General introduction to sprites

Sprites are freely movable 24×21 hires pixel sized objects. There are 8 of them, while it is possible to display more than 8, the rule of thumb is that *it is not possible display more than 8 on the same rasterline.*

Sprites have the following attributes:

- Display Enabled/Disabled
- X,Y position
- Multicolor/Hires mode
- One individual color/sprite
- In multicolor mode you have 2 colors that are the same for all sprites and one individual color per sprite
- you can stretch the sprites horizontally or vertically
- the shape of the sprite, ie you can control in 64 byte steps which memory area is displayed in each sprite.
- Sprite/Background Priority
- Sprite/Sprite collision detection
- Sprite/Background collision detection

It is important to note that sprites are totally independent of the gfx mode.

Turning Display of Sprites on/off

Each bit in $d015 controls whether the corresponding sprite is displayed or not. The lowest bit controls Sprite0 and so on.

Coordinates

- $d000 controls sprite0's X position
- $d001 controls sprite0's Y position
- $d002 sprite1 X pos
- $d003 sprite1 Y pos
- etc

The X coordinates are 9 bits long, and the 9th bit of all sprites are collected at $d010, similar to how $d015 works.

Colors

Each sprite has one individual color, the registers to set them are the following:

- $d027 sprite0's color
- $d028 sprite1's color
- etc

$d025 and $d026 defines the 2 colors that are shared amongst all sprites in multicolor mode

Multicolor/Hires mode

Each bit in $d01c controls whether the corresponding sprite is displayed in multicolor mode or not. The lowest bit controls Sprite0 and so on.

In Hires mode each '1' bit will be colored with the Sprite's individual Color. Bits '0' will be transparent.

In Multicolor mode the resolution halves, and each 2 bit represents a pixel.

- bitpair '00' = transparent
- bitpair '01' = color defined at $d025
- bitpair '11' = color defined at $d026
- bitpair '10' = sprite's individual color defined at $d027-$d02f

Stretching

This works by simpling doubling the pixel sizes either vertically or horizontally.

- $d017 controls wether the sprite should be stretched *vertically*, works like $d015
- $d01d controls wether the sprite should be stretched *horizontally*, works like $d015

Sprite Shape / animation

You can have a maximum of 256 sprite gfx definitions in one VIC bank. That which one is displayed in a given sprite is controlled by the last 8 bytes of the displayed screen memory. Each sprite definition takes up 64 bytes, so the formula to calculate the memory area holding the sprite gfx is: sprite pointer*64.

- screenmem+$03f8 = sprite0's gfx pointer
- screenmem+$03f9 = sprite1's gfx pointer
- screenmem+$03fa = sprite2's gfx pointer
- etc

Priority

(copy paste from the Vic Article)

As soon as several graphics elements (sprites and text/bitmap graphics) overlap on the screen, it has to be decided which element is displayed in the foreground. To do this, every element has a priority assigned and only the element with highest priority is displayed.

The sprites have a rigid hierarchy among themselves: Sprite 0 has the highest and sprite 7 the lowest priority. If two sprites overlap, the sprite with the higher number is displayed only where the other sprite has a transparent pixel.

The priority of the sprites to the text/bitmap graphics can be controlled within some limits. First of all, you have to distinguish the text/bitmap graphics between foreground and background pixels. Which bit combinations belong to the foreground or background is decided by the MCM bit in register $d016 independently of the state of the graphics data sequencer and of the BMM and ECM bits in register $d011:

```
            | MCM=0 |   MCM=1
------------+-------+-----------
Bits/pixel  |   1   |     2
Pixels/byte |   8   |     4
Background  |  "0"  | "00", "01"
Foreground  |  "1"  | "10", "11"
```

In multicolor mode (MCM=1), the bit combinations “00” and “01” belong to the background and “10” and “11” to the foreground whereas in standard mode (MCM=0), cleared pixels belong to the background and set pixels to the foreground. It should be noted that this is also valid for the graphics generated in idle state.

With the MxDP bits from register $d01b, you can separately specify for each sprite if it should be displayed in front of or behind the foreground pixels (the table in [2] is wrong):

```
MxDP=0:

      +-----------------------+
      |  Background graphics  |  low priority
    +-----------------------+ |
    |  Foreground graphics  |-+
  +-----------------------+ |
  |       Sprite x        |-+
+-----------------------+ |
|     Screen border     |-+
|                       |   high priority
+-----------------------+

MxDP=1:

      +-----------------------+
      |  Background graphics  |  low priority
    +-----------------------+ |
    |       Sprite x        |-+
  +-----------------------+ |
  |  Foreground graphics  |-+
+-----------------------+ |
```

```
 |     Screen border     |-+
 |                       |   high priority
 +-----------------------+
```

Of course, the graphics elements with lower priority than an overlayed sprite are visible where the sprite has a transparent pixel.

Collision Detection

(copy paste from the Vic Article)

Sprite/Sprite

Collision of sprites among themselves is detected as soon as two or more sprite data sequencers output a non-transparent pixel in the course of display generation (this can also happen somewhere outside of the visible screen area). In this case, the MxM bits of all affected sprites are set in register $d01e and (if allowed), an interrupt is generated. The bits remain set until the register is read by the processor and are cleared automatically by the read access.

Sprite/Background

A collision of sprites and other graphics data is detected as soon as one or more sprite data sequencers output a non-transparent pixel and the graphics data sequencer outputs a foreground pixel in the course of display generation. In this case, the MxD bits of the affected sprites are set in register $d01f and (if allowed), an interrupt is generated. As with the sprite-sprite collision, the bits remain set until the register is read by the processor.

Enabling the VIC to generate an interrupt when a collision happens

(from AAY64)

```
$D01A/53274/VIC+26:  Interrupt Mask Register (IMR)

   1 = IRQ enabled

   +----------+-------------------------------------------------------+
   | Bit 7-4  |   Always 1                                            |
   | Bit 3    |   Light-Pen Triggered IRQ Flag                        |
   | Bit 2    |   Sprite to Sprite Collision IRQ Flag     (see $D01E) |
   | Bit 1    |   Sprite to Background Collision IRQ Flag (see $D01F) |
   | Bit 0    |   Raster Compare IRQ Flag                 (see $D012) |
```

```
+----------+-------------------------------------------------------+

An IRQ will be initiated, if equal bits are set in IRR and IMR.
Default Value: $00/0 (%00000000).
```