VERA Programmer's Reference

Version 0.9

Author: Frank van den Hoef

This is preliminary documentation and the specification can still change at any point.

This document describes the **V**ersatile **E**mbedded **R**etro **A**dapter or VERA. The VERA consists of:

- Video generator featuring:
  - Multiple output formats (VGA, NTSC Composite, NTSC S-Video, RGB video) at a fixed resolution of 640x480@60Hz
  - Support for 2 layers, both supporting either tile or bitmap mode.
  - Support for up to 128 sprites.
  - Embedded video RAM of 128kB.
  - Palette with 256 colors selected from a total range of 4096 colors.
- 16-channel Programmable Sound Generator with multiple waveforms (Pulse, Sawtooth, Triangle, Noise)
- High quality PCM audio playback from an 4kB FIFO buffer featuring up to 48kHz 16-bit stereo sound.
- SPI controller for SecureDigital storage.

Registers

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F20	ADDRx_L (x=ADDRSEL)	VRAM Address (7:0)							
$9F21	ADDRx_M (x=ADDRSEL)	VRAM Address (15:8)							
$9F22	ADDRx_H (x=ADDRSEL)	Address Increment				DECR	Nibble Increment	Nibble Address	VRAM Address (16)
$9F23	DATA0	VRAM Data port 0							
$9F24	DATA1	VRAM Data port 1							
$9F25	CTRL	Reset	DCSEL						ADDRSEL
$9F26	IEN	IRQ line (8)	Scan line (8)	-		AFLOW	SPRCOL	LINE	VSYNC
$9F27	ISR	Sprite collisions				AFLOW	SPRCOL	LINE	VSYNC
$9F28	IRQLINE_L (Write only)	IRQ line (7:0)							
$9F28	SCANLINE_L (Read only)	Scan line (7:0)							
$9F29	DC_VIDEO (DCSEL=0)	Current Field	Sprites Enable	Layer1 Enable	Layer0 Enable	NTSC/RGB: 240P	NTSC: Chroma Disable / RGB: HV Sync	Output Mode	
$9F2A	DC_HSCALE (DCSEL=0)	Active Display H-Scale							
$9F2B	DC_VSCALE (DCSEL=0)	Active Display V-Scale							
$9F2C	DC_BORDER (DCSEL=0)	Border Color							

$9F29	DC_HSTART (DCSEL=1)	Active Display H-Start (9:2)							
$9F2A	DC_HSTOP (DCSEL=1)	Active Display H-Stop (9:2)							
$9F2B	DC_VSTART (DCSEL=1)	Active Display V-Start (8:1)							
$9F2C	DC_VSTOP (DCSEL=1)	Active Display V-Stop (8:1)							
$9F29	FX_CTRL (DCSEL=2)	Transp. Writes	Cache Write Enable	Cache Fill Enable	One-byte Cache Cycling	16-bit Hop	4-bit Mode	Addr1 Mode	
$9F2A	FX_TILEBASE (DCSEL=2) (Write only)	FX Tile Base Address (16:11)						Affine Clip Enable	2-bit Polygon
$9F2B	FX_MAPBASE (DCSEL=2) (Write only)	FX Map Base Address (16:11)						Map Size	
$9F2C	FX_MULT (DCSEL=2) (Write only)	Reset Accum.	Accumulate	Subtract Enable	Multiplier Enable	Cache Byte Index		Cache Nibble Index	Two-byte Cache Incr. Mode
$9F29	FX_X_INCR_L (DCSEL=3) (Write only)	X Increment (-2:-9) (signed)							
$9F2A	FX_X_INCR_H (DCSEL=3) (Write only)	X Incr. 32x	X Increment (5:-1) (signed)						
$9F2B	FX_Y_INCR_L (DCSEL=3) (Write only)	Y/X2 Increment (-2:-9) (signed)							
$9F2C	FX_Y_INCR_H (DCSEL=3) (Write only)	Y/X2 Incr. 32x	Y/X2 Increment (5:-1) (signed)						
$9F29	FX_X_POS_L (DCSEL=4) (Write only)	X Position (7:0)							
$9F2A	FX_X_POS_H (DCSEL=4) (Write only)	X Pos. (-9)	-				X Position (10:8)		
$9F2B	FX_Y_POS_L (DCSEL=4) (Write only)	Y/X2 Position (7:0)							
$9F2C	FX_Y_POS_H (DCSEL=4) (Write only)	Y/X2 Pos. (-9)	-				Y/X2 Position (10:8)		
$9F29	FX_X_POS_S (DCSEL=5) (Write only)	X Postion (-1:-8)							
$9F2A	FX_Y_POS_S (DCSEL=5)	Y/X2 Postion (-1:-8)							

	(Write only)								
$9F2B	FX_POLY_FILL_L (DCSEL=5, 4-bit Mode=0) (Read only)	Fill Len >= 16	X Position (1:0)		Fill Len (3:0)				0
$9F2B	FX_POLY_FILL_L (DCSEL=5, 4-bit Mode=1, 2-bit Polygon=0) (Read only)	Fill Len >= 8	X Position (1:0)		X Pos. (2)	Fill Len (2:0)			0
$9F2B	FX_POLY_FILL_L (DCSEL=5, 4-bit Mode=1, 2-bit Polygon=1) (Read only)	X2 Pos. (-1)	X Position (1:0)		X Pos. (2)	Fill Len (2:0)			X Pos. (-1)
$9F2C	FX_POLY_FILL_H (DCSEL=5) (Read only)	Fill Len (9:3)							0
$9F29	FX_CACHE_L (DCSEL=6) (Write only)	Cache (7:0) | Multiplicand (7:0) (signed)							
$9F29	FX_ACCUM_RESET (DCSEL=6) (Read only)	Reset Accumulator							
$9F2A	FX_CACHE_M (DCSEL=6) (Write only)	Cache (15:8) | Multiplicand (15:8) (signed)							
$9F2A	FX_ACCUM (DCSEL=6) (Read only)	Accumulate							
$9F2B	FX_CACHE_H (DCSEL=6) (Write only)	Cache (23:16) | Multiplier (7:0) (signed)							
$9F2C	FX_CACHE_U (DCSEL=6) (Write only)	Cache (31:24) | Multiplier (15:8) (signed)							
$9F29	DC_VER0 (DCSEL=63) (Read only)	The ASCII character "V"							
$9F2A	DC_VER1 (DCSEL=63) (Read only)	Major release							
$9F2B	DC_VER2 (DCSEL=63) (Read only)	Minor release							
$9F2C	DC_VER3 (DCSEL=63) (Read only)	Minor build number							
$9F2D	L0_CONFIG	Map Height		Map Width		T256C	Bitmap Mode	Color Depth	

$9F2E	L0_MAPBASE	Map Base Address (16:9)							
$9F2F	L0_TILEBASE	Tile Base Address (16:11)						Tile Height	Tile Width
$9F30	L0_HSCROLL_L	H-Scroll (7:0)							
$9F31	L0_HSCROLL_H	-				H-Scroll (11:8)			
$9F32	L0_VSCROLL_L	V-Scroll (7:0)							
$9F33	L0_VSCROLL_H	-				V-Scroll (11:8)			
$9F34	L1_CONFIG	Map Height		Map Width		T256C	Bitmap Mode	Color Depth	
$9F35	L1_MAPBASE	Map Base Address (16:9)							
$9F36	L1_TILEBASE	Tile Base Address (16:11)						Tile Height	Tile Width
$9F37	L1_HSCROLL_L	H-Scroll (7:0)							
$9F38	L1_HSCROLL_H	-				H-Scroll (11:8)			
$9F39	L1_VSCROLL_L	V-Scroll (7:0)							
$9F3A	L1_VSCROLL_H	-				V-Scroll (11:8)			
$9F3B	AUDIO_CTRL	FIFO Full / FIFO Reset	FIFO Empty (read-only)	16-Bit	Stereo	PCM Volume			
		FIFO Loop (write-only)							
$9F3C	AUDIO_RATE	PCM Sample Rate							
$9F3D	AUDIO_DATA	Audio FIFO data (write-only)							
$9F3E	SPI_DATA	Data							
$9F3F	SPI_CTRL	Busy	-					Slow clock	Select

VRAM address space layout

Address range	Description
$00000 - $1F9BF	Video RAM
$1F9C0 - $1F9FF	PSG registers
$1FA00 - $1FBFF	Palette
$1FC00 - $1FFFF	Sprite attributes

Important note: Video RAM locations 1F9C0-1FFFF contain registers for the PSG/Palette/Sprite attributes. Reading anywhere in VRAM will always read back the 128kB VRAM itself (not the contents of the (write-only) PSG/Palette/Sprite attribute registers). Writing to a location in the register area will write to the registers in addition to writing the value also to VRAM. Since the VRAM contains random values at startup the values read back in the register area will not correspond to the actual values in the write-only registers until they are written to once. Because of this it is highly recommended to initialize the area from 1F9C0-1FFFF at startup.

Video RAM access

The video RAM (VRAM) isn't directly accessible on the CPU bus. VERA only exposes an address space of 32 bytes to the CPU as described in the section Registers . To access the VRAM (which is 128kB in size) an indirection mechanism is used. First the address to be accessed needs to be set (ADDRx_L/ADDRx_M/ADDRx_H) and then the data on that VRAM address can be read from or written to via the DATA0/1 register. To make accessing the VRAM more efficient an auto-increment mechanism is present.

There are 2 data ports to the VRAM. Which can be accessed using DATA0 and DATA1. The address and increment associated with the data port is specified in ADDRx_L/ADDRx_M/ADDRx_H. These 3 registers are multiplexed using the ADDR_SEL in the CTRL register. When ADDR_SEL = 0, ADDRx_L/ADDRx_M/ADDRx_H become ADDR0_L/ADDR0_M/ADDR0_H.
When ADDR_SEL = 1, ADDRx_L/ADDRx_M/ADDRx_H become ADDR1_L/ADDR1_M/ADDR1_H.

By setting the 'Address Increment' field in ADDRx_H, the address will be increment after each access to the data register. The increment register values and corresponding increment amounts are shown in the following table:

Register value	Increment amount
0	0
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	256
10	512
11	40
12	80
13	160
14	320
15	640

Setting the **DECR** bit, will decrement instead of increment by the value set by the 'Address Increment' field.

Reset

When **RESET** in **CTRL** is set to 1, the FPGA will reconfigure itself. All registers will be reset. The palette RAM will be set to its default values.

Interrupts

Interrupts will be generated for the interrupt sources set in the lower 4 bits of **IEN**. **ISR** will indicate the interrupts that have occurred. Writing a 1 to one of the lower 3 bits in **ISR** will clear that interrupt status. **AFLOW** can only be cleared by filling the audio FIFO for at least 1/4.

IRQ_LINE (write-only) specifies at which line the **LINE** interrupt will be generated. Note that bit 8 of this value is present in the **IEN** register. For interlaced modes the interrupt will be generated each field and the bit 0 of **IRQ_LINE** is ignored.

SCANLINE (read-only) indicates the current scanline being sent to the screen. Bit 8 of this value is present in the **IEN** register. The value is 0 during the first visible line and 479 during the last. This value continues to count beyond the last visible line, but returns $1FF for lines 512-524 that are beyond its 9-bit resolution. **SCANLINE** is not affected by interlaced modes and will return either all even or all odd values during an even or odd field, respectively. Note that VERA renders lines ahead of scanout such that line 1 is being rendered while line 0 is being scanned out. Visible changes may be delayed one scanline because of this.

The upper 4 (read-only) bits of the **ISR** register contain the sprite collisions as determined by the sprite renderer.

Display composer

The display composer is responsible of combining the output of the 2 layer renderers and the sprite renderer into the image that is sent to the video output.

The video output mode can be selected using OUT_MODE in DC_VIDEO.

OUT_MODE	Description
0	Video disabled
1	VGA output
2	NTSC (composite/S-Video)
3	RGB 15KHz, composite or separate H/V sync, via VGA connector

Setting **'Chroma Disable'** disables output of chroma in NTSC composite mode and will give a better picture on a monochrome display. *(Setting this bit will also disable the chroma output on the S-video output.)*

Setting **'HV Sync'** enables separate HSync/VSync signals in RGB output mode. Clearing the bit will enable the default of composite sync over RGB.

Setting **'240P'** enables 240P progressive mode over NTSC or RGB. It has no effect if the VGA output mode is active. Instead of 262.5 scanlines per field, this mode outputs 263 scanlines per field. On CRT displays, the scanlines from both the even and odd fields will be displayed on even scanlines.

'Current Field' is a read-only bit which reflects the active interlaced field in composite and RGB modes. In non-interlaced modes, this reflects if the current line is even or odd. (0: even, 1: odd)

Setting **'Layer0 Enable'** / **'Layer1 Enable'** / **'Sprites Enable'** will respectively enable output from layer0 / layer1 and the sprites renderer.

DC_HSCALE and **DC_VSCALE** will set the fractional scaling factor of the active part of the display. Setting this value to 128 will output 1 output pixel for every input pixel. Setting this to 64 will output 2 output pixels for every input pixel.

DC_BORDER determines the palette index which is used for the non-active area of the screen.

DC_HSTART/**DC_HSTOP** and **DC_VSTART**/**DC_VSTOP** determines the active part of the screen. The values here are specified in the native 640x480 display space. HSTART=0, HSTOP=640, VSTART=0, VSTOP=480 will set the active area to the full resolution. Note that the lower 2 bits of **DC_HSTART**/**DC_HSTOP** and the lower 1 bit of **DC_VSTART**/**DC_VSTOP** isn't available. This means that horizontally the start and stop values can be set at a multiple of 4 pixels, vertically at a multiple of 2 pixels.

DC_VER0, **DC_VER1**, **DC_VER2**, and **DC_VER3** can be queried for the version number of the VERA bitstream. If reading **DC_VER0** returns `$56`, the remaining registers returns values forming the major, minor, and build numbers respectively. If **DC_VER0** returns a value other than `$56`, the VERA bitstream version number is undefined.

Layer 0/1 registers

'Map Base Address' specifies the base address of the tile map. *Note that the register only specifies bits 16:9 of the address, so the address is always aligned to a multiple of 512 bytes.*

'Tile Base Address' specifies the base address of the tile data. *Note that the register only specifies bits 16:11 of the address, so the address is always aligned to a multiple of 2048 bytes.*

'H-Scroll' specifies the horizontal scroll offset. A value between 0 and 4095 can be used. Increasing the value will cause the picture to move left, decreasing will cause the picture to move right.

'V-Scroll' specifies the vertical scroll offset. A value between 0 and 4095 can be used. Increasing the value will cause the picture to move up, decreasing will cause the picture to move down.

'Map Width', **'Map Height'** specify the dimensions of the tile map:

Value	Map width / height
0	32 tiles
1	64 tiles
2	128 tiles

3	256 tiles

'Tile Width', **'Tile Height'** specify the dimensions of a single tile:

Value	Tile width / height
0	8 pixels
1	16 pixels

In bitmap modes, the **'H-Scroll (11:8)'** register is used to specify the palette offset for the bitmap.

Layer display modes

The features of the 2 layers are the same. Each layer supports a few different modes which are specified using **T256C** / **'Bitmap Mode'** / **'Color Depth'** in Lx_CONFIG.

'Color Depth' specifies the number of bits used per pixel to encode color information:

Color Depth	Description
0	1 bpp
1	2 bpp
2	4 bpp
3	8 bpp

The layer can either operate in tile mode or bitmap mode. This is selected using the **'Bitmap Mode'** bit; 0 selects tile mode, 1 selects bitmap mode.

The handling of 1 bpp tile mode is different from the other tile modes. Depending on the **T256C** bit the tiles use either a 16-color foreground and background color or a 256-color foreground color. Other modes ignore the **T256C** bit.

Tile mode 1 bpp (16 color text mode)

T256C should be set to 0.

MAP_BASE points to a tile map containing tile map entries, which are 2 bytes each:

Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Character index							
1	Background color				Foreground color			

TILE_BASE points to the tile data.

Each bit in the tile data specifies one pixel. If the bit is set the foreground color as specified in the map data is used, otherwise the background color as specified in the map data is used.

Tile mode 1 bpp (256 color text mode)

T256C should be set to 1.

MAP_BASE points to a tile map containing tile map entries, which are 2 bytes each:

Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Character index							
1	Foreground color							

TILE_BASE points to the tile data.

Each bit in the tile data specifies one pixel. If the bit is set the foreground color as specified in the map data is used, otherwise color 0 is used (transparent).

Tile mode 2/4/8 bpp

MAP_BASE points to a tile map containing tile map entries, which are 2 bytes each:

Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Tile index (7:0)							
1	Palette offset				V-flip	H-flip	Tile index (9:8)	

TILE_BASE points to the tile data.

Each pixel in the tile data gives a color index of either 0-3 (2bpp), 0-15 (4bpp), 0-255 (8bpp). This color index is modified by the palette offset in the tile map data using the following logic:

- Color index 0 (transparent) and 16-255 are unmodified.
- Color index 1-15 is modified by adding 16 x palette offset.

Bitmap mode 1/2/4/8 bpp

MAP_BASE isn’t used in these modes. **TILE_BASE** points to the bitmap data.

TILEW specifies the bitmap width. TILEW=0 results in 320 pixels width and TILEW=1 results in 640 pixels width.

The palette offset (in **'H-Scroll (11:8)'**) modifies the color indexes of the bitmap in the same way as in the tile modes.

SPI controller

The SPI controller is connected to the SD card connector. The speed of the clock output of the SPI controller can be controlled by the **'Slow Clock'** bit. When this bit is 0 the clock is 12.5MHz, when 1 the clock is about 390kHz. The slow clock speed is to be used during the initialization phase of the SD card. Some SD cards require a clock less than 400kHz during part of the initialization.

A transfer can be started by writing to **SPI_DATA**. While the transfer is in progress the BUSY bit will be set. After the transfer is done, the result can be read from the **SPI_DATA** register.

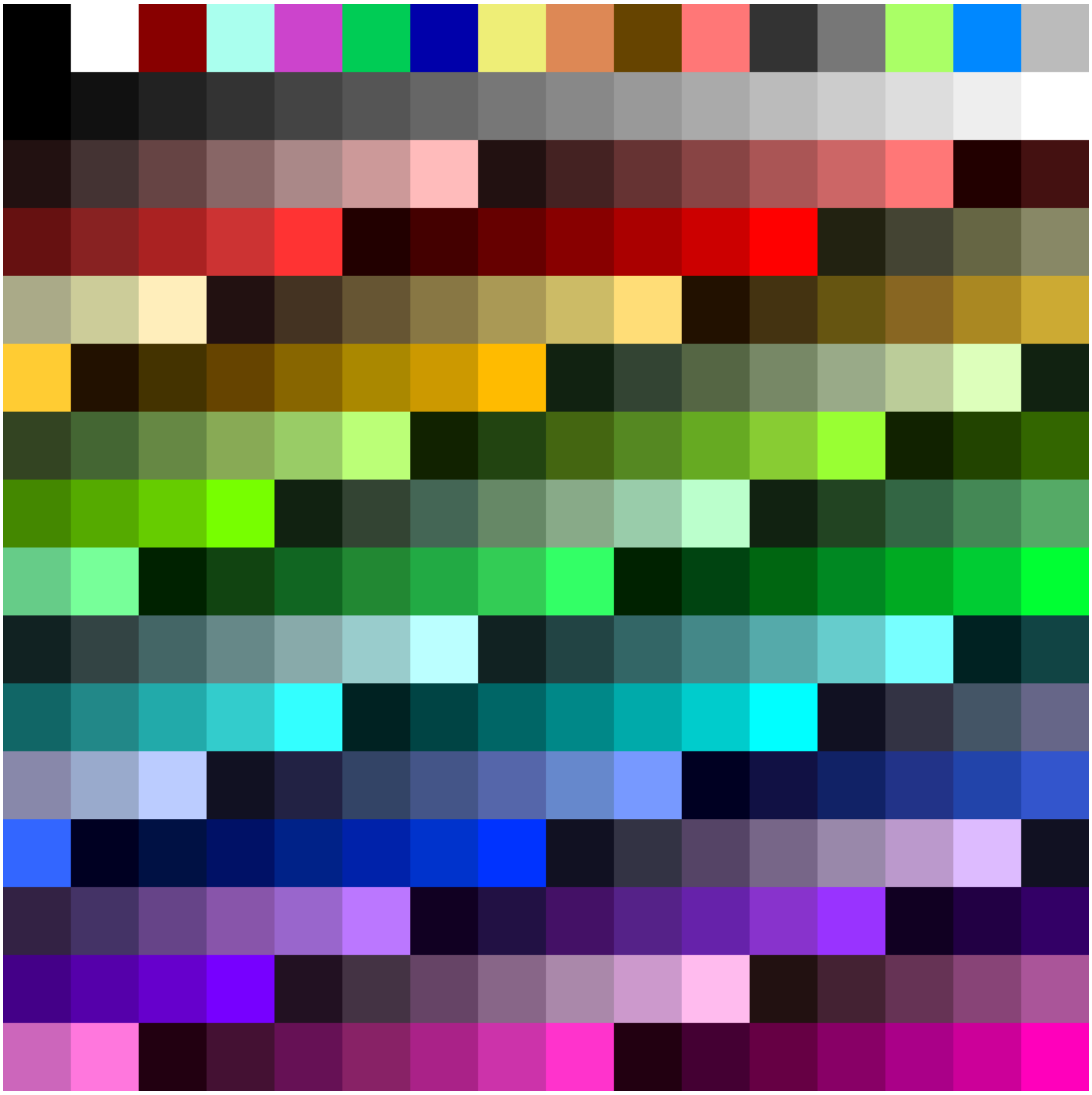
The chip select can be controlled by writing the **SELECT** bit. Writing 1 will assert the chip-select (logic-0) and writing 0 will release the chip-select (logic-1).

Palette

The palette translates 8-bit color indexes into 12-bit output colors. The palette has 256 entries, each with the following format:

Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Green				Blue			
1	-				Red			

At reset, the palette will contain a predefined palette:



- Color indexes 0-15 contain a palette somewhat similar to the C64 color palette.
- Color indexes 16-31 contain a grayscale ramp.
- Color indexes 32-255 contain various hues, saturation levels, brightness levels.

Sprite attributes

128 entries of the following format:

Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Address (12:5)							
1	Mode	-			Address (16:13)			
2	X (7:0)							

3	-						X (9:8)	
4	Y (7:0)							
5	-						Y (9:8)	
6	Collision mask				Z-depth		V-flip	H-flip
7	Sprite height		Sprite width		Palette offset			

Mode	Description
0	4 bpp
1	8 bpp

Z-depth	Description
0	Sprite disabled
1	Sprite between background and layer 0
2	Sprite between layer 0 and layer 1
3	Sprite in front of layer 1

Sprite width / height	Description
0	8 pixels
1	16 pixels
2	32 pixels
3	64 pixels

Rendering Priority The sprite memory location dictates the order in which it is rendered. The sprite whose attributes are at the lowest location will be rendered in front of all other sprites; the sprite at the highest location will be rendered behind all other sprites, and so forth.

Palette offset works in the same way as with the layers.

Sprite collisions

At the start of the vertical blank **Collisions** in **ISR** is updated. This field indicates which groups of sprites have collided. If the field is non-zero the **SPRCOL** interrupt will be set. The interrupt is generated once per field / frame and can be cleared by making sure the sprites no longer collide.

Note that collisions are only detected on lines that are actually rendered. This can result in subtle differences between non-interlaced and interlaced video modes.

VERA FX

The FX feature set is available in VERA firmware version v0.3.1 or later. The Commander X16 emulators also have this feature officially as of R44.

FX is a set of mainly addressing mode changes. VERA FX does not accelerate rendering, but it merely assists the CPU with some of the slower tasks, and when used cleverly, can allow for the programmer to perform some limited perspective transforms or basic 3D effects.

FX features are controlled mainly by registers $9F29-$9F2C with DCSEL set to 2 through 6. FX_CTRL ($9F29 w/ DCSEL=2) is the master switch for enabling or disabling FX behaviors. When writing an application that uses FX, it is important that the FX mode be preserved and disabled in interrupt handlers in cases where the handler accesses VERA registers or VRAM, including the PSG sound registers. Reading from FX_CTRL returns the current state, and writing 0 to FX_CTRL suspends the FX behaviors so that the VERA can be accessed normally without mutating other FX state.

Preliminary documentation for the feature can be found here , but as this is a brand new feature, examples and documentation still need to be written.

Programmable Sound Generator (PSG)

The audio functionality contains of 2 independent systems. The first is the PSG or Programmable Sound Generator. The second is the PCM (or Pulse-Code Modulation) playback system.

16 entries (channels) of the following format:

Offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Frequency word (7:0)							
1	Frequency word (15:8)							
2	Right	Left	Volume					
3	Waveform		Pulse width					

Frequency word sets the frequency of the sound. The formula for calculating the output frequency is:

```
sample_rate = 25MHz / 512 = 48828.125 Hz

output_frequency = sample_rate / (2^17) * frequency_word
```

Thus the output frequency can be set in steps of about 0.373 Hz.

Example: to output a frequency of 440Hz (note A4) the ***Frequency word*** *should be set to 440 / (48828.125 / (2^17)) = 1181*

Volume controls the volume of the sound with a logarithmic curve; 0 is silent, 63 is the loudest. The **Left** and **Right** bits control to which output channels the sound should be output.

Waveform controls the waveform of the sound:

Waveform	Description
0	Pulse
1	Sawtooth
2	Triangle
3	Noise

Pulse width controls the duty cycle of the pulse waveform. A value of 63 will give a 50% duty cycle or square wave, 0 will give a very narrow pulse.

Just like the other waveform types, the frequency of the noise waveform can be controlled using frequency. In this case a higher frequency will give brighter noise and a lower value will give darker noise.

PCM audio

For PCM playback, VERA contains a 4kB FIFO buffer. This buffer needs to be filled in a timely fashion by the CPU. To facilitate this an **AFLOW** (Audio FIFO low) interrupt can be generated when the FIFO is less than 1/4 filled.

Audio registers

`AUDIO_CTRL ($9F3B)`

FIFO Full (bit 7) is a read-only flag that indicates whether the FIFO is full. Any writes to the FIFO while this flag is 1 will be ignored. Writing a 1 to this register (**FIFO Reset**) will perform a FIFO reset, which will clear the contents of the FIFO buffer, except when written in combination with a 1 in bit 6.

FIFO Loop (bit 6+7): If a 1 is written to both bit 6 and 7 (at the same time), the FIFO will loop when played. Any other write to AUDIO_CTRL clears this loop flag. Note: this feature is currently only available in x16-emulator and is not in any released VERA firmware.

FIFO Empty (bit 6) is a read-only flag that indicates whether the FIFO is empty.

16-bit (bit 5) sets the data format to 16-bit. If this bit is 0, 8-bit data is expected.

Stereo (bit 4) sets the data format to stereo. If this bit is 0 (mono), the same audio data is send to both channels.

PCM Volume (bits 0..3)controls the volume of the PCM playback, this has a logarithmic curve. A value of 0 is silence, 15 is the loudest.

`AUDIO_RATE ($9F3C)`

PCM sample rate controls the speed at which samples are read from the FIFO. A few example values:

PCM sample rate	Description
128	normal speed (25MHz / 512 = 48828.125 Hz)
64	half speed (24414 Hz)
32	quarter speed (12207 Hz)
0	stop playback
>128	*invalid*

Using a value of 128 will give the best quality (lowest distortion); at this value for every output sample, an input sample from the FIFO is read. Lower values will output the same sample multiple times to the audio DAC. Input samples are always read as a complete set (being 1/2/4 bytes).

`AUDIO_DATA ($9F3D)`

Audio FIFO data Writes to this register add one byte to the PCM FIFO. If the FIFO is full, the write will be ignored.

NOTE: When setting up for PCM playback it is advised to first set the sample rate at 0 to stop playback. First fill the FIFO buffer with some initial data and then set the desired sample rate. This can prevent undesired FIFO underruns.

Audio data formats

Audio data is two's complement signed. Depending on the selected mode the data needs to be written to the FIFO in the following order:

Mode	Order in which to write data to FIFO
8-bit mono	<mono sample>
8-bit stereo	<left sample> <right sample>
16-bit mono	<mono sample (7:0)> <mono sample (15:8)>
16-bit stereo	<left sample (7:0)> <left sample (15:8)> <right sample (7:0)> <right sample (15:8)>

VERA FX Reference

Author: MooingLemur, based on documentation written by JeffreyH

This is preliminary documentation and the specification can still change at any point.

Introduction

This is a reference for the VERA FX features. It is meant to be a complement to the tutorial, currently found here .

The FX Update mainly adds "helpers" inside of VERA that can be used by the CPU. There is no "magic button" that allows you to do 3D graphics for example. It mainly helps at certain CPU time-consuming tasks, most notably the ones that are present in the (deep) inner-loop of a game/graphics engine. The FX Update does therefore not fundamentally change the architecture or nature of VERA, it extends and improves it.

In other words: the CPU is still the orchestrator of all that is done, but it is alleviated from certain operations where it is not (very) good at or does not have direct access to.

FX Update extends addressing modes, it does not add or extend renderers.

Usage

DCSEL

VERA is mapped as 32 8-bit registers in the memory space of the Commander X16, starting at address $9F20 and ending at $9F3F. Many of these are (fully) used, but some bits remain unused. The DCSEL bits in register $9F25 (also called CTRL) has been extended to 6-bits to allow for the 4 registers $9F29-$9F2C to have additional meanings.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F25	CTRL	Reset	**DCSEL**						ADDRSEL

The FX features use DCSEL values 2, 3, 4, 5, and 6. This effectively gives FX 20 8-bit registers. Note that 15 of these registers are *write-only*, 2 of them are *read-only* and 3 are both *readable* and *writable*,

Important: *unless DCSEL values of 2-6 are used, the behavior of VERA is exactly the same as it was before the FX update. This ensures that the FX update is backwards compatible with traditional non-FX uses of VERA*.

Addr1 Mode

When DCSEL=2, the main FX configuration register becomes available (FX_CTRL/$9F29), which is both readable and writable. The 2 lower bits are the addr1 mode bits, which will change the behavior of how and when ADDR1 is updated. This puts the FX helpers in a certain "role".

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	Transp. Writes	Cache Write Enable	Cache Fill Enable	One-byte Cache Cycling	16-bit Hop	4-bit Mode	**Addr1 Mode**	

Addr1 Mode	Description
0	Traditional VERA behavior
1	Line draw helper
2	Polygon filler helper
3	Affine helper

By default, Addr1 Mode is set to 0 (=00b), which is the **normal** and already-known behavior of `ADDR1` .

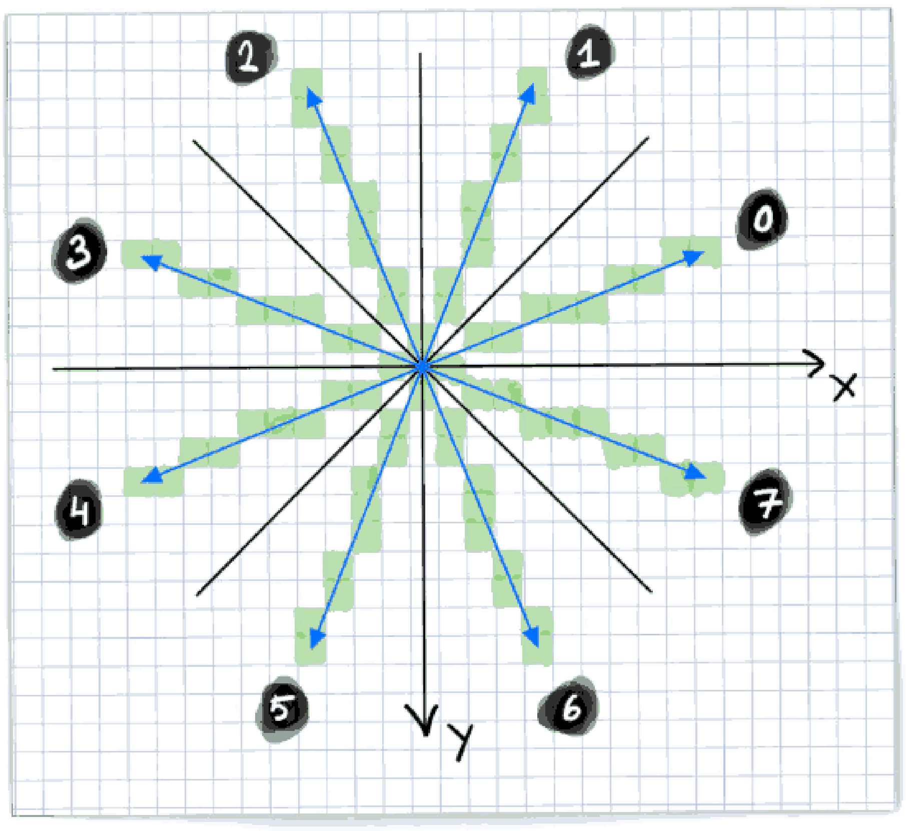
Line draw helper

When Addr1 Mode is set to 1 (=01b) the **line draw helper** is enabled.

Setting up the line draw helper

- Set `ADDR1` to the address of the starting pixel
- Determine the octant (see below) you are going to draw in, which will inform your `ADDR0` and `ADDR1` increments.
  - Set `ADDR1` increment in the direction you will ***always*** increment each step
    - For 8-bit mode: (+1, -1, -320, or +320)
    - For 4-bit mode: (-0.5, +0.5, -160, or +160)
  - Set `ADDR0` increment in the direction you will ***sometimes*** increment. Even though this is the increment for `ADDR0`, we are using it in line draw mode as an incrementer for `ADDR1`.
    - For 8-bit mode: (+1, -1, -320, or +320).
    - For 4-bit mode: (-0.5, +0.5, -160, or +160)
  - For 4-bit mode, the half increments are set via the Nibble Increment bit and optionally the DECR bit in `ADDRx_H`. For the Nibble Increment bit to have effect, the main Address Increment must be set to 0, and the 4-bit Mode bit must be set in FX_CTRL ($9F29, DCSEL=2).

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F22	ADDRx_H (x=ADDRSEL)	Address Increment				**DECR**	**Nibble Increment**	Nibble Address	VRAM Address (16)



Octant	8-bit `ADDR1` increment	8-bit `ADDR0` increment	4-bit `ADDR1` increment	4-bit `ADDR0` increment
0	+1	-320	+0.5	-160
1	-320	+1	-160	+0.5
2	-320	-1	-160	-0.5
3	-1	-320	-0.5	-160
4	-1	+320	-0.5	+160
5	+320	-1	+160	-0.5
6	+320	+1	+160	+0.5
7	+1	+320	+0.5	+160

- Set your slope into the two "X Increment" registers (DCSEL=3, see below). Note that increment registers are 15-bit signed fixed-point numbers, and for this mode, the range should be 0.0 to 1.0 inclusive, so you'll either want to store the value of 1, or you'll want to set only the fractional part.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_X_INCR_L (DCSEL=3) (Write only)	**X Increment (-2:-9) (signed)**							
$9F2A	FX_X_INCR_H (DCSEL=3) (Write only)	X Incr. 32x	X Increment (5:1) (signed)					**X Incr. (0)**	**X Incr. (-1)**

Note: Of the two incrementers, the line draw helper uses only the X incrementer. However depending on the octant you are drawing in, this incrementer will be used to depict either x or y pixel increments. So the "X" should not be taken literally here, it just means the first of the two incrementers.

- As a side effect of in line draw mode, by setting `FX_X_INCR_H` ($9F2A, DCSEL=3), the fractional part (the lower 9 bits) of *X Position* are automatically set to half a pixel. Furthermore, the lowest bit of the pixel position (which acts as an overflow bit) is set to 0 as well. This effectively sets the starting X-position to 0.5 (the center) of a pixel.

Note: There is no need to set the higher bits of the X position, since the FX X position (accumulator) is only used to track the fractional (subpixel) part of the line draw.

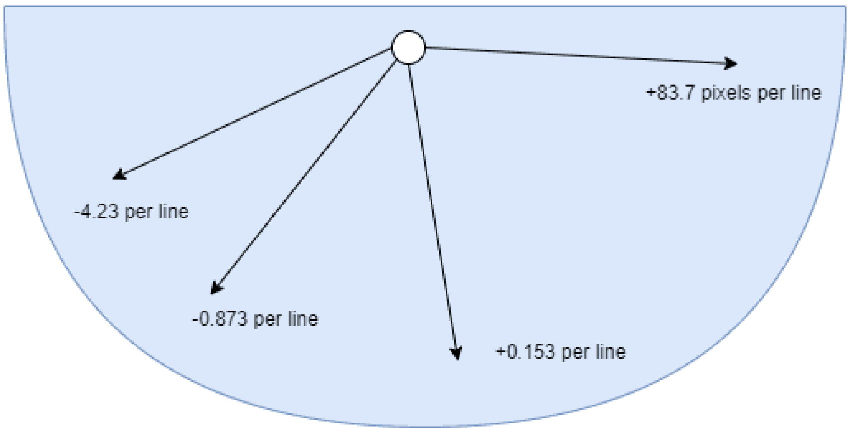
Polygon filler helper

When Addr1 Mode is set to 2 (=10b) the polygon filler helper is enabled.

Setting up the polygon filler helper

Assuming a 320 pixel-wide screen

- Set `ADDR0` to the address of the y-position of the top point of the triangle and x=0 (so on the left of the screen). Set its increment to +320 (for 8-bit mode) or +160 (for 4-bit mode).
  - Note: `ADDR0` is used as "base address" for calculating `ADDR1` for each horizontal line of the triangle. `ADDR0` should therefore start at the top of the triangle and increment exactly one line each time.
  - There is no need to set `ADDR1`. This is done by VERA.
- Calculate your slopes (dx/dy) for both the left and right point. Unlike the line draw helper, these slopes can be negative and can exceed 1.0. They are not dependent on octant, but cover the whole 180 degrees downwards. Below is an illustration of some (not-to-



scale) examples of increments:
- Set `ADDR1` increment to +1 (for 8-bit mode) or +0.5 (for 4-bit mode)
  - `ADDR1` increment can also be +4 if you use 32-bit cache writes, explained later)
- Set your left slope into the two "X increment" registers and your right slope into the two "Y increment" registers (DCSEL=3, see below).
  - Important: They should be set to half the increment (or decrement) per horizontal line! This is because the polygon filler increments in two steps per line.
- Note that increment registers are 15-bit signed fixed-point numbers:
- 6 bits for the integer pixel increment
- 9 bits for the fractional (subpixel) increment
- 1 additional bit that indicates the actual value should be multiplied by 32

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_X_INCR_L (DCSEL=3) (Write only)	X Increment (-2:-9) (signed)							
$9F2A	FX_X_INCR_H (DCSEL=3) (Write only)	X Incr. 32x	X Increment (5:0) (signed)						X Incr. (-1)
$9F2B	FX_Y_INCR_L (DCSEL=3) (Write only)	Y/X2 Increment (-2:-9) (signed)							
$9F2C	FX_Y_INCR_H (DCSEL=3) (Write only)	Y/X2 Incr. 32x	Y/X2 Increment (5:0) (signed)						Y/X2 Incr. (-1)

- Due to the fact that we are in "polygon fill"-mode, by setting the high bits of the "X increment" ($9F2A, DCSEL=3), the "X position" (the lower 9 bits of the position in DCSEL=4 and DCSEL=5) are automatically set to half a pixel. The same goes for the high bits of

the Y/X2 increment ($9F2C, DCSEL=3) and Y/X2 position.
- Set the "X position" and "Y/X2 position" to the x-pixel-position of the top triangle point.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_X_POS_L (DCSEL=4) (Write only)	**X Position (7:0)**							
$9F2A	FX_X_POS_H (DCSEL=4) (Write only)	X Pos. (-9)	-				**X Position (10:8)**		
$9F2B	FX_Y_POS_L (DCSEL=4) (Write only)	**Y/X2 Position (7:0)**							
$9F2C	FX_Y_POS_H (DCSEL=4) (Write only)	Y/X2 Pos. (-9)	-				**Y/X2 Position (10:8)**		

Steps that are needed for filling a triangle part with lines:

- Read from `DATA1`
  - This will not return any useful data but will do two things in the background:
    - Increment/decrement the X1 and X2 positions by their corresponding increment values.
    - Set `ADDR1` to `ADDR0` + X1
- Then read the “Fill length (low)”-register. Its output depends on whether you're in 4 or 8-bit mode.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2B	FX_POLY_FILL_L (DCSEL=5, 4-bit Mode=0) (Read only)	Fill Len >= 16	X Position (1:0)		Fill Len (3:0)				0
$9F2B	FX_POLY_FILL_L (DCSEL=5, 4-bit Mode=1, 2-bit Polygon=0) (Read only)	Fill Len >= 8	X Position (1:0)		X Pos. (2)	Fill Len (2:0)			0

- If fill_len >= 16 (or >= 8 in 4-bit mode) then also read the “Fill length (high)”-register:

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2C	FX_POLY_FILL_H (DCSEL=5) (Read only)	Fill Len (9:3)							0

  Important: when the two highest bits of Fill Len (bits 8 and 9) are both 1, it means there is a negative fill length. The line should not be drawn!
- Together they give you 10-bits of fill length (ignore the other bits for now). Since `ADDR1` is already set properly you can immediately start drawing this number of pixels (given by Fill Len).
  - `sta DATA1` ; as many times as Fill Len states
- Then read from `DATA0` : this will (also) increment X1 and X2
- Check if all lines of this triangle part have been drawn, if not go to the first step.

There is also a 2-bit polygon mode, which is better explained in the tutorial

Affine helper

When Addr1 Mode is set to 3 (=11b) the affine (transformation) helper is enabled.

When reading from ADDR1 in this mode, the affine helper reads tile data from a special tile area defined by two new FX registers:

- FX_TILEBASE is pointed to a set of 8x8 tiles in either 4-bit or 8-bit depth. FX can support up to 256 tile definitions, and can overlap the traditional layer tile bases.
- FX_MAPBASE points to a square-shaped tile map, one byte per tile. This tile map has no attribute bytes. unlike the traditional layer 0/1 tile maps.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2A	FX_TILEBASE (DCSEL=2) (Write only)	**FX Tile Base Address (16:11)**						**Affine Clip Enable**	2-bit Polygon
$9F2B	FX_MAPBASE (DCSEL=2) (Write only)	**FX Map Base Address (16:11)**						**Map Size**	

- **Affine Clip Enable** changes the behavior when the X/Y positions are outside of the tile map such that it always reads data from tile 0. The default behavior is to wrap the X/Y position to the opposite side of the map.
- **Map Size** is a 2 bit value that affects both the width and height of the tile map.

Map Size	Dimensions
0	2×2
1	8×8
2	32×32
3	128×128

- The **Transparent Writes** toggle in FX_CTRL is especially useful in Affine helper mode. Setting this toggle causes a write of zero to leave the byte (or the nibble) at the target address intact. This toggle is not limited to affine helper mode, and it affects writes to both DATA0 and DATA1.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	**Transp. Writes**	Cache Write Enable	Cache Fill Enable	One-byte Cache Cycling	16-bit Hop	4-bit Mode	Addr1 Mode	

When using the affine helper, the X and Y position registers (DCSEL=4) are used to set ADDR1 to the source pixel indirectly in the aforementioned tile map, while the X and Y increments determine the step after each read of ADDR1.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_X_POS_L (DCSEL=4) (Write only)	**X Position (7:0)**							
$9F2A	FX_X_POS_H (DCSEL=4) (Write only)	X Pos. (-9)	-				**X Position (10:8)**		
$9F2B	FX_Y_POS_L (DCSEL=4) (Write only)	**Y/X2 Position (7:0)**							
$9F2C	FX_Y_POS_H (DCSEL=4) (Write only)	Y/X2 Pos. (-9)	-				**Y/X2 Position (10:8)**		

The affine helper supports the full range of X and Y increment values, including negative values.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

$9F29	FX_X_INCR_L (DCSEL=3) (Write only)	X Increment (-2:-9) (signed)							
$9F2A	FX_X_INCR_H (DCSEL=3) (Write only)	X Incr. 32x	X Increment (5:0) (signed)						X Incr. (-1)
$9F2B	FX_Y_INCR_L (DCSEL=3) (Write only)	Y/X2 Increment (-2:-9) (signed)							
$9F2C	FX_Y_INCR_H (DCSEL=3) (Write only)	Y/X2 Incr. 32x	Y/X2 Increment (5:0) (signed)						Y/X2 Incr. (-1)

32-bit cache

When the CPU reads a byte via DATA0 or DATA1, and "cache fill enable" is set, the value read will be copied into an indexed location inside the 32-bit cache.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	Transp. Writes	Cache Write Enable	**Cache Fill Enable**	One-byte Cache Cycling	16-bit Hop	4-bit Mode	Addr1 Mode	

In 8-bit mode, a byte is cached, but in 4-bit mode, a nibble is cached instead. Afterwards, by default, the index into the cache is incremented, and loops back around to 0 after the last index. The index can be set explicitly via the FX_MULT register. 8-bit mode uses bits 3:2 and ranges from 0-3. 4-bit mode uses bits 3:1 and ranges from 0-7.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2C	FX_MULT (DCSEL=2) (Write only)	Reset Accum.	Accumulate	Subtract Enable	Multiplier Enable	**Cache Byte Index**		**Cache Nibble Index**	Two-byte Cache Incr. Mode

Alternatively, the cache index can cycle between two adjacent bytes: 0, 1, and back to 0; or 2, 3, and back to 2. This option only has effect in 8-bit mode.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2C	FX_MULT (DCSEL=2) (Write only)	Reset Accum.	Accumulate	Subtract Enable	Multiplier Enable	Cache Byte Index		Cache Nibble Index	**Two-byte Cache Incr. Mode**

Setting the cache data directly

Instead of filling the cache by reading from DATA0 or DATA1, the cache data can also be set directly by writing to the FX_CACHE* registers. Setting the cache directly does not affect the cache index.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CACHE_L (DCSEL=6) (Write only)	Cache (7:0) | Multiplicand (7:0) (signed)							
$9F2A	FX_CACHE_M (DCSEL=6)	Cache (15:8) | Multiplicand (15:8) (signed)							

	(Write only)	
$9F2B	FX_CACHE_H (DCSEL=6) (Write only)	Cache (23:16) | Multiplier (7:0) (signed)
$9F2C	FX_CACHE_U (DCSEL=6) (Write only)	Cache (31:24) | Multiplier (15:8) (signed)

Writing the cache to VRAM

If "Cache write enabled" is set, the cache contents are written to VRAM when writing to DATA0 or DATA1. The primary use is to write all or part of the 32-bit cache to the 4-byte-aligned region of memory at the current address.

Control over which parts are written are chosen by the value written to DATA0 or DATA1. The value written is treated as a **nibble mask** where a 0-bit writes the data and a 1-bit masks the data from being written.In other words, writing a 0 will flush the entire 32-bit cache. Writing `#%00001111` will write the second and third byte in the cache to VRAM in the second and third memory locations in the 4-byte-aligned region.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	Transp. Writes	**Cache Write Enable**	Cache Fill Enable	One-byte Cache Cycling	16-bit Hop	4-bit Mode	Addr1 Mode	

Transparency writes

Transparent writes, when enabled, also applies to cache writes. If enabled, zero bytes (or zero nibbles in 4-bit mode) in the cache, which are treated as transparency pixels, are not written.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	**Transp. Writes**	Cache Write Enable	Cache Fill Enable	One-byte Cache Cycling	16-bit Hop	4-bit Mode	Addr1 Mode	

When "one-byte cache cycling" is turned on and DATA0 or DATA1 is written to, the byte at the current cache index is written to VRAM. When "Cache write enable" is set as well, the byte is duplicated 4 times when writing to VRAM.

Usually the incrementing of the cache index is only triggered by reading from DATA0 or DATA1 when cache filling is enabled. However it can also be triggered by reading from DATA0 in polygon mode when cache filling is not enabled and "one-byte cache cycling" is enabled.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	Transp. Writes	Cache Write Enable	Cache Fill Enable	**One-byte Cache Cycling**	16-bit Hop	4-bit Mode	Addr1 Mode	

Multiplier and accumulator

The 32-bit cache also doubles as an input to the hardware multiplier when Multiplier Enable is set.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2C	FX_MULT (DCSEL=2) (Write only)	Reset Accum.	Accumulate	Subtract Enable	**Multiplier Enable**	Cache Byte Index		Cache Nibble Index	Two-byte Cache Incr. Mode

To do a single multiplication, put the two 16-bit inputs into the two halves of the 32-bit cache.

```
    lda #(2 << 1)
    sta VERA_CTRL        ; $9F25
    stz VERA_FX_CTRL     ; $9F29 (mainly to reset Addr1 Mode to 0)
    lda #%00010000
    sta VERA_FX_MULT     ; $9F2C
    lda #(6 << 1)
    sta VERA_CTRL        ; $9F25
    lda #<69
    sta VERA_FX_CACHE_L  ; $9F29
    lda #>69
    sta VERA_FX_CACHE_M  ; $9F2A
    lda #<420
    sta VERA_FX_CACHE_H  ; $9F2B
    lda #>420
    sta VERA_FX_CACHE_U  ; $9F2C
```

The accumulator can be used to accumulate the sum of several multiplications. Before doing this single multiplication, ensure this is reset this to zero, otherwise the output will be added to the value of the accumulator before being written. There are two methods to do this. The first is to write a 1 into bit 7 of FX_MULT ($9F2C, DCSEL=2). The other, more conveniently, is to read FX_ACCUM_RESET (the same register location as VERA_FX_CACHE_L).

```
    lda FX_ACCUM_RESET   ; $9F29 (DCSEL=6)
```

To perform the multiplication, it must be written to VRAM first. This is done via the cache write mechanism. Usually the cache itself is written to VRAM if "Cache Write Enable" is set. However, if the "Multiplier Enable" bit is also enabled, the multiplier result is written to VRAM instead.

```
    ; Set the ADDR0 pointer to $00000 and write our multiplication result there
    lda #(2 << 1)
    sta VERA_CTRL        ; $9F25
    lda #%01000000       ; Cache Write Enable
    sta VERA_FX_CTRL     ; $9F29
    stz VERA_ADDRx_L     ; $9F20 (ADDR0)
    stz VERA_ADDRx_M     ; $9F21
    stz VERA_ADDRx_H     ; $9F22 ; no increment
    stz VERA_DATA0       ; $9F23 ; multiply and write out result
    lda #%00010000       ; Increment 1
    sta VERA_ADDRx_H     ; $9F22 ; so we can read out the result
    lda VERA_DATA0
    sta $0400
    lda VERA_DATA0
    sta $0401
    lda VERA_DATA0
    sta $0402
    lda VERA_DATA0
    sta $0403
```

Note: the VERA works by pre-fetching the contents from VRAM whenever the address pointer is changed or incremented. This happens even when the address increment is 0. Due to this behavior, it is possible to have stale data latched in one of the two data ports if the underlying VRAM is changed via the other data port. This example avoids this scenario by only using ADDR0/DATA0. This potential gotcha was not introduced by the FX update, but rather has always been how VERA behaves.

Accumulation

One can also trigger the multiplication and add it to (or subtract it from) the multiplication accumulator by calling "accumulate" in one of two different ways. We could write a 1 into bit 6 of FX_MULT ($9F2C, DCSEL=2), but more conveniently, we can read FX_ACCUM (the same register location as VERA_FX_CACHE_M)

```
    lda FX_ACCUM         ; $9F2A (DCSEL=6)
```

Once the accumulation is triggered, the result of the operation is stored back into the accumulator.

The default accumulation operation is (multiply then) add. This can be switched to subtraction by setting the Subtract Enable bit in FX_MULT

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F2C	FX_MULT (DCSEL=2) (Write only)	Reset Accum.	Accumulate	**Subtract Enable**	Multiplier Enable	Cache Byte Index		Cache Nibble Index	Two-byte Cache Incr. Mode

If the multiplication accumulator has a nonzero value, any multiplications carried out via a VRAM Cache write will be offset by the value of the accumulator (either added to or subtracted from the accumulator), but they will not change the value of the accumulator.

16-bit hop

There is a special address increment mode that can be used to read pairs of bytes via ADDR1.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$9F29	FX_CTRL (DCSEL=2)	Transp. Writes	Cache Write Enable	Cache Fill Enable	One-byte Cache Cycling	**16-bit Hop**	4-bit Mode	Addr1 Mode	

In this mode, setting ADDR1's increment to +4 will result in alternating increments of +1 and +3. Setting it to +320 will result in alternating increments of +1 and +319. All other increment values, including negative increments, lack this special hop property.

After this bit is set, writing to ADDRx_L resets the hop alignment such that the first increment is +1.

This mode is useful for reading out a series of 16-bit values after a series of multiplications.

For a more detailed explanation of chained math operations, see the tutorial .