGA+ mode are 64-bit aligned and 64-pixels wide.

Sprite Data for AGA 32- and 64-Pixel Wide Sprites

To enable 32- or 64-pixel wide sprites requires changing the bit settings in the **FMODE** register as follows:

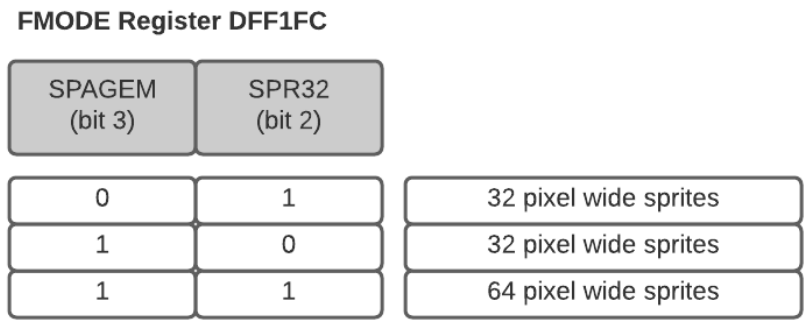


Figure 1-4: FMODE Register Settings

For 32-pixel wide sprites the data must be aligned on a *32-bit boundary* and each entry in the data-structure becomes a full 32-bit value with padding:

```
dc.w $2020, $3000                  ; 16-bit equivalent would become:

dc.w $2020, $0000, $3000, $0000    ; or
dc.l $20200000, $30000000
```

For 64-pixel wide sprites the data must be aligned on a *64-bit boundar*y and each entry in the data-structure becomes a full 64-bit value with padding:

```
dc.w $2020, $3000                  ; 16-bit equivalent would become:

dc.w $2020, $0000, $0000, $0000, $3000, $0000, $0000, $0000    ; or
dc.l $20200000, $00000000, $30000000, $00000000
```

AGA+ Extended Sprite Data Structure

AGA+ mode uses ONLY 32-pixel wide sprites. This feature is by design to ensure that the same amount of data per row (64-bits) can be used to provide the full 4-bit (16 colour) index as opposed to having a 64-pixel wide 4 colour only sprite.

The data structure for AGA+ operation is thus, quite different from AGA.

Control Words are extended to 64-bit, a new flag in bit 1 has been added to mark the sprite for horizontal repeat.

Sprite image data no longer makes use of attachment, instead all 4 bits are directly included in the sprite's data.

The sprite's image data is indirectly referenced via a pointer in the sprite control data-structure. The advantage of this approach is that it allows a single sprite control structure to be updated to pointed at a different image (for example an animated sprite) each frame without having to re-write the image data into the existing structure or create unnecessary extra control words on each sprite image frame. Another advantage to this is that each sprite's control structure is now a fixed size.

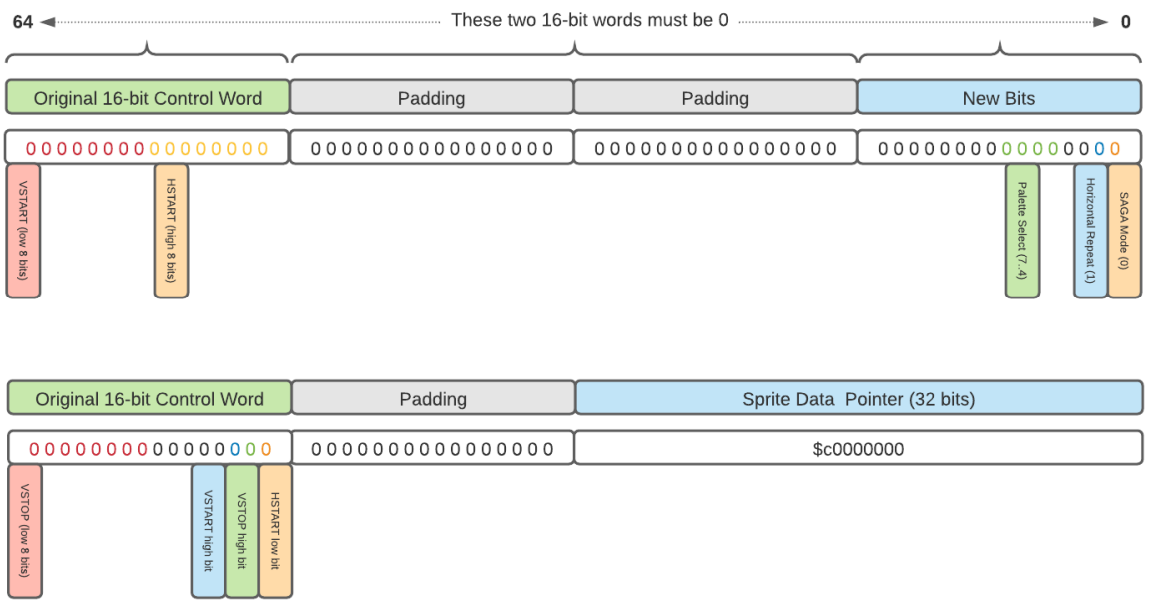


Figure 1-5: AGA+ Control Word Structure

```
   CNOP 0,8

Sprite0_Control:
dc.l $50600000, $00000021
dc.l $60000000, PlayerImage

PlayerImage:
dc.l $ffffffff, $00000000, $ffffffff, $00000000
```

- The Sprite Image data would yield a binary value of 1010 for each of the 32 pixels in that row.

AGA Sprite Colour Bank Selection

AGA provides some flexibility in allowing sprites to map to different ranges of the full 256 colour palette. This is accomplished through bits 3..0 in the **BPLCON4** registers which allows the assignment of a 16-color page for odd numbered sprites and bits 7..4 for even numbered sprites. Attached sprites in AGA will both use the same palette page as defined by the odd numbered sprite.

AGA Horizontal Sprite Tiling

The **SSCAN2** bit in the **FMODE** register can be used to cause sprites to appear twice per horizontal row. This trick has been used in several classic Amiga games such as Fantastic Dizzy CD32 to create a repeating tree background layer. Four 64-pixel wide sprites are used in attached mode and then a repeat happens at horizontal position + 256 pixels.

Extended Colour Palette Operation

When operating in AGA+ mode a separate colour palette is made available allowing Sprites to use a completely unique palette from bit-plane image data.

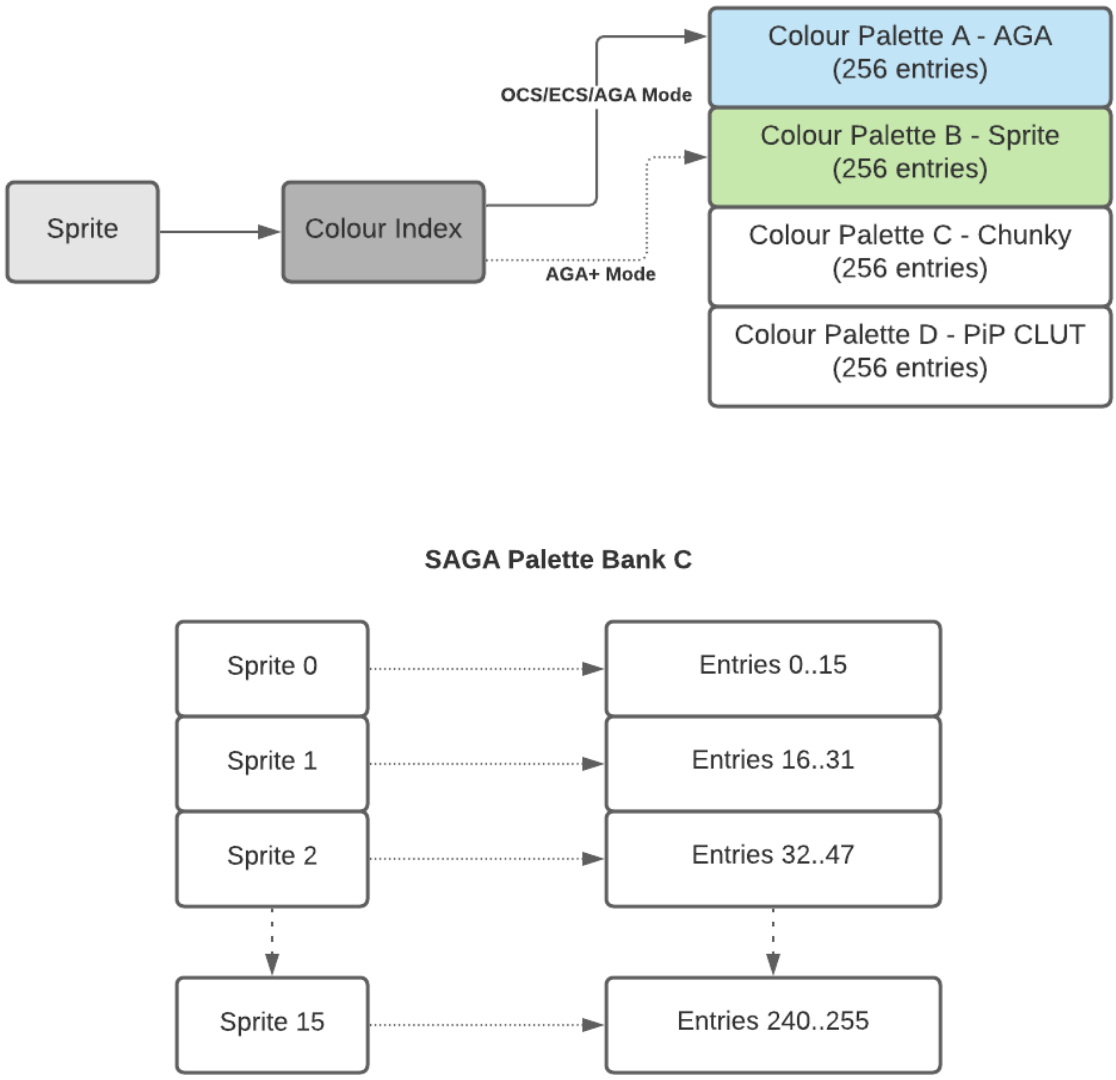
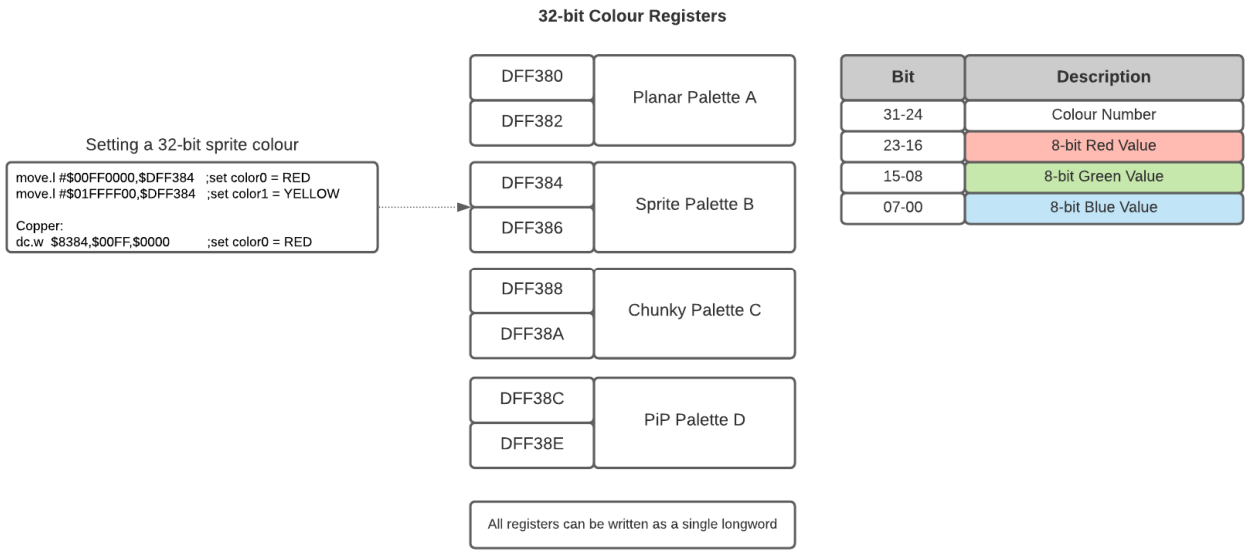


Figure 1-4: AGA+ Colour Palette Banks

Fast Colour Setting Registers

A new 32-bit longword colour register is provided per bank which supports a full 24-bit colour and allows setting the palette entry in a single operation either from the CPU or from the Copper via a new Copper Move Long instruction.



- As long as the **FMODE SAGA** enable bit has been set the new 32-bit Copper Move instruction is identical to a regular 16-bit move with the highest order bit set.

Displaying a Sprite

Once Sprite Data has been constructed you need to tell the system to display it. This section describes “automatic” or DMA based Sprite display. In this mode the Sprite DMA channel will automatically fetch the data and continue to display it until it reaches the VSTOP value. Operation will then continue based on how the End-Of-Data words have been configured.

The follow steps are used in displaying a Sprite:

1. Decide which of the eight (or 16 for AGA+) Sprite DMA channels to use.
2. Set the Sprite pointers to tell the system where to find the Sprite Data.
3. Turn on the Sprite DMA if not already on.
4. For each subsequent frame, during the vertical blanking interval, re-write the sprite pointers.

Note

If Sprite DMA is disabled while a sprite is still being displayed (between VSTART and VSTOP) the system will continue to display the last row fetched causing a bar to be displayed on screen. It is advised that Sprite DMA should only be disabled during a vertical blank or when you are sure that absolutely no Sprite data is being displayed.

Selecting a DMA Channel and Setting Pointers

When selecting a DMA channel to use with your Sprite Data you should take into consideration the colour palette mapping and the fixed sprite priority.

The Sprite DMA channel uses two pointers to read in the Sprite Data and Control Words. During the Vertical Blanking and before the first display of the Sprite you need to write the sprite's memory address to these pointers. The pointer registers are named **SPRxPTH** and **SPRxPTL** for each corresponding DMA channel *x*.

The least significant bit of **SPRxPTL** is ignored as Sprite Data must be aligned on at least a 16-bit boundary. As usual you can write a full 32-bit value into the register pair using a single move.

```
MOVE.L #$200000,SPR0PTH+CUSTOM
```

These pointers are dynamic and are updated by the DMA channel as the Sprite Data is consumed. The data fetched is loaded into the corresponding control and data registers automatically in a similar fashion to that described in the **Manual Mode** section.

Resetting the Address Pointers

As was already mentioned the address pointers are dynamic and thus for each display frame require resetting. This reset can be performed via Copper lists or as part of a Vertical Blanking routine.

- Using AGA+ 32-bit Copper Move instruction simplifies this procedure for Copperlists:

```
dc.w  $8120       ; Sprite 0 pointer
dc.l  Sprite0_Control

dc.w  $8124       ; Sprite 1 pointer
dc.l  Sprite1_Control

dc.w  $8128       ; Sprite 2 pointer
dc.l  Sprite2_Control

dc.w  $812C       ; Sprite 3 pointer
dc.l  Sprite3_Control

dc.w  $8130       ; Sprite 4 pointer
dc.l  Sprite4_Control

dc.w  $8134       ; Sprite 5 pointer
dc.l  Sprite5_Control

dc.w  $8138       ; Sprite 6 pointer
dc.l  Sprite6_Control

dc.w  $813C       ; Sprite 7 pointer
dc.l  Sprite7_Control

dc.w  $8320       ; Sprite 8 pointer
dc.l  Sprite8_Control
```

Creating Additional Sprites

To use additional sprites, the above structures must be created per channel and the respective pointer registers loaded (SPR1PTH, SPR1PTL, SPR2PTH, SPR2PTL etc).

Enabling DMA for a sprite will automatically enable DMA for all sprites and place them in automatic mode. Thus, you do not need to repeat this step per sprite.

Once DMA is enabled all sprite pointers *must* be initialized to valid sprite data structures or point to a safe NULL sprite.

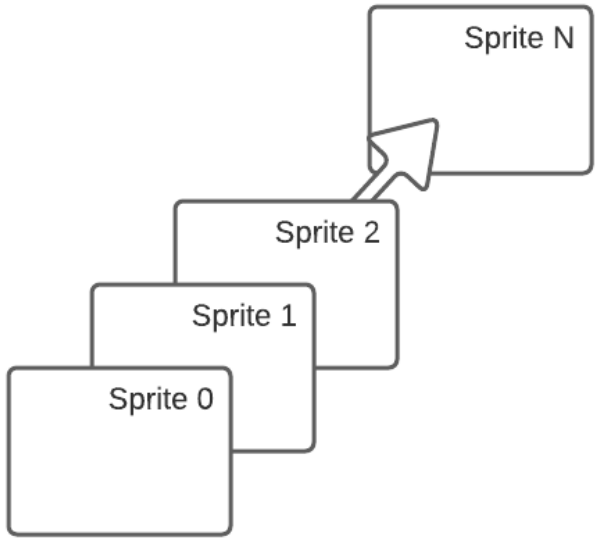
Moving a Sprite

Sprites generated in automatic mode can be made to move by simply updating their position values in the data structure. For each display frame this data is re-read and the sprite drawn at the new position. Moving sprites should only be performed when Sprite DMA is not actively reading the data as this may cause unexpected visual glitches. As with resetting address pointers these updates are usually best performed by a Vertical Blanking routine or with the Copper.

As Sprites move about the screen, they can collide with each other or with either of the two playfields. You can use the System hardware to detect these collisions. This feature is described in the next chapter.

Sprite Priorities

Sprites have a fixed priority relative to each other with Sprite 0 being displayed in front of the others. This applies to both original AGA (8 sprite) and AGA+ (16 sprite) modes.

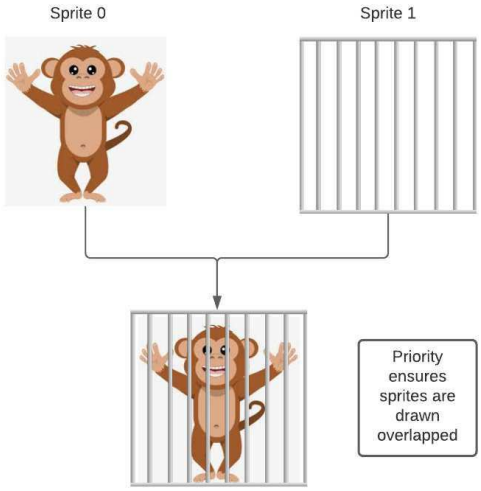


Sprite priority with regards to playfields is described in the Playfield Hardware chapter.

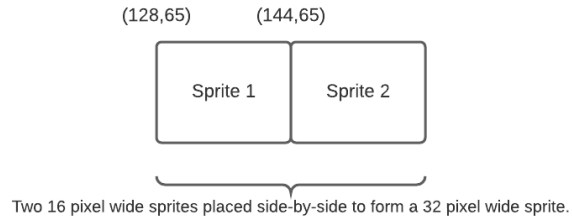
Sprite Overlays and Wider Sprites

Due to the fixed sprite priority it is possible to simply draw a lowered numbered sprite on-top of another one to create more complex objects. To draw a sprite that is larger than the current fixed side is as simple as using two Sprite DMA channels to place the sprites horizontally adjacent to each other.

Simple Overlap (Non-Attached)

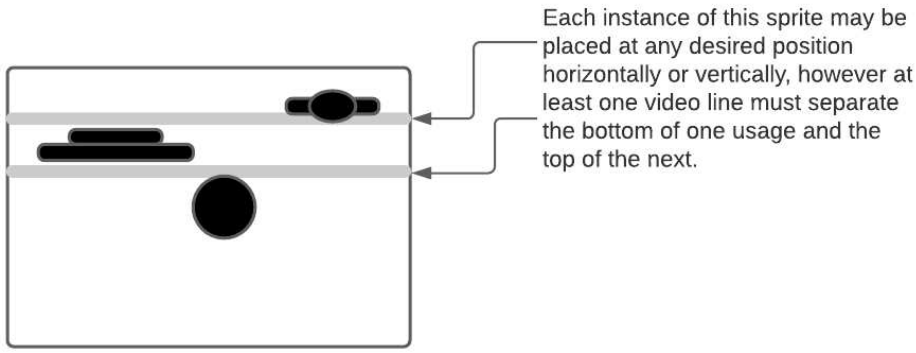


Side-By-Side Drawing



Reusing Sprite DMA Channels

Each of the Sprite DMA channels can produce more than one independently controllable image with the only restriction that the re-use must occur vertically.



Note

AGA+ operation removes this restriction by performing a “double-fetch” on the last row of the sprite. This allows the re-use to occur on the immediately adjacent row.

In normal operation a pair of zero-words are placed at the end of the Sprite Data to prevent the DMA channel from fetching any more data. By replacing these words with a completely new sprite structure including the controls words and positional data the same channel can re-instantiate another sprite at a lower position on the screen.

Attached Sprites

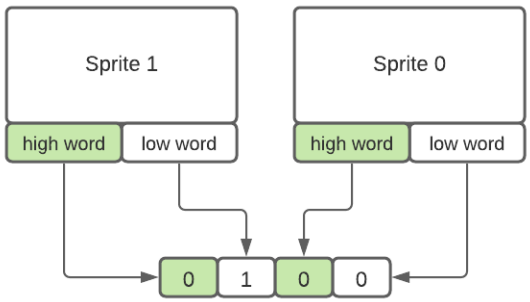
For OCS/ECS and AGA systems a pair of sprites can be “attached” to allow for a palette of 15+1 colours. To attach a sprite, you must:

- Use two channels per sprite, creating two sprites of the same size and located at the same position.
- Set the **ATTACH** bit in the second sprite control word (The odd numbered sprite).

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

Data from both sprites are fetched and combined bitwise to produce a value between 0-15. These values map to colour register indices 16-31 with *0000 = 16 = transparent*. The highest numbered sprite contributes the high-order bits as follows:



Note

Attached sprites are not required in AGA+ operation as 16 colours are always available freeing up all 16 sprites for general purpose use.

Manual Mode

It is almost always best to use automatic DMA mode with Sprites. Sometimes, however, it is useful to load these registers directly from the CPU. Sprites may be activated “manually” whenever they are not being used by a DMA channel. The same sprite that is showing an image at the top of the screen using DMA can be reloaded manually there-after.

You display sprites manually by writing to the sprite data registers **SPRxDATB** and **SPRxDATA**, in that order. The order is important as writing to **SPRxDATA** “arms” the sprite to be output at the next horizontal comparison. This same data will be displayed on every row unless more data is loaded on subsequent horizontal rows, in which case complex images can be produced.

A sprite can be “disarmed” by writing to the **SPRxCTL** register. Writing to the **SPRxPOS** register allows you to manually move the sprite horizontally at any time, even during normal sprite usage.

Sprite Registers

The original registers for Sprite control are still valid in AGA+ operation for the first eight Sprites. Another bank of registers is specified for the second set of eight.

Address	Name	R/W	Description
DFF120	SPR0PTH	W	Sprite 0 pointer (high 5 bits – was 3)
DFF122	SPR0PTL	W	Sprite 0 pointer (low 15 bits)
DFF124	SPR1PTH	W	Sprite 1 pointer (high 5 bits – was 3)
DFF126	SPR1PTL	W	Sprite 1 pointer (low 15 bits)
DFF128	SPR2PTH	W	Sprite 2 pointer (high 5 bits – was 3)
DFF12A	SPR2PTL	W	Sprite 2 pointer (low 15 bits)
DFF12C	SPR3PTH	W	Sprite 3 pointer (high 5 bits – was 3)
DFF12E	SPR3PTL	W	Sprite 3 pointer (low 15 bits)
DFF130	SPR4PTH	W	Sprite 4 pointer (high 5 bits – was 3)
DFF132	SPR4PTL	W	Sprite 4 pointer (low 15 bits)
DFF134	SPR5PTH	W	Sprite 5 pointer (high 5 bits – was 3)
DFF136	SPR5PTL	W	Sprite 5 pointer (low 15 bits)
DFF138	SPR6PTH	W	Sprite 6 pointer (high 5 bits – was 3)
DFF13A	SPR6PTL	W	Sprite 6 pointer (low 15 bits)
DFF13C	SPR7PTH	W	Sprite 7 pointer (high 5 bits – was 3)
DFF13E	SPR7PTL	W	Sprite 7 pointer (low 15 bits)
DDF140	SPR0POS	W	Sprite 0 Position
DDF142	SPR0CTL	W	Sprite 0 Control
DDF144	SPR0DATA	W	Sprite 0 Data A
DDF146	SPR0DATB	W	Sprite 0 Data B

**The register rows highlighted in green are repeated for each Sprite channel and follow on sequentially.*

AGA+ Extended Registers

Address	Name	R/W	Description
DFF320	SPR8PTH	W	Sprite 8 pointer (high 5 bits – was 3)
DFF322	SPR8PTL	W	Sprite 8 pointer (low 15 bits)
DFF324	SPR9PTH	W	Sprite 9 pointer (high 5 bits – was 3)
DFF326	SPR9PTL	W	Sprite 9 pointer (low 15 bits)
DFF328	SPRAPTH	W	Sprite 10 pointer (high 5 bits – was 3)
DFF32A	SPRAPTL	W	Sprite 10 pointer (low 15 bits)
DFF32C	SPRBPTH	W	Sprite 11 pointer (high 5 bits – was 3)
DFF32E	SPRBPTL	W	Sprite 11 pointer (low 15 bits)
DFF330	SPRCPTH	W	Sprite 12 pointer (high 5 bits – was 3)
DFF332	SPRCPTL	W	Sprite 12 pointer (low 15 bits)
DFF334	SPRDPTH	W	Sprite 13 pointer (high 5 bits – was 3)
DFF336	SPRDPTL	W	Sprite 13 pointer (low 15 bits)
DFF338	SPREPTH	W	Sprite 14 pointer (high 5 bits – was 3)
DFF33A	SPREPTL	W	Sprite 14 pointer (low 15 bits)
DFF33C	SPRFPTH	W	Sprite 15 pointer (high 5 bits – was 3)
DFF33E	SPRFPTL	W	Sprite 15 pointer (low 15 bits)
DFF340	SPR8POS	W	Sprite 8 Position
DFF342	SPR8CTL	W	Sprite 8 Control
DFF344	SPR8DATA	W	Sprite 8 Data A
DFF346	SPR8DATB	W	Sprite 8 Data B

**The register rows highlighted in green are repeated for each Sprite channel and follow on sequentially.*

All registers are grouped per sprite and are functionality identical, thus for brevity we shall describe the registers for Sprite 0 only.

- Registers **DFF120** and **DFF122** contain the 20-bit memory address (2MB Chip Memory Limit for classic machines) or the full 32-bit address for AGA+.

- **SPRxDAT** registers buffer sprite image data. They are usually loaded by the Sprite DMA channel but may be loaded manually at any time. When a horizontal coincidence occurs, the buffers are dumped into shift registers are serially outputted to the display, MSB first on the left.

  NOTE: Writing the A buffer enables (arms) the sprite. Writing the **SPRxCTL** register disables the sprite. If enabled data in the A and B buffers will be output whenever the beam counter equals the sprites horizontal position value **SPRxPOS**. DATB bits are the 2SB (worth 2) for the colour registers. DATA bits are LSBs of the pixels.

SPRxPOS		
Bit	Name	Function
15-08	SV7 - SV0	Start vertical value. High bit (SV8) is in SPRxCTL register.
07-00	SH10 - SH3	Sprite horizontal start value. Low order 3 bits (SH0-SH2) are in SPRxCTL register. If SSCAN2 bit is enabled in FMODE register, SH10 is then disabled and free for Alice to use as an individual scan-double enable.

SPRxCTL		
Bit	Name	Function
15-08	EV7 - EV0	End vertical value. Low 8 bits.
07	ATT	Sprite attach control bit (odd numbered sprites only).
06	SV9	Start vertical value bit 10.
05	EV9	End vertical value bit 10.
04	SH1=0	Start horizontal value, 70ns increment
03	SH0=0	Start horizontal value, 35ns increment
02	SV8	Start vertical value bit 9.
01	EV8	End vertical value bit 9.
00	SH2	Start horizontal value, 140ns increment

These two registers work together as position, size and feature controls for a sprite. They are usually loaded automatically by Sprite DMA during horizontal blanking, but can be manually programmed at any time. Writing to SPRxCTL will disable the sprite.

Additional Information

The following link contains a detailed analysis of a number of clever tricks implemented with sprites in a number of classic Amiga games:

https://codetapper.com/amiga/sprite-tricks/