FalvoTech Data Book

Samuel A. Falvo II

Feb 06, 2021

CONTENTS

FalvoTech is a small freelance FPGA and electronics kit development company focusing on the retro- and neo-retro computing hobbies. You can get in touch with FalvoTech in the following ways:

Email: Samuel A. Falvo II <kc5tja@arrl.net>

Mastodon: @vertigo@hackers.town

Twitter: @samuelafalvoii

For sales, check out our store at Tindie.com.

CHAPTER

ONE

ABOUT FALVOTECH

1.1 Designs for Retrocomputing

Retrocomputing is a fun hobby to engage in. Many enthusiasts dream of *building their own* computers with the look and feel of the 70s, 80s, and early 90s. These are entirely new designs, but devices which maintain a classic feel. Alternatively, many enthusiasts may wish to keep their existing computers of those eras in good, working condition. However, many of these systems used highly customized support logic, frequently in the form of specialized, purpose-built chips. Today, the inventory of genuine, legacy custom support logic is declining. Preservationists eventually will be incentivized to hoarde these custom chips and modules in the future. This is already happening with the world famous MOS 6581 SID chip.

FalvoTech aims to offer ready-to-use designs in both gateware[1] and hardware forms, particularly of the lesser known but no less important support logic not already handled through other vendors. We combine open-source development tooling with supported FPGAs to relieve the customer of the burden of starting on customized logic design from scratch. Using our inventory of pre-made designs, retrocomputing fans can once again build the computers of their dreams, or fix ones they cherish dearly.

[1] The programming used by field-programmable gate arrays, or FPGAs.

CHAPTER

TWO

BX-PLORER KIT

Hi there.

CHAPTER

THREE

VDC-II CORE DATA SHEET

Warning: This data sheet is preliminary.

- Natural upgrade to MOS 8568 Video Display Controller chip.
- 64KiB video RAM support.
- Increased bandwidth to video memory reduces software busy-wait loop delays.
- 512 color palette with 16 programmable color registers.
- Integrated CRTC based on MOS 6545 register set.
- 25.175MHz dot clock supports VGA displays.
- Interlace mode emulation.
- Character heights from 1 to 32 pixels tall.
- Programmable hardware cursor supports underline or block shapes.
- Automatic increment on video RAM read or write.
- Three attribute modes.
  - Mode 0 (VDC) supports 512 characters, underline, blink, and reverse video with 16 foreground colors.
  - Mode 1 (VDC-II) supports 512 characters, 8 background colors, and 16 foreground colors.
  - Mode 2 (VDC-II) supports 256 characters, 16 background colors, and 16 foreground colors.
- Programmable character widths from 3 to 8 pixels.
- Horizontal and vertical smooth scroll support.
- Block Write function rapidly fills video memory with a constant byte.
- Block Copy function supports scrolling video data.
- Semigraphic mode eliminates gaps between characters.
- Bitmapped graphics mode without attributes (monochrome) or with attributes (tiled colors).

3.1 Introduction

The VDC-II Video Display Controller provides an integrated solution to displaying text and graphics on a progressive-scanned video display. It is a natural upgrade to the MOS 8563 and 8568 Video Display Controller chips.

Unlike Commodore's earlier chips, the VDC-II is designed to operate with IBM VGA-compatible display devices using progressive-scan video. For example, with interlace mode turned off, the VDC-II will render every other scanline as black, emulating non-interlaced mode on NTSC monitors. With interlace mode enabled, access to the full vertical resolution is enabled.

Depending on how the VDC-II is configured, it might also work with some super-VGA display resolutions, provided you can fit the resources into display memory, and you can provide a fast enough dot-clock.

Much of the information in this document comes from the Commodore 128 Programmer's Reference Guide, amended as required for documenting the VDC-II behavior.

3.2 Ports

I/O ports, or just *ports*, refer to registers which are directly addressible by the host CPU. The VDC-II, like all of its predecessors, exposes only two byte-wide registers to the host CPU.

Offset	R/W	Function
0	R	Status Register
0	W	Register Select
1	R/W	Data Register

3.2.1 Status Register

	R/W	7	6	5	4	3	2..0
STATUS	R	JOB	0	VBL	0	0	V2 V1 V0

Along with the VDC-II, there are four different versions of VDC chip, each requiring slightly different register initialization values to produce a stable display. The difference between the 8563-R7A and subsequent revisions (8563-R8, 8563-R9, all 8568 revisions, and VDC-II) is in the horizontal smooth scroll feature, represented by bits 3-0 of register 25. All versions of the 8563 and 8568 support interlaced mode, whereas the VDC-II does not.

Even if these features are not used, the correct values must be placed into the VDC(-II) registers for a normal display. The VDC version can be easily ascertained through software by reading the status register. Bits 0 - 2 contain the version number. Refer to the table below for the actual data to be used.

VDC Revision	Bits 2-0
MOS 8563 R7A	0
MOS 8563 R8, R9	1
MOS 8568	2
FalvoTech VDC-II Core	3
reserved	4-7

Bit 5 is set while the vertical sync pulse is asserted, and cleared otherwise.

Bit 6 is not used by the VDC-II.

Note: Bit 6 used to be used to indicate whether a light pen trigger occurred. However, light pen support is not included in the VDC-II core.

Bit 7 indicates whether or not the VDC-II is currently performing a video memory read, write, block copy, or block write operation (1) or if it is idle (0).

3.2.2 Register Select

	R/W	7	6	5..0
REGSEL	W	-	-	RSEL

This lets the host CPU select one of the internal registers.

3.2.3 Data Register

	R/W	7..0
DATA	R/W	data

Depending on which internal register is currently selected by REGSEL, the view of the internal register appears here if read. The selected internal register may be changed if written to.

Note: Writing to this register may cause one or more side-effects, depending on which internal register is selected.

3.3 Internal Register Set

Much of the information in this section comes from the Commodore 128 Programmer's Reference Guide, amended as required for documenting the VDC-II behavior.

3.3.1 CRTC Registers

The CRT Controller section is based on the MOS 6545 design.

HTOTAL

			7 .. 0
R0	HTOTAL	R/W	HT

The number of characters between successive horizontal sync pulses, minus one. This number includes the displayed part of a character row, all borders, and the blanking interval during horizontal sync pulses.

HDISP

			7 .. 0
R1	HDISP	R/W	HD

The number of characters displayed on each character row. This number sets the width of the displayed part of the character row.

HSPOS

			7 .. 0
R2	HSPOS	R/W	HP

The number of characters from the beginning of the displayed part of a character row to the start of the horizontal sync pulse.

This register controls the horizontal position of the text on the display. Increasing this register's value shifts the display to the left. This register should be greater than the number in R1.

SYNCWID

			7 .. 4	3 .. 0
R3	SYNCWID	R/W	VW	HW

This register contains the widths of both the horizontal and vertical sync pulses.

Bits 3-0 contain the width of the horizontal sync pulse in characters, plus one. Bits 7-4 contain the width of the vertical sync pulse in scan lines.

Note: In the original MOS VDC designs, if interlace mode were enabled, then you would need to program the vertical sync width to be *double* the intended number of scanlines. This is because all vertical counters change by *two* when in interlaced mode, instead of by one.

Since VDC-II only operates with progressive-scanned displays, **do not** double the vertical sync width even if interlace emulation is enabled.

VTOTAL

			7 .. 0
R4	VTOTAL	R/W	VT

The number of characters rows between successive vertical sync pulses, minus one. This number includes the displayed rows, vertical borders, and the blanking interval during the vertical sync pulse. It, in conjunction with VTADJ (register R5), determines the vertical sync rate.

VTADJ

			7 .. 5	4 .. 0
R5	VTADJ	R/W	111	VT

The number of scan lines added to the end of the frame for fine adjustment of the vertical sync rate.

Note: In the original MOS VDC designs, if interlace mode were enabled, then you would need to program the vertical total adjust to be *double* the intended number of scan lines.

Since VDC-II only operates with progressive-scanned displays, **do not** double the vertical total adjust setting, even if interlace emulation is enabled.

VDISP

			7 .. 0
R6	VDISP	R/W	VD

The number of character rows displayed in a frame. This number sets the height of the displayed part of the frame.

VSPOS

			7 .. 0
R7	VSPOS	R/W	VP

The number of character rows from the first displayed character row to the start of the vertical sync pulse, plus one. This number should be greater than the number in VDISP (register R6).

This register also determines where the displayed character rows will appear on the screen. Increasing this number will shift the display upward.

IMCTRL

			7 .. 2	1	0
R8	IMCTRL	R/W	111111	IM1	IM0

The original MOS VDC defined three raster scan display modes: non-interlaced, interlaced sync, and, interlaced sync and video.

Mode	IM1	IM0
Non-interlaced	-	0
Interlaced Sync	0	1
Interlaced Sync and Video	1	1

Non-interlaced

The original MOS VDC chips would scan video data to the NTSC monitor with an identity mapping from raster row in memory to monitor scan line. Since the display was not interlaced, the display would have black lines every other line.

The VDC-II emulates this behavior by rendering every other scan line as black.

Interlaced Sync

The original MOS VDC chips would scan video data *twice* to the NTSC monitor, thus filling in the black interstitial lines. If the monitor's display phosphor wasn't persistent enough, the viewer might see some display flicker.

The VDC-II emulates this behavior by progressively rendering the same scanline twice.

Interlaced Sync and Video

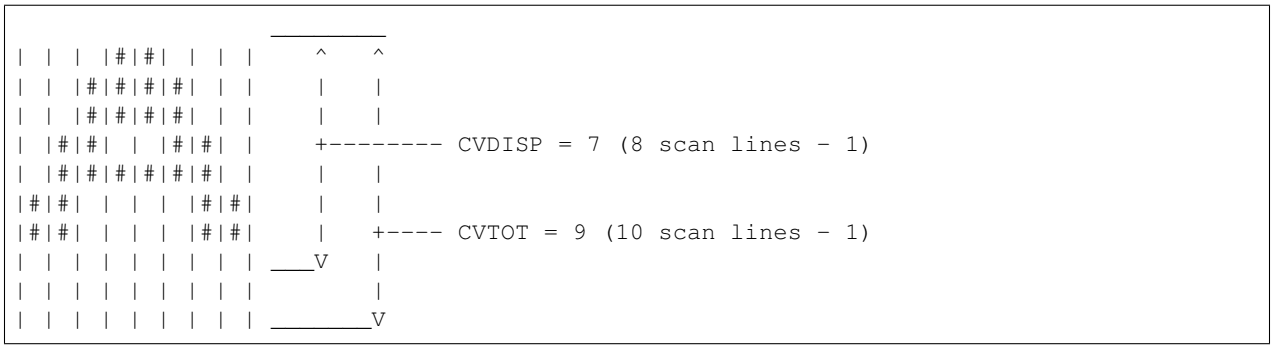
The original MOS VDC chips would scan video data *once* to the NTSC monitor, but would advance display pointers by *two* scan lines as it refreshed the display. This had the effect of rendering scan lines 0, 2, 4, 6, 8, ... on even fields, and scan lines 1, 3, 5, 7, 9, ... on odd fields. A single frame would consist of an odd and even field. While this doubled vertical resolution, the viewer might see substantial display flicker.

The VDC-II emulates this behavior by progressively rendering scanlines once.

CVTOT

			7 .. 5	4 .. 0
R9	CVTOT	R/W	111	CV

The number of scan lines in a character, minus one. This number includes the displayed part of a character and the vertical intercharacter spacing beneath that. This register sets the vertical dimension of a character and the vertical dimension of a character row.



See also *CVDISP*.

CURMOD

			7	6 .. 5	4 .. 0
R10	CURMOD	R/W	1	CM	CS

This is a three-purpose register which enables or disables the hardware cursor, determines its blink rate (if any), and sets the starting scan line for the cursor display.

Cursor Mode

Bits 6-5 of the CURMOD register actually determines whether the cursor is visible (enabled) or invisible (disabled), and if visible, sets its blink rate.

Cursor Mode	CM	Enabled?
Solid Cursor	0	Yes
No Cursor (disabled)	1	No
Blinking Cursor (1/16 frame rate)	2	Yes
Blinking Cursor (1/32 frame rate)	3	Yes

Cursor Start Scan Line

Bits 4-0 of the CURMOD register sets the starting scan line on which the cursor will be visible. Combined with CUREND (register R11), this field can help define the overall shape of the cursor. For example, if CS = CE - 1, then the resulting shape of the cursor will be an underline. However, if the cursor start is set to 0 and the cursor end is set to the height of the character row, you'll end up with a block cursor.

CUREND

			7 .. 5	4 .. 0
R11	CUREND	R/W	111	CE

Bits 4-0 of the CUREND register sets the final scan line on which the cursor will be visible.

Note: In the original MOS VDC designs, you had to set this register to the final scan line *plus one*. Otherwise, you would end up with a cursor which did not cover the entire glyph. Users of the Commodore 128 in 80-column mode will recognize that the cursor is short one scan line, because the KERNAL developers failed to account for this behavior.

This visual bug is fixed in the VDC-II, and the *plus one* requirement is waived. That said, there is never any harm in adding one.

DISPADR

			7 .. 0
R12	DISPADRH	R/W	DA (high)
R13	DISPADRL	R/W	DA (low)

The address of the first (left) eight pixels of the top scan line of the frame is defined by R12 and R13. In text mode, this address points to the first character to be displayed. In bitmapped mode, this address points to the first byte to be treated directly as pixels to be displayed. The addresses of subsequent bytes on a scan line (bitmapped mode) or characters on a character row (text mode) follow sequentially. The gap *between* scan lines or character rows, if any, is determined by ADRINC (register R27).

The most significant byte sits in R12, the least significant in R13.

CURPOS

			7 .. 0
R14	CURPOSH	R/W	CP (high)
R15	CURPOSL	R/W	CP (low)

The position of the cursor is set by R14 and R15. These two registers contain the 16-bit address of the cursor. R14 is the most significant byte, while R15 is the least significant. If the address in CURPOS is the address of a character within the displayed part of the frame, then that character will be displayed in reverse video for those scanlines determined by CURMOD and CUREND.

LPEN

			7 .. 0
R16	LPENH	R	11111111
R17	LPENL	R	11111111

These vestigial registers used to refer to which character on the screen the light pen was pointing at. For the VDC-II, these registers are no longer used and are now reserved.

3.3.2 VDC Registers

RAMPTR

			7 .. 0
R18	RAMPTRH	R	RP (high)
R19	RAMPTRL	R	RP (low)

The CPU communicates to the VDC-II video memory via address and data registers in the VDC-II. The RAMPTR registers indicates *where* the CPU will perform this interaction.

Reading from Video Memory

For the CPU to read a memory location, it must place that address in registers R18 (most significant byte) and R19 (least significant). The VDC-II responds by executing a read of that memory location and placing that data into RAMDAT (register R31), which the CPU may then read. During the time that the read is pending and data is not yet valid in R31, the Update Ready bit of the status register (bit 7) will be 0. Upon completion of the read cycle, data will be valid in R31 and the bit will be 1. When the CPU reads the data in R31, the VDC-II increments the address in R18/R19 and performs a read of that address. This allows the CPU to read successive memory locations without repeatedly changing the addresses in R18/R19.

Writing to Video Memory

For the CPU to write data to a memory location, the address must be written to R18/R19 as in a read, described above. Following the automatic read and after the Update Ready bit is a 1, the CPU should write the desired data to RAMDAT (register R31). The VDC-II then writes the data to video memory at the address defined by R18/R19, increments the address in R18/R19, reads the data from the incremented memory location, and places it in R31. The VDC-II then sets the Update Ready bit to a 1. At all times, if the Update Ready bit is a 0, then CPU access to video memory is pending. Additional access of R18, R19, or R31 should be avoided until the Update Ready bit is a 1.

ATTRADR

			7 .. 0
R20	ATTRADRH	R/W	AA (high)
R21	ATTRADRL	R/W	AA (low)

The address for the attribute of the first (top-left) character of the frame is defined in R20 and R21. The most significant 8 bits is in R20, the least significant in R21. These set the complete 16-bit address for that first attribute.

The address of attributes of subsequent characters are incremented from the previous address; so, horizontally adjacent characters have attributes in adjacent memory locations.

CHTOT

			7 .. 4	3 .. 0
R22	CHTOT	R/W	CT	CD

Bits 7-4 specifies the total number of pixels across for each character, minus one. This number includes the displayed part of a character *and* the horizontal intercharacter spacing to the right of the displayed part.

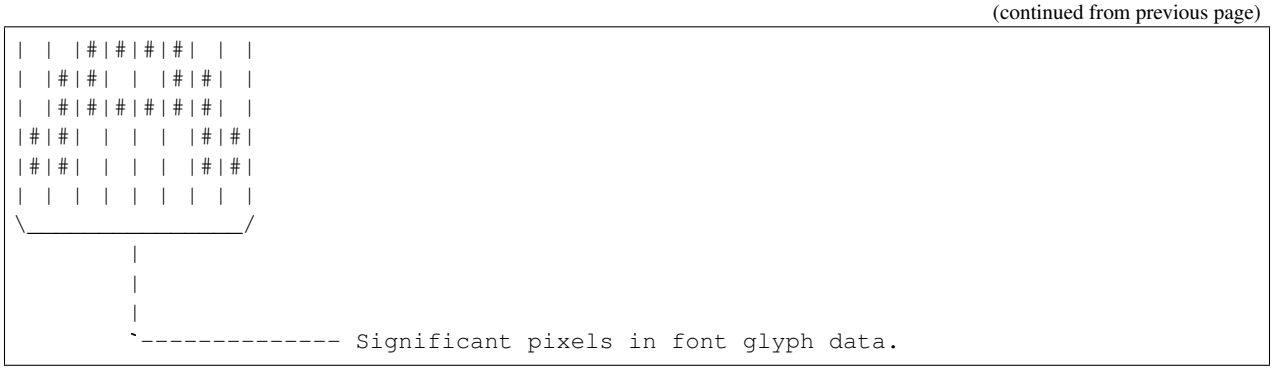
Bits 3-0 sets the width of the displayed part of the character, in pixels. If the defined character is to have no horizontal intercharacter spacing, then bits 3-0 should be initialized with a value equal to the value of bits 7-4.

```
|<---- CD ----->|  CHTOT[7-4] = 7 (8 pixels - 1)
|               |
|<---- CT ----->|  CHTOT[3-0] = either 7 or 8 will work
|               |

 7 6 5 4 3 2 1 0
| | | |#|#| | | |
| | |#|#|#|#| | |
```

(continues on next page)

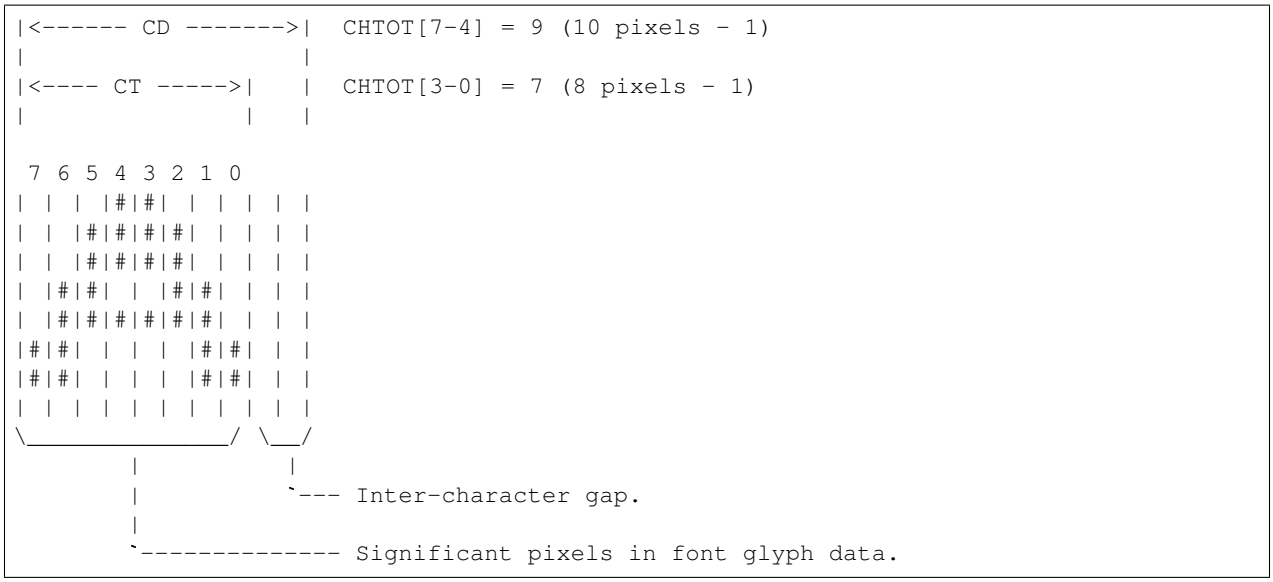
(continued from previous page)



This value may remain constant even with the use of horizontal smooth scrolling.

Note: The original VDC specifications indicates that one should set bits 3-0 to one plus the value in bits 7-4 if there are no intercharacter gaps. Experiments seems to show that the *plus one* isn't strictly required. As long as bits 3-0 are *greater than or equal* to the value in bits 7-4, no intercharacter gap will be displayed.

If the defined character is to have horizontal intercharacter spacing, then bits 3-0 should be initialized with a value equal to the number of pixels in the displayed part of a character *minus* one.



In this case, the value of bits 3-0 *must* be changed along with HSCROLL (register R25) to implement horizontal smooth scrolling.

CVDISP

			7 .. 5	4 .. 0
R23	CVDISP	R/W	111	VD

The number of scan lines of the displayed part of a character, minus one. This number sets the height of the displayed part of the character, and defines the vertical intercharacter spacing below the displayed part. If CVDISP is greater than or equal to CVTOT, then the displayed part of the character is equal to the total character, and the entire character is displayed, with no vertical intercharacter spacing.

See also *CVTOT*.

VSCROLL

			7	6	5	4 .. 0
R24	VSCROLL	R/W	CPY	RVS	BLR	VS

Bits 4-0 determines the vertical smooth scroll. The sudden scroll of an entire character row may be too abrupt a change for some applications. The VDC-II supports a method of vertically scrolling the screen one scan line at a time, up to one full character row. This is called vertical smooth scroll. This operates along with vertical scrolling to allow the screen to be scrolled up or down smoothly for as many character rows as desired.

To scroll the screen one scan line *up* vertically, the CPU should write a 1 to VSCROLL bits 4-0. This will skip the first scan line of the first character row. The first character row will then begin with the second scan line, moving it 'up' one scan line. All character rows on the frame will be movqd 'up' one scan line as well. At the bottom of the frame, one additional scan line will be displayed of a new character row. This will actually display parts of N + 1 character rows instead of the standard N.

The number in bits VSCROLL bits 4-0 can be increased, increasing the number of scan lines scrolled, until it equals CVTOT bits 4-0. This is the maximum amount of smooth scroll allowed. At this point, only one scan line (the bottom-most) will exist for the first character row and the last character row will have scrolled onto the frame with only the last scan line not visible. To scroll further, the CPU must change bits 4-0 of VSCROLL back to 0, and increase DISPADR (R12/R13; display start address) *and* ATTRADR (R20/R21; attribute start address) to scroll vertically by one character row. This sequence will have smooth scrolled by one character row.

Note: If you are using a monochrome bitmapped display, you do not need to use VSCROLL to smooth scroll vertically. Instead, you can just update DISPADR to the address of the desired top scan line of bitmap data.

If you are using attributes with a bitmapped display, however, you will still need to update VSCROLL as above, for attributes will continue to work in terms of characters, not in pixels.

Bit 5 controls how fast any characters with the BLN (blink) attribute blinks. If clear (0), characters will blink at 1/16 frame rate. If set (1), characters will blink at 1/32 frame rate.

Bit 6 controls whether the display will swap *all* foreground and background colors (1) or will leave foreground and background colors unchanged (0). This reversal compounds with other reversals, such as any character with the RVS attribute set and block cursor.

Bit 7 controls whether the block copy or block write mode is selected. To further improve the speed at which the CPU can manipulate video memory, two additional features have been added to the VDC-II: Block Write and Block Copy.

Block Write Operation

Block Write is an extension of the CPU write cycle, except that Block Write writes the same data to more than one successive memory location. This operation is set up as follows:

1. Write the initial address to R18/R19, waiting for the Update Ready Status bit to be 1,
2. Write the data to R31 and again waiting for the Update Ready Status bit to be 1.
3. If not already done, the CPU must then write a 0 to bit 7, selecting the Block Write mode,
4. Finally, write to BYTECT the number of successive memory locations the VDC-II should write.

Following the write to R30, the VDC-II will initiate one or more write cycles to video memory, writing the data currently in R31 to successive memory locations. The contents of R18/R19 will increment with the addresses being written. After the Block Writes are completed, R18/R19 will contain the address following the last memory location written.

Note: If you want to write N bytes of video memory this way, you should set BYTECT to N-1. Recall that the write to RAMDAT (register R31) will store the byte written to video memory as well, which contributes to the N bytes you wish to write.

Block Copy Operation

Block Copy is similar to Block Write, except that the data written to video memory is obtained from other successive memory locations. The Block Copy actually copies one part of video memory to another; it does not transfer data between video and CPU memory. The following steps initiates a block copy operation.

1. The CPU writes the initial *destination address* to RAMPTR (R18/R19), then waits for the Update Ready Status bit to be 1.
2. If not already done, the CPU then writes a 1 to the CPY bit of VSCROLL (bit 7 of R24), selecting the Block Copy mode.
3. The CPU then writes the initial *source address* to SRCPTR (R32/R33).
4. Then, the CPU writes the number of successive memory locations to be copied into to BYTECT (R30).

The 8563 will then read the contents of the first source address and write that data to the first destination address. Additional copies will occur at addresses incremented from the source and destination addresses, for as many words as are defined in BYTECT.

See also *BYTECT* and *SRCPTR*.

HSCROLL

			7	6	5	4	3 .. 0
R25	HSCROLL	R/W	BMM	AEN	SGM	0	HS

Bits 3-0 determine the current horizontal scroll setting.

The VDC-II has the ability to scroll smoothly in the horizontal direction. Horizontal smooth scrolling moves all characters on the screen to the *left* the desired number of pixels, up to the width of one character. Before horizontal smooth scrolling can occur, the VDC-II must have something to scroll onto the right side of the screen; so, ADRINC

(register R27) should be greater than 0. This sets up a virtual screen at least one character greater than the displayed screen.

An unscrolled screen is obtained with the value of bits 3-0 equal to the value of bits 7-4 of CHTOT (register R22, character total, horizontal). The screen is scrolled left by one pixel by *decrementing* the contents of bits 3-0. Additional scrolling occurs with decreasing values programmed. Maximum scrolling is achieved with a value of 0. At this point, only one (the right-most) pixel will exist for the first character of each character row and the last character will have scrolled onto the row with only one (the right-most) pixel still not visible. To scroll further, the CPU must change bits 3-0 back to its maximum value and increase DISPADR (R12/R13, Display Start Address) and ATTRADR (R20/R21, Attribute Start Address) each by 1 to scroll horizontally by one character. This sequence will have smooth scrolled left by one character.

If the character has an intercharacter gap, then the value of R22 bits 3-0 (character displayed, horizontal) must be changed along with R25(3-0). If we let p = the number of horizontal pixels in the *displayed part* of a character, HS = the current value of bits 3-0 of HSCROLL, and CT = the current value of bits 7-4 of CHTOT, then the value of bits 3-0 of R22 should be equal to the following formula:

$$CD = (p + HS) mod (CT + 1)$$

In this case both registers bits 3-0 of R25 *and* R22 must be changed to scroll horizontally.

For example, for a character eight pixels wide with five pixels displayed:

	CT	HS	CD
Unscrolled	7	7	4
Scroll 1	7	6	3
Scroll 2	7	5	2
Scroll 3	7	4	1
Scroll 4	7	3	0
Scroll 5	7	2	7
Scroll 6	7	1	6
Scroll 7	7	0	5

With the VDC-II, there are four different versions of VDC chip, requiring slightly different register initialization values. The difference between the 8563-R7A and subsequent revisions (8563-R8, 8563-R9, all 8568 revisions, and VDC-II) is in the horizontal smooth scroll feature, represented by bits 3-0 of register 25. Even if this feature is not utilized, the correct value must be placed into this register for a normal display. Software designed specifically for 8563-R7As will exhibit a problem with the leftmost side of the display when run on a system with R8s or later, and software designed specifically for 8563-R8s and later will exhibit a problem with the rightmost side of the display when run on a system with R7A. To run correctly on any system, the software must initialize register 25 with the correct data for the particular 8563 version in the system. The 8563 version can be easily ascertained through software by reading the 8563 status register. See *Status Register* for more information on chip identification.

Bit 4 is not currently used by the VDC-II and is reserved.

Note: In the original VDC chips from Commodore, bit 4 of HSCROLL used to control horizontal display resolution. If clear (0), it would render the display in a high resolution (e.g., 640 pixels per scan line). If set (1), it would halve the display resolution (e.g., 320 pixels per scan line). It otherwise offered no additional features.

It worked by internally halving the dot-clock, which means that all the CRTC registers would need to be reprogrammed in terms of the new character periods.

Bit 5 controls semi-graphic mode (1) or non-graphic mode (0).

Bit 6 controls whether attributes are enabled (1) or the display is in a monochrome mode (0). In some applications, attributes are not necessary. In these cases, the amount of video memory used by the attributes may be excessive,

so the VDC-II has a control bit to disable attribute usage. In this case, no character may be underlined, individually reversed, or blinked. Only one character set of 256 characters may be used. The foreground color of all characters will be the same and will be determined by R26 bits 7-4. The background color will still be set by R26 bits 3-0. With attributes enabled, however, per-character attributes will be fetched and used to provide a more visually interesting display. It also allows the use of two character sets of 256 characters each to be freely mixed on the same display.

Bit 7 controls whether the VDC-II operates in bit-mapped mode (1) or in text mode (0). The VDC-II is primarily a text display device, but it does support a limited bit-mapped graphics mode. In the text mode, the displayed frame is an matrix of characters, each with a unique pointer to reference the character, and an attribute to define additional display characteristics. In the bit-mapped mode each pixel is controlled by a unique bit in video memory.

FGBG

			7 .. 4	3 .. 0
R26	FGBG	R/W	FP	BP

In all cases, the background color is determined by bits 3-0. The borders surrounding the screen and the vertical and horizontal intercharacter spaces are displayed at the background color.

When attributes are disabled, the foreground color for all characters is determined by bits 7-4. When attributes are enabled, bits 7-4 are ignored.

ADRINC

			7 .. 0
R27	ADRINC	R/W	AI

An additional feature of the VDC-II is the ability to scroll horizontally, both left and right through video memory. To scroll a window of i horizontal bytes by j vertical scan lines across a virtual screen of k horizontal by l vertical ($i < k$), there must be (k x l) bytes of bit-mapped data in video memory. The VDC-II must skip (k-i) bytes on every scan line, because only i bytes are displayed of a virtual line of k bytes in video memory. The VDC-II has the ability to skip a number of bytes, set by writing to ADRINC. If ADRINC is 0, then no bytes will be skipped. A nonzero value in R27 is necessary to allow horizontal smooth-scrolling.

The value in R27 is used to increment the address of the bit-mapped data from one scan line to the next and to increment the address of the attributes from one character row to the next. Both the bit-mapped data and the attributes will be incremented by the same amount.

CHRSET

			7 .. 5	4 .. 0
R28	CHRSET	R/W	CS	11111

Bits 4-0 are not supported. The VDC-II exposes a *static RAM* interface to the surrounding hardware. When synthesized onto an FPGA or embedded in an ASIC, the engineer is expected to couple the VDC-II with an appropriate memory controller interface.

Note: In MOS VDC designs, this field used to select between 16Kx4 and 64Kx1, asynchronous, dynamic RAMs. The VDC-II is intended to be used with on-chip block memory and/or with external synchronous dynamic RAMs.

Thus, this field is no longer relevant. Writing to this field does nothing, and always returns all 1s.

Bits 7-5 determine the starting location of the character set in video memory, assuming the characters are 16 or fewer pixels in height. These bits form the most significant three bits of the 16-bit address used to access character data. The address of a character's glyph data is formed by OR-ing several data:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CS15	CS14	CS13	CC8	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	RL3	RL2	RL1	RL0

$$addr = 8192 \times cs_{15..13} + 16 \times charcode + scanline$$
$$where, 0 <= charcode < 512 \wedge 0 <= scanline < 16$$

If CVTOT is greater than 15, then only bits 7-6 will be used, because each character set will then occupy 16KiB of video memory.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CS15	CS14	CC8	CC7	CC6	CC5	CC4	CC3	CC2	CC1	CC0	RL4	RL3	RL2	RL1	RL0

$$addr = 16384 \times cs_{15..14} + 32 \times charcode + scanline$$
$$where, 0 <= charcode < 512 \wedge 0 <= scanline < 32$$

The character pointers, pointed at by DISPADR, correspond to a displayed character position on the frame. The 8-bit pointer selects one of the 256 characters in the character set to be displayed at that position on the frame. The 9th pointer bit (CC8) comes, if enabled, from the corresponding attribute byte. For a frame of i rows of j characters each, the first j pointers will define the characters in the first (top) character row, and (i x j) pointers will need to be defined in display RAM. For 25 character rows of 80 characters each, 2000 pointers will be needed, located at DISPADR.

ULINE

			7 .. 5	4 .. 0
R29	ULINE	R/W	111	UL

The number of the scan line used for underline. The underline is defined as a single scan lnie. Scan line 0 refers to the scan line at the top of the character.

BYTECT

			7 .. 0
R30	BYTECT	R/W	BC

This register hold the number of bytes to copy or write into video memory during a block write or copy operation.

See also *VSCROLL* and *SRCPTR*.

Note: On MOS VDC design documents, this register is, strangely, known as *word count*, even though it counts bytes instead of words.

RAMDAT

			7 .. 0
R31	RAMDAT	R/W	RD

The CPU communicates to the VDC-II video memory via address and data registers in the VDC-II. The RAMDAT register indicates *what* the CPU will perform this interaction with.

See *RAMPTR* for more details on how RAMPTR and RAMDAT are used.

SRCPTR

			7 .. 0
R32	SRCADRH	R/W	SA (high)
R33	SRCADRL	R/W	SA (low)

When copying data from one part of video RAM to another, this register provides the address of the source data. See *VSCROLL* for more information on block copy operation.

DSPBEG

			7 .. 0
R34	DSPBEG	R/W	DB

This register is not supported by the VDC-II core.

Warning: The Commodore 128 Programmer's Reference Guide does not provide a sufficiently detailed description of what this register does. Its functionality seems to overlap entirely with other CRTC registers, particularly the HTOTAL and HDISP registers.

The VDC-II core ignores this register.

See also *DSPEND*.

DSPEND

			7 .. 0
R35	DSPEND	R/W	DE

This register is not supported by the VDC-II core.

Warning: The Commodore 128 Programmer's Reference Guide does not provide a sufficiently detailed description of what this register does. Its functionality seems to overlap entirely with other CRTC registers, particularly the HTOTAL and HDISP registers.

What follows is the description taken from the Commodore 128 Programmer's Reference Guide in the hopes that it will be useful.

The VDC-II core ignores this register.

During the horizontal and vertical synchronization pulses, the R, G, B and I signals must be blanked (brought to a low level) to prevent the display of the scanning beam during retrace times. Two registers allow the user to adjust the beginning (R34) and end (R35) of the horizontal blanking interval.

Register R34 is programmed with the number of characters from the first dis- played character of a row to the first blanked character in that row. Register R35 is programmed with the number of characters from the first displayed character of a row to the last blanked character in that row. Blanking occurs on all scan lines of a frame.

RFRESH

			7 .. 0
R36	RFRESH	R/W	RF

This register is not supported by the VDC-II core.

Note: In MOS VDC designs, this register was used to configure how many DRAM refresh cycles occurred on each scan line. As the VDC-II exposes a static RAM interface to video memory, this register is redundant and no longer used.

HSPOL

			7	6	5 .. 0
R37	HSPOL	R/W	HSP	VSP	111111

This register first appears with the MOS 8568 VDC, and is used to determine the default HSYNC and VSYNC polarities.

If HSP is 1, then HSYNC is active *low*. Otherwise, it is active *high*. If VSP is 1, then similarly, VSYNC is active *low* as well. Otherwise, it is active *high*.

3.3.3 VDC-II Registers

The registers in this section are VDC-II specific. Unless otherwise documented elsewhere, attempting to *read* any of these registers on any MOS VDC device will return all ones.

DETECT

			7 .. 4	3	2	1	0
R38	DETECT	R	1111	ATR	IRQ	PAL	FDV

The VDC-II is a “soft” core design, implemented with programmable logic, and is therefore easily configurable to suit the implementor’s needs. More features use more logic on FPGAs and other logic devices; therefore, an implementor may elect to omit certain features to save logic space. This feature detect register is used to identify which features are

currently supported. It uses *negative logic*, as when reading from unsupported registers, the MOS VDC designs will return all-ones.

Bit 0 reflects the availability of R38 itself. If set, then the DETECT feature itself is not present, and *all* other VDC-II features discussed in this section are missing. If clear, the remaining bits hold valid feature support information.

Bit 1 reflects whether or not this VDC-II supports the programmable 16-color palette. If clear, then the palette registers discussed below will exist. If set, then the VDC-II will used the fixed color palette of the original MOS VDC designs.

Bit 2 reflects whether or not IRQs are supported by this VDC-II device. If clear, interrupt facilities beyond those of the 8568 are supported. If set, the VDC-II has the same interrupt abilities as a MOS 8568 VDC.

Bit 3 reflects whether or not alternative attribute modes are supported. If clear, the ATRMOD is supported. If set, the VDC-II only supports the MOS 8563 attribute bits.

CLRSEL

			7 .. 4	3 .. 0
R39	CLRSEL	R/W	0000	CL

This register exists only if DETECT.PAL = 0.

This register selects which color look-up register the red, green, and blue registers address.

Software may discover how many color registers are supported by writing $FF to this register, then reading the value back. The bits which remain set indicate a mask which determines the total number of color registers supported.

CLRRGB

			7 .. 5	4 .. 0
R40	CLRRGB	R/W	RGB	00000

This register exists only if DETECT.PAL = 0.

This register can be used to read or write color lookup information. Three bits of primary color information is supported, allowing a complete palette of 512 colors.

The bits are placed in a way which directly supports expanded palettes in the future. To determine total palette size, you can write $FF to a color's red, green, and blue channels, then read those channels back. The mask returned will indicate how many bits each channel supports.

Note: If your software is designed to work with a smaller palette size, then when writing color data to this register, you should *replicate* the data across the byte. This will make better use of a future VDC-II's expanded palette, providing a more vibrant display. Otherwise, you risk the resulting colors being a bit too dark.

After writing a color register number to CLRSEL, this register may then be used to set red, green, and blue bits (in that order) for that color. To provide a concrete example, suppose you're writing a program to alter a color register for a Z80-based computer. Assume the VDC-II is found at I/O port 6Eh. The following code will perform the task:

```
ChangeColor:
        LD      A,40
        OUT     (6Eh),A
```

(continues on next page)

(continued from previous page)

```
        LD      A,(ColorSelection)
        OUT     (6Fh),A

        LD      A,41
        OUT     (6Eh),A

        LD      A,(Red)
        OUT     (6Fh),A

        LD      A,(Green)
        OUT     (6Fh),A

        LD      A,(Blue)
        OUT     (6Fh),A

        RET
```

Note: *Any* write to CLRSEL will configure the CLRRGB register to next read or write the *red* channel.

CLRRGB will automatically increment CLRSEL after reading or writing the *blue* channel. This allows you to set a complete range of colors in a single operation.

```
ChangeColorRange:
        LD      A,40
        OUT     (6Eh),A

        LD      A,(ColorSelection)
        OUT     (6Fh),A

        LD      A,41
        OUT     (6Eh),A

        LD      HL,(ColorTable)
        LD      A,(NumColors)
        LD      B,A
        ADD     B               ; A = 2*NumColors
        ADD     B               ; A = 3*NumColors
        LD      B,A             ; B = number of bytes to output
        LD      C,6Fh
        OTIR

        RET
```

This type of program code can be useful for implementing various color effects, like color cycling.

To query the current color palette settings, you can read from this register as well. Remember that lower-order bits will be set to 0, depending on the specific VDC-II hardware you're working with.

```
ReadColorRange:
        LD      A,40
        OUT     (6Eh),A

        XOR     A
        OUT     (6Fh),A
```

(continues on next page)

(continued from previous page)

```
        LD      A,41
        OUT     (6Eh),A

        LD      HL,(ColorTable)
        LD      B,3*16
        LD      C,6Fh
        INIR
        RET
```

IRQENA

			7	6	5	4 .. 1	0
R41	IRQENA	R/W	JOB	0	VBL	0000	RCI

This register is only present if DETECT.IRQ = 0.

This register allows the CPU to control exactly which interrupt sources are able to interrupt the processor. The bits are as follows:

Bit 0 is the *raster compare interrupt*. See *RSTCMP* for details.

Bit 5 is the *vertical blank interrupt*. This interrupt fires whenever the vertical sync pulse is asserted, and is useful for synchronizing video games with the video frame rate.

Bit 7 is the *job done interrupt*. This interrupt fires whenever the current CPU-initiated video memory operation (read, write, block write, or block copy) completes.

Bits which are set to 1 *enables* the corresponding interrupt.

IRQPND

			7	6	5	4 .. 1	0
R42	IRQPND	R/W	JOB	0	VBL	0000	RCI

This register is only present if DETECT.IRQ = 0.

This register allows the CPU to detect outstanding conditions and events which might be of interest to the processor. The bits follows:

Bit 0 is the *raster compare interrupt*. See *RSTCMP* for details.

Bit 5 is the *vertical blank interrupt*. This interrupt fires whenever the vertical sync pulse is asserted, and is useful for synchronizing video games with the video frame rate.

Bit 7 is the *job done interrupt*. This interrupt fires whenever the current CPU-initiated video memory operation (read, write, block write, or block copy) completes.

Bits which are set to 1 indicates an outstanding condition.

Note: Just because a corresponding bit in IRQENA is 0 doesn't mean that a condition cannot happen. It only means that the condition *will not interrupt* the host processor. Thus, even with all interrupts disabled, the host processor may still use IRQPND to discover when certain events happen.

Reading from IRQPND will *not* reset this register to zero. To clear one or more bits in this register, write to this register with the corresponding mask bits set. For example, a quick and easy way to zero this register in an interrupt handler follows:

```
LD      A,43
OUT     (6Eh),A
IN      A,(6Fh)          ; Get interrupts pending
PUSH    AF

CALL    processInterruptsHere

POP     AF
OUT     (6Fh),A          ; and acknowledge them all.
```

RSTCMP

			7 .. 0
R43	RSTCMPH	R/W	RC (high)
R44	RSTCMPL	R/W	RC (low)

The RSTCMP register can be used to trigger the CPU on any supported raster line.

Note: Not all bits of this 16-bit wide register may be supported. The host processor can detect which bits are supported by writing $FFFF, then reading the resulting bit-mask back. For example, a VDC-II that supports a 1024x768 maximum display resolution might only support bits 10-0.

When VSYNC negates, an internal raster counter register resets to zero. Every HSYNC pulse, this counter increments. When this register is equal to RSTCMP, the IRQPND.RCI bit is set. If the corresponding IRQENA.RCI bit is set, then the host processor may receive an interrupt. The host processor can use this interrupt to, for example, change color palette or horizontal scroll settings.

Note: Many of VDC-II's settings are reloaded during horizontal sync, while others are reloaded only during vertical sync. The range of effects you can implement using raster compare interrupts is significantly limited, especially compared to more asynchronous video display devices, such as the MOS VIC-II chips. Many of these limitations stem from the memory technologies that this core is designed to work with.

Still, lack of a raster compare interrupt was an oft-cited oversight which the VDC-II finally fixes.

ATRMOD

			7	6	5 .. 2	1 .. 0
R45	ATRMOD	R/W	CC8	BP0	0000	AM

This register is only valid if DETECT.ATR = 0.

The ATRMOD register configures how attribute bytes are to be interpreted by the VDC-II core.

Bit 7 is used to supply the 9th character pointer bit when attributes are disabled or when 16 background color attribute mode is selected. Otherwise, this bit is ignored.

Bit 6 is used to supply the background color intensity bit when attributes are disabled or when 16 background color attribute mode is *not* selected. Otherwise, this bit is ignored.

Bits 1-0 selects how each attribute byte is to be interpreted.

		Attribute Byte Layout				
Mode	AM	7	6	5	4	3 .. 0
MOS 8563 Attributes	0	CC8	RVS	UNL	BLN	FP
VDC-II Attributes 1	1	CC8	BP3-BP1			FP
VDC-II Attributes 2	2	BP				FP
reserved	3	-				

In attribute mode 0, 512 characters are available for display. Each character may have any of 16 foreground colors, but all share a common background color, as set in the FGPG register. Each character may be individually reversed, underlined, or set to blink.

In attribute mode 1, 512 characters are available for display. Each character may have any of 16 foreground colors, and any of 8 colors drawn either from pens 0, 2, 4, 6, 8, 10, 12, and 14; or, from 1, 3, 5, 7, 9, 11, 13, and 15, depending on how BP0 is configured in ATRMOD. With the default color VDC palette, this provides either low-intensity or high-intensity RGB color.

In attribute mode 2, only 256 characters are available for display; the precise set of 256 characters is determined from the CC8 bit in ATRMOD. Each character may have any of 16 foreground colors *and* background colors.