Cereon

Cereon VDS1 reference manual

Authors:
Andrei Kapustin

Revision history

Date	Comment
21 October 2006	Initial draft.
24 May 2007	Removed unnecessary interrupts.
11 August 2007	Added 32*24 and 64*24 text modes.

1 Contents

2 Introduction

This manual is a definitive specification of the Cereon VDS1 architecture and operation.

2.1 Features overview

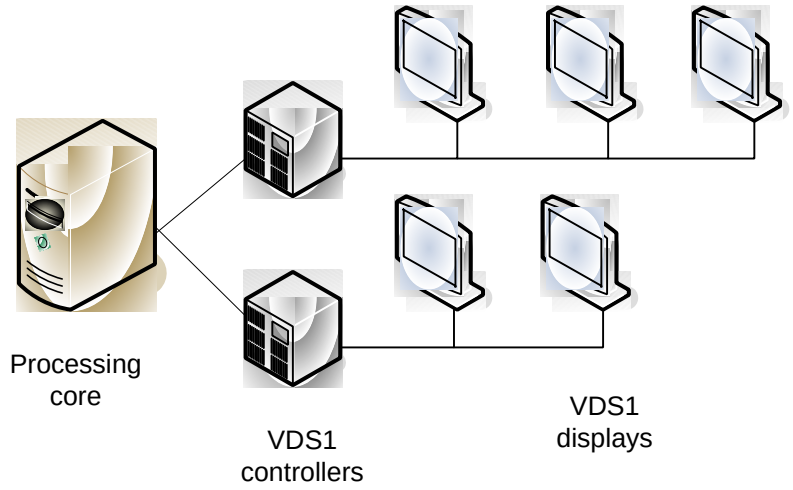
The Cereon VDS1 (Video Display System, version 1, henceforth referred to as simply VDS1) architecture defines a set of hardware components used for low-to-medium quality video display system, which provides peripheral devices for displaying textual and graphical data on compatible video display units. It is characterized by:

- Support for both textual and graphical information display.
- Low-bandwidth interface, which allows a large number of VDUs to be connected to the same computer.
- Controller sharing, which allows a single controller to manage a large number of display units.
- Paged video memory, which helps to eliminate the screen distortion during frequent screen updates.

All of these features are natural consequences of the main VDS1 design goal of supporting massively multi-user system configurations geared towards development and office activities. These goals are largely achieved by sacrificing features needed by an average home user (such as video streaming, video gaming, etc.)

2.2 Architecture overview

The following diagram presents a general overview of the VDS1 architecture:



Individual components of the VDS1 are explained in the following sections:

2.2.1 Processing core

This is a main processing core of a computer system, which contains main processor(s), memory, and other system components.

2.2.2 VDS1 controllers

Depending on the specific model and configuration of the computer where the VDS1 subsystem is used, VDS1 controllers can be bundled with the main processing core, connected as external devices or even bundled with VDS1 displays. In either case, there is no logical difference in the functions performed by VDS1 controllers – the said functions being receiving commands and data sent by the main processing core and using these commands and data to present required information in a required format on VDS display(s) connected to the controller.

Depending on the model, a single VDS1 controller can manage from 1 to 256 independent VDS1 displays. Similarly, particular computer systems can be equipped with any number of VDS1 controllers.

2.2.3 VDS1 display

A VDS display is a peripheral device that performs actual data visualization. Both the format and the contents of the visualized data are managed by the VDS1 controller to which the display is connected.

Note that the main processing core has no means of communicating with VDS1 displays directly – all communications are between the main processing core and the VDS1 controllers.

2.3 Programming overview

As far as the main processing core is concerned, each VDS1 controller connected to the main processing core is represented by a set of registers. Writing to these registers sends commands to the VDS1 controller; similarly, reading from these registers allows the main processing core to examine the status of the VDS1 controller.

The exact means by which VDS1 controller registers are made available to the main processing core can vary from system to system. For example:

- On a system with memory-mapped I/O, registers of a particular VDS1 controller can be mapped at predefined memory addresses within the I/O memory address space.
- On a system with non-memory-mapped I/O, registers of a particular VDS1 controller can be mapped onto a predefined set of I/O ports.

The above list is not exhaustive (for example, some system may allocate a separate coprocessor for VDS1 subsystem and map VDS1 controller registers to that coprocessor's registers, etc.) However, regardless of how VDS1 registers are presented to the main processing core, the same set of VDS1 registers is always defined for a VDS1 controller.

In the remainder of this manual, all information about programming VDS1 controllers will be given in terms of these logical VDS1 controller registers. In order to apply this documentation to a particular computer system, the programmer shall consult that computer system's documentation in order to understand how VDS1 registers are accessed by the main processing core.

3 Programming the VDS1

This chapter describes the programmer's interface to a VDS1 subsystem.

3.1 The VDS1 controller basics

A single VDS1 controller is capable of managing from 1 to 256 VDS1 displays. Depending on the model of the VDS1 controller in question, the maximum supported number of displays can vary.

3.1.1 Compartments

Internally, a VDS1 controller consists of compartments, each compartment capable of managing one VDS1 display. The maximum number of displays that can be managed by a VDS1 controller is, therefore, limited by the number of compartments in that VDS1 controller.

Compartments within a single VDS1 controller are identified by 8-bit numbers in range [0..255].

3.1.2 Compartment structure

All compartments of a VDS1 controller are identical in structure. Each compartment contains:

- 64KB (65536 8-bit bytes) of internal video memory. This video memory is used to hold information that is to be presented by the corresponding VDS1 display. The exact way in which the content of this video memory is presented on the corresponding display depends on the current compartment's settings (such as video mode, video page number, etc.)
- A number of registers, used for controlling both the compartment's behaviour and the related display. These registers are used to control the video mode, switch between video pages, participate in data transfer between video memory and main processing core, etc.

At any given time, one of the compartments within a VDS1 controller is selected as "current".

3.1.3 Controller registers

Registers of a VDS1 controller are subdivided into:

- Public registers, which are used for communication between the main processing core and the VDS1 controller, and
- Private registers, which are not directly accessible to the main processing core, but must be manipulated using commands sent to the public registers.

For example, to change the video mode of the display corresponding to the current compartment the program must write a new value to the private "video mode" register of that compartment. Since that private register is not directly accessible by the main processing core, the goal is achieved by issuing the "change video mode" command to public VDS1 controller registers.

3.2 Video memory

As already mentioned, each VDS1 display is allocated a 64KB video RAM in the VDS1 controller it is attached to. From the main processing core's point of view, this 64kB of video RAM represents 65536 individually addressable 8-bit bytes, with addresses [0..65535]. However, to a VDS1 subsystem the story is entirely different.

3.2.1 Video pages

A video page is an area within the video memory whose contents is made visible by the VDS1 display. It is often the case that several video pages can be kept in the video memory at the same time, each in its own video memory region. In this case, switching between pages provides a very fast way of updating the contents presented by a VDS1 display. Typically, a program would place data into a video page memory that corresponds to the not-currently-visible video page, and then make that video page current; this allows for flicker-free screen updates.

3.2.2 Video modes

Each compartment of a VDS1 controller has an associated video mode code, which is kept in a dedicated dynamic register. A video mode specifies:

- Whether the contents of video memory is presented in graphical or textual form.
- The display resolution (specified in characters for textual video modes or in pixels for graphical ones).

The following table summarizes video modes supported by the VDS1 subsystem:

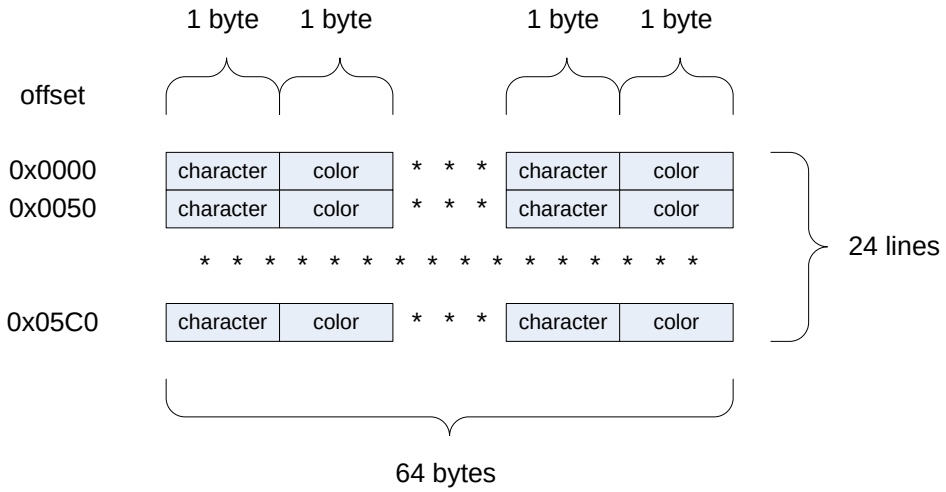
Video mode	Type	Width	Height	Colors	Pages
0	Textual	32	24	16	32
1	Textual	64	24	16	16
2	Textual	40	24	16	32
3	Textual	80	24	16	16
4	Graphical	256	192	16	2
5	Graphical	512	192	16	1
6	Graphical	512	384	2	2

The following sections elaborate on the supported video modes.

3.2.2.1 Textual 32*24

In this video mode, a VDS1 display presents 24 rows of 32 characters each. One of 16 colours can be selected for character and background independently.

The 64KB video memory is subdivided into 32 pages where page 0 occupies video memory addresses from 0 to 0x07FF, page 1 occupies video memory addresses from 0x0800 to 0x0FFF, and so on. Within each of these regions, the memory is organized as shown on the following diagram:



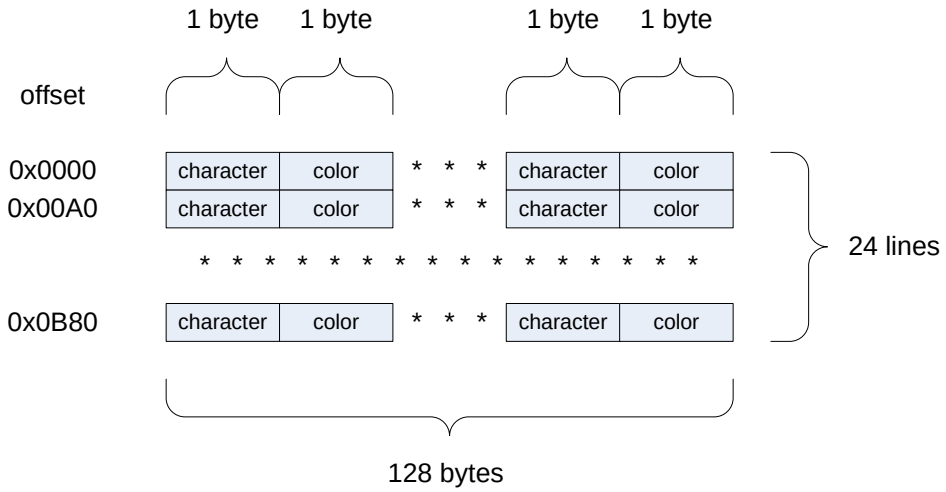
The video page is organized as 24 consecutive blocks of 64 bytes, each describing one line of 32 characters in a top-to-bottom line order. Within each line there are 32 <character>/<color> pairs, which describe the 32 displayed characters of that line in a left-to-right order. Within each pair, the <character> portion specified the 8-bit character code of the character occupying the corresponding screen position, while the 8-bit <color> value contains a background color code in the upper 4 bits and a foreground color code in the lower 4 bits. The character at the corresponding position is displayed using the foreground color, with the rest of the character position filled with a background color.

VDS1 controller allows character shapes to be changed dynamically, see the "Character generator" section below for details.

3.2.2.2 Textual 64*24

In this video mode, a VDS1 display presents 24 rows of 64 characters each. One of 16 colours can be selected for character and background independently.

The 64KB video memory is subdivided into 16 pages where page 0 occupies video memory addresses from 0 to 0x0FFF, page 1 occupies video memory addresses from 0x1000 to 0x1FFF, and so on. Within each of these regions, the memory is organized as shown on the following diagram:



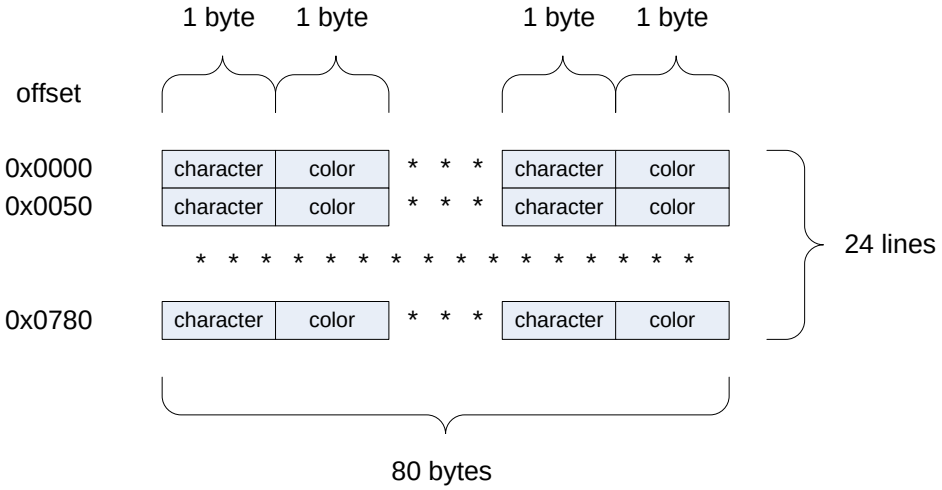
The video page is organized as 24 consecutive blocks of 160 bytes, each describing one line of 80 characters in a top-to-bottom line order. Within each line there are 64 <character>/<color> pairs, which describe the 64 displayed characters of that line in a left-to-right order. Within each pair, the <character> portion specified the 8-bit character code of the character occupying the corresponding screen position, while the 8-bit <color> value contains a background color code in the upper 4 bits and a foreground color code in the lower 4 bits. The character at the corresponding position is displayed using the foreground color, with the rest of the character position filled with a background color.

VDS1 controller allows character shapes to be changed dynamically, see the “Character generator” section below for details.

3.2.2.3 Textual 40*24

In this video mode, a VDS1 display presents 24 rows of 40 characters each. One of 16 colours can be selected for character and background independently.

The 64KB video memory is subdivided into 32 pages where page 0 occupies video memory addresses from 0 to 0x07FF, page 1 occupies video memory addresses from 0x0800 to 0x0FFF, and so on. Within each of these regions, the memory is organized as shown on the following diagram:



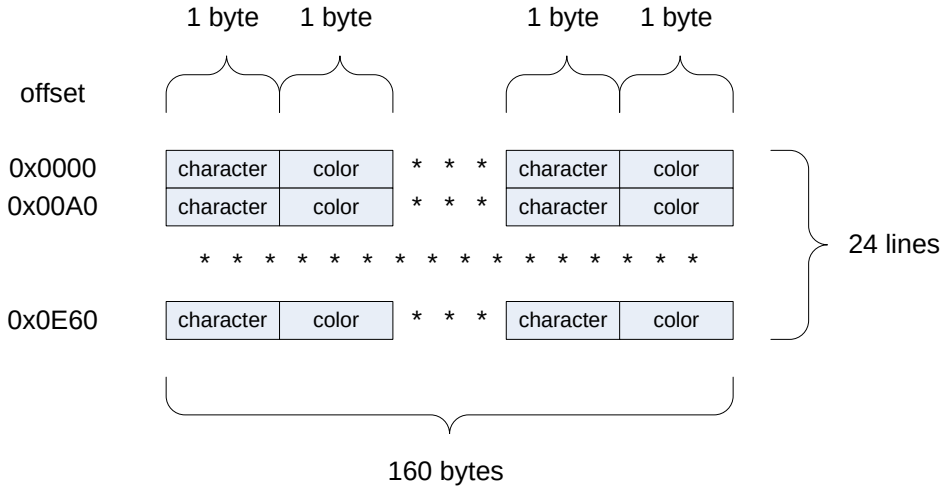
The video page is organized as 24 consecutive blocks of 80 bytes, each describing one line of 40 characters in a top-to-bottom line order. Within each line there are 40 <character>/<color> pairs, which describe the 40 displayed characters of that line in a left-to-right order. Within each pair, the <character> portion specified the 8-bit character code of the character occupying the corresponding screen position, while the 8-bit <color> value contains a background color code in the upper 4 bits and a foreground color code in the lower 4 bits. The character at the corresponding position is displayed using the foreground color, with the rest of the character position filled with a background color.

VDS1 controller allows character shapes to be changed dynamically, see the "Character generator" section below for details.

3.2.2.4 Textual 80*24

In this video mode, a VDS1 display presents 24 rows of 80 characters each. One of 16 colours can be selected for character and background independently.

The 64KB video memory is subdivided into 16 pages where page 0 occupies video memory addresses from 0 to 0x0FFF, page 1 occupies video memory addresses from 0x1000 to 0x1FFF, and so on. Within each of these regions, the memory is organized as shown on the following diagram:



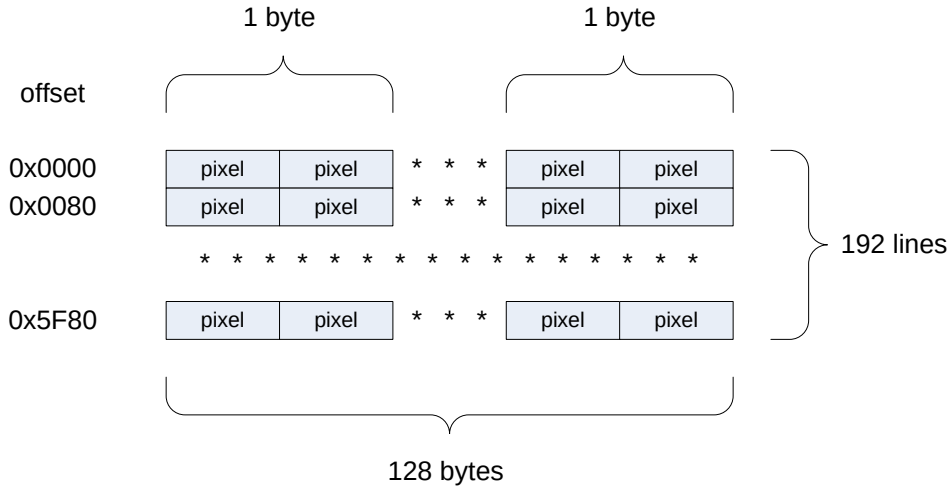
The video page is organized as 24 consecutive blocks of 160 bytes, each describing one line of 80 characters in a top-to-bottom line order. Within each line there are 80 <character>/<color> pairs, which describe the 80 displayed characters of that line in a left-to-right order. Within each pair, the <character> portion specified the 8-bit character code of the character occupying the corresponding screen position, while the 8-bit <color> value contains a background color code in the upper 4 bits and a foreground color code in the lower 4 bits. The character at the corresponding position is displayed using the foreground color, with the rest of the character position filled with a background color.

VDS1 controller allows character shapes to be changed dynamically, see the “Character generator” section below for details.

3.2.2.5 Graphical 256*192

In this video mode, a VDS1 display presents 192 rows of 256 pixels each. One of 16 colours can be selected for each pixel.

The 64KB video memory is subdivided into 2 pages where page 0 occupies video memory addresses from 0 to 0x7FFF and page 1 occupies video memory addresses from 0x8000 to 0xFFFF. Within each of these regions, the memory is organized as shown on the following diagram:

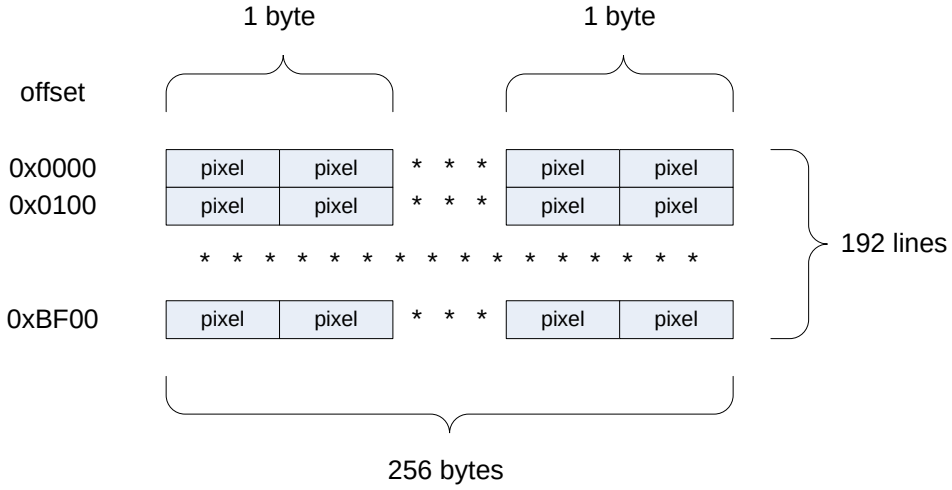


The video page is organized as 192 consecutive blocks of 128 bytes, each describing one line of 256 pixels in a top-to-bottom line order. Within each line there are 128 bytes, each representing two pixels in a left-to-right order. Within each byte, the upper 4 bits specify the color of the leftmost pixel of the pair and the lower 4 bits specify the color of the rightmost pixel of the pair.

3.2.2.6 Graphical 512*192

In this video mode, a VDS1 display presents 192 rows of 512 pixels each. One of 16 colours can be selected for each pixel.

The 64KB video memory is all covered by 1 video page. Within that page, the memory is organized as shown on the following diagram:

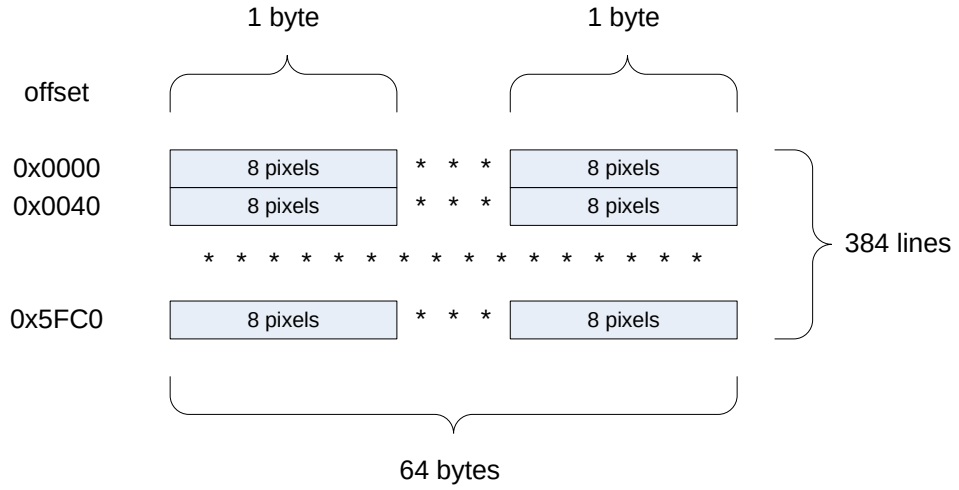


The video page is organized as 192 consecutive blocks of 256 bytes, each describing one line of 512 pixels in a top-to-bottom line order. Within each line there are 256 bytes, each representing two pixels in a left-to-right order. Within each byte, the upper 4 bits specify the color of the leftmost pixel of the pair and the lower 4 bits specify the color of the rightmost pixel of the pair.

3.2.2.7 Graphical 512*384

In this video mode, a VDS1 display presents 384 rows of 512 pixels each. One of 2 colours can be selected for each pixel.

The 64KB video memory is subdivided into 2 pages where page 0 occupies video memory addresses from 0 to 0x7FFF and page 1 occupies video memory addresses from 0x8000 to 0xFFFF. Within each of these regions, the memory is organized as shown on the following diagram:



The video page is organized as 384 consecutive blocks of 64 bytes, each describing one line of 512 pixels in a top-to-bottom line order. Within each line there are 64 bytes, each representing eight pixels in a left-to-right order. Within each byte, the highest bit specifies the color of the leftmost pixel of the eight and the lowest bit specifies the color of the rightmost pixel of the eight.

3.2.2.8 Color codes

In 16-color mode, the color codes map to the specified RGB values (assuming R, G and B dimension of a color are all in range [0..255]:

Code	Red	Green	Blue
0	0x00	0x00	0x00
1	0x00	0x00	0x80
2	0x00	0x80	0x00
3	0x00	0x80	0x80
4	0x80	0x00	0x00
5	0x80	0x00	0x80
6	0x80	0x80	0x00
7	0xC0	0xC0	0xC0
8	0x80	0x80	0x80
9	0x00	0x00	0xFF

10	0x00	0xFF	0x00
11	0x00	0xFF	0xFF
12	0xFF	0x00	0x00
13	0xFF	0x00	0xFF
14	0xFF	0xFF	0x00
15	0xFF	0xFF	0xFF

In 2-color mode, the color codes map to the specified RGB values (assuming R, G and B dimension of a color are all in range [0..255]:

Code	Red	Green	Blue
0	0x00	0x00	0x00
1	0xFF	0xFF	0xFF

3.2.3 Invisible video memory

Given the description of the video modes supported by VDS1, one might have noticed that in each video mode there are areas of the video memory that never contribute to the information visible on display. For example, if video page 0 is displayed in video mode 0 (textual 32*24), then bytes at offsets 0x0600..0x07FF within the video page memory do not correspond to any characters that are displayed.

The VDS1 controller utilizes the last 1536 or 2048 (depending on video mode) bytes of the video memory (video memory addresses 0xFA00..0xFFFF or 0xF800..0xFFFF, respectively) for the use by character generator (note that this restricts the number of usable video pages in video modes 0/2 and 1/3 to 31 and 15 respectively, as in both modes the last video page partially overlaps with the video memory area used by character generator). All other unused portions of the video memory can be used at the programmer's discretion.

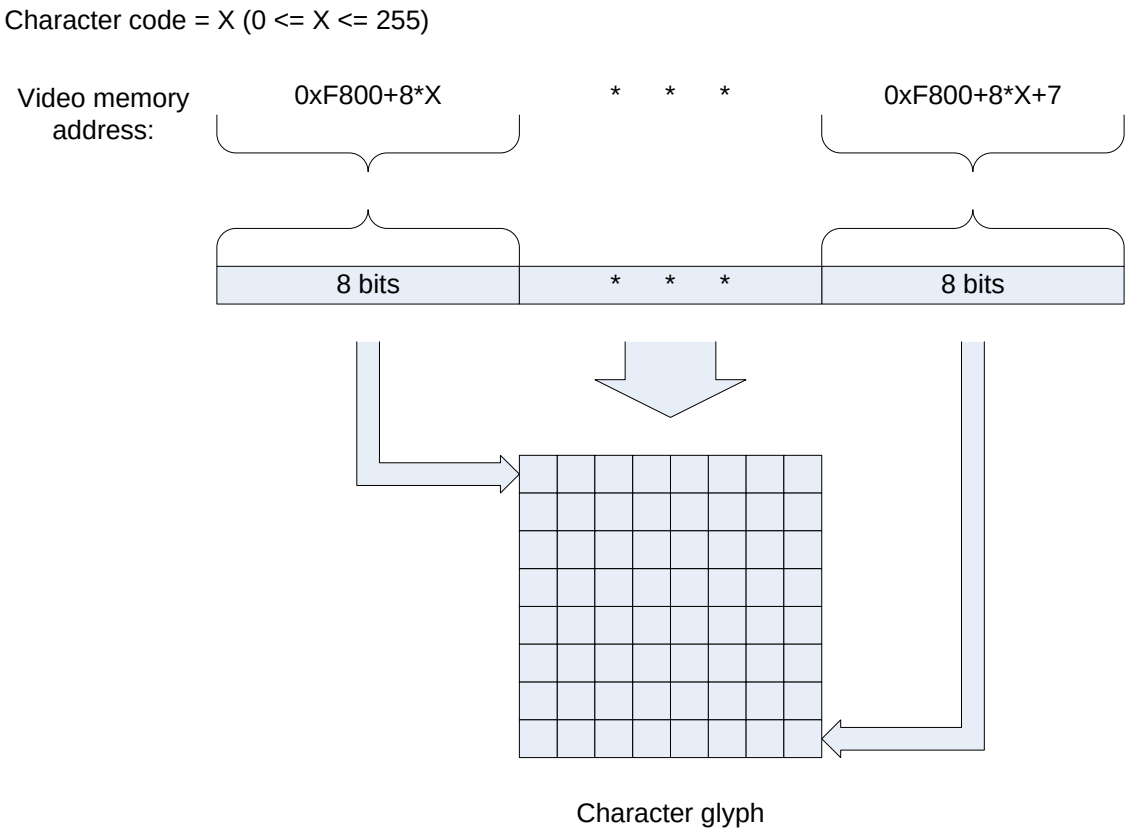
3.2.4 Character generator

In graphical modes, bits of the video memory correspond directly to pixels presented by VDS1 displays. However, in textual mode the video memory contains character codes, which must be translated into pixels by the VDS1 controller. This translation is performed by the controller's character generator – a hardware component that uses the contents of the video memory to convert character data into pixels that can be presented by VDS1 displays.

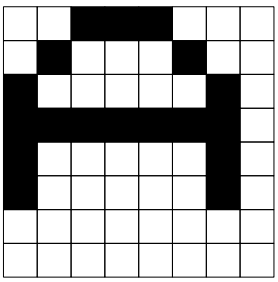
3.2.4.1 32*24 and 64*24

In these video modes, the character generator converts 8-bit character codes into 8*8 blocks of pixels representing character glyphs. The conversion is governed by the character table stored in the video memory at addresses 0xF800..0xFFFF. The said character table is organized as 256 consecutive blocks of 8 bytes, each describing the glyph for the corresponding character code.

The following diagram illustrates the process of translating a character code into a character glyph:



Each of the 8 bytes defining a single character specifies one row of 8 pixels, with the highest bit of a byte corresponding to the leftmost pixel and the lowest bit of the byte corresponding to the rightmost pixel of a row. The 8 lines correspond to 8 bytes in a top-to-bottom order, so, for example, a sequence of the 8 bytes {0x38, 0x44, 0x82, 0xFE, 0x82, 0x82, 0x00, 0x00} defines the character glyph for a Latin capital character ‘A’:

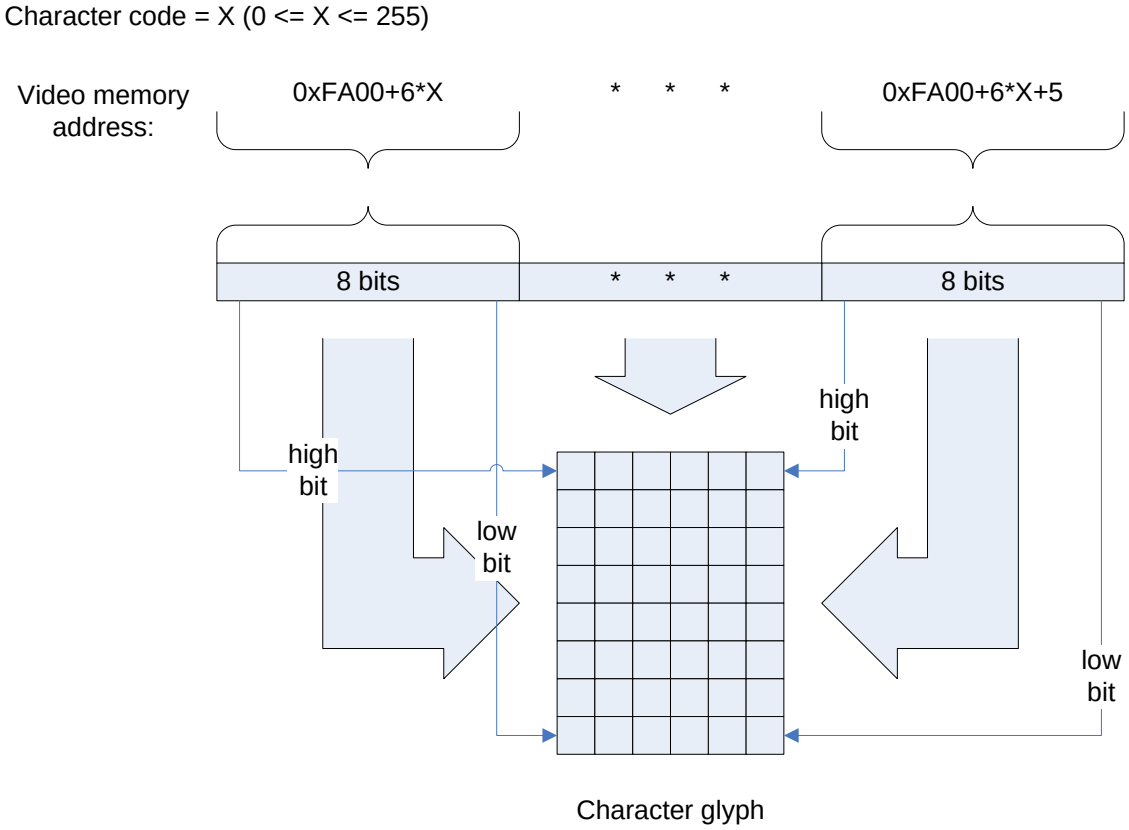


Naturally, the main power of a character generator lies in the fact that character glyphs are themselves stored in video memory, which allows defining arbitrary character glyphs by simply modifying the character table. Although restricted to 8-bit character sets, this feature allows any 8–bit character set to be properly displayed in character mode.

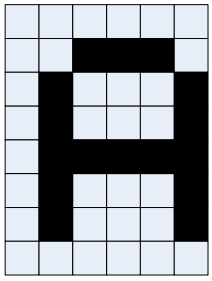
3.2.4.2 40*24 and 80*24

In these video modes, the character generator converts 8-bit character codes into 6*8 blocks of pixels representing character glyphs. The conversion is governed by the character table stored in the video memory at addresses 0xFA00..0xFFFF. The said character table is organized as 256 consecutive blocks of 6 bytes, each describing the glyph for the corresponding character code.

The following diagram illustrates the process of translating a character code into a character glyph:



Each of the 6 bytes defining a single character specifies one column of 8 pixels, with the highest bit of a byte corresponding to the topmost pixel and the lowest bit of the byte corresponding to the bottom pixel of a column. The 6 columns correspond to 6 bytes in a left-to-right order, so, for example, a sequence of the 6 bytes {0x00, 0x3E, 0x48, 0x48, 0x48, 0x3E} defines the character glyph for a Latin capital character ‘A’:



Naturally, the main power of a character generator lies in the fact that character glyphs are themselves stored in video memory, which allows defining arbitrary character glyphs by simply modifying the character table. Although restricted to 8-bit character sets, this feature allows any 8–bit character set to be properly displayed in character mode.

3.2.5 Refreshing the display

The contents of a VDS1 display is refreshed at a predefined intervals (usually 30 or more times per second), with the VDS1 controller providing information about what pixels to display. Therefore, all changes to the contents of a video memory's active page are propagated to the display as soon as the next refresh occurs. If a long sequence of writes to the video memory's active page does not finish until the next refresh cycle, the display will present the partially updated picture.

In graphical modes (except that with only 1 video page) it is possible to avoid flicker entirely by writing data to an inactive video page and, when the writing is complete, making the page active. The same works for textual modes as long as the character table remains unmodified. If, however, a character table is in the middle of being modified when a display refresh occurs, the picture presented on the display may display some characters using the already-modified character glyphs and some other characters with character glyphs that are yet to be modified; this may result in an occasional flicker when the character table is changed.

In the 512*384 graphical video mode there is only one display page, so double-buffering is not an option. Therefore, it is only recommended to use this video mode to present information that requires the best possible resolution but changes infrequently.

3.3 Public controller registers

This section describes public registers of a VDS1 controller. These registers are used for communications between the VDS1 controller and the main processing core.

3.3.1 STATE

The `STATE` register of a VDS1 controller is a 8-bit read-only register that reflects the current state of the controller. Any attempts of the main processing core to change the value of this register are ignored.

Individual bits within the `STATE` register have the following meaning:

Bit	Mask	Meaning
`ON`	`0x01`	1 if the VDS1 controller is operational, 0 otherwise.
`COMMAND_READY`	`0x02`	1 if the controller is ready to accept a command byte, 0 otherwise. Any attempt to write to a `COMMAND` register when this bit is 0 is ignored.
`DATA_IN_READY`	`0x04`	1 if the value read from `DATA` register is valid,

		0 otherwise.
DATA_OUT_READY	0x08	1 if the controller is ready to accept a data byte, 0 otherwise. Any attempt to write to a DATA register when this bit is 0 is ignored.
COMMAND_IN_PROGRESS	0x10	1 iff the controller currently has a command in progress; 0 otherwise. Waiting for this bit to become 0 is a standard way of ensuring that the VDS1 controller has finished with the previous command. If a command requires arguments or yields results, this bit will not become 0 until all command arguments are supplied, the command is carried out and all results are read.
-	0xE0	Unused bits; read as unpredictable.

3.3.2 COMMAND

The COMMAND register of a VDS1 controller is a 8-bit write-only register used for issuing commands to the controller. Any attempts of the main processing core to read the value of this register return an unpredictable byte.

Some commands may take a while to carry out; at other times the main processing core may be so much faster than the VDS1 controller that the latter is unable to carry out commands at the speed required by the main processing core. To allow for synchronization, several means are provided:

- The COMMAND_READY bit of the STATE register is set to 1 when the VDS1 controller is ready for the next command. When the command is received, and while it is executed, this bit reads as 0, thus signalling the main processing core to wait.
- A VDS1 controller may be instructed to issue an I/O interrupt each time it becomes ready for the next command. This happens every time the COMMAND_READY bit of the STATE register changes its value from 0 to 1.

A typical command will be represented by a command code byte written to the COMMAND register, optionally followed by one or more command argument bytes written to the DATA register, then optionally followed by one or more result bytes read from the DATA register.

3.3.3 DATA

The DATA register of a VDS1 controller is a 8-bit read-write register used for providing arguments to commands and reading results of command execution.

When providing arguments for a command, the main processing core must wait until the VDS1 controller is ready to accept each of the argument bytes, To allow for synchronization, several means are provided:

- The DATA_OUT_READY bit of the STATE register is set to 1 when the VDS1 controller is ready for the next byte to be written to the DATA register. When the data byte is received, and while it is processed by the controller, this bit reads as 0, thus signalling the main processing core to wait.
- A VDS1 controller may be instructed to issue an I/O interrupt each time it becomes ready for the next data byte to be written to the DATA register. This happens every time the DATA_OUT_READY bit of the STATE register changes its value from 0 to 1.

When reading results of a command execution, the main processing core must wait until the VDS1 controller is ready to provide each of the result bytes, To allow for synchronization, several means are provided:

- The DATA_IN_READY bit of the STATE register is set to 1 when the VDS1 controller is ready for the next result byte to be read from the DATA register. After the data byte is read, and while the next one is prepared by the controller, this bit reads as 0, thus signalling the main processing core to wait.
- A VDS1 controller may be instructed to issue an I/O interrupt each time a next result byte becomes ready to be read from the DATA register. This happens every time the DATA_IN_READY bit of the STATE register changes its value from 0 to 1.

3.3.4 Command pre-emption

The *command pre-emption* is a VDS1 controller feature that allows incomplete commands to be interrupted. It works in two contexts:

- If a command requiring one or more arguments is issued (written to the COMMAND register) but not yet all of the command arguments have been provided (via the DATA register), another command can be issued by writing to the COMMAND register again. This has the effect of discarding the first, partially-received, command and starting with a new one.
- If a command has been executed and one or more result bytes must now be read (from the DATA register), another command can be issued by writing to the COMMAND register again. This has the effect of discarding the yet-unread results of the first command and starting with a new one.

To achieve both goals, the COMMAND_READY bit of the STATE register becomes 1 as soon as the next command is received, even before any of the command's arguments are read. This allows commands (such as controller reset, compartment reset, etc.) to be issued regardless of what other commands may already be partially underway.

3.4 Private controller registers

This section describes private registers of a VDS1 controller. These registers are mostly used for device-specific operations; each compartment within a VDS1 controller will normally have its own private copy of these registers (unless specified otherwise). As far as the main processing core is concerned, there is only one set of private registers; however, the mapping of these private registers onto the actual

registers of VDS1 controller changes each time a new compartment is selected as current.

3.4.1 INTERRUPT_MASK

The INTERRUPT_MASK register of a VDS1 controller's compartment is a 8-bit read/write register used by the VDS1 controller to determine which state change events cause I/O interrupts to be sent to the main processing core. Individual bits within this register have the following meaning:

Bit	Mask	Meaning
COMMAND_READY_ON	0x01	When this bit is 1, a COMMAND_READY_ON I/O interrupt occurs each time the COMMAND_READY bit of the STATE register changes from 0 to 1.
DATA_IN_READY_ON	0x02	When this bit is 1, a DATA_IN_READY_ON I/O interrupt occurs each time the DATA_IN_READY bit of the STATE register changes from 0 to 1.
DATA_OUT_READY_ON	0x04	When this bit is 1, a DATA_OUT_READY_ON I/O interrupt occurs each time the DATA_OUT_READY bit of the STATE register changes from 0 to 1.
COMMAND_IN_PROGRESS_OFF	0x08	When this bit is 1, a COMMAND_IN_PROGRESS_OFF I/O interrupt occurs each time the COMMAND_IN_PROGRESS bit of the STATE register changes from 1 to 0.

Regardless of the actual number of available compartments, a VDS1 controller has only one INTERRUPT_MASK register.

When a new value is written into the INTERRUPT_MASK register, it takes effect from the moment the COMMAND_IN_PROGRESS bit of the STATE register goes from 1 to 0, thus signifying that interrupt mask modification has been completed.

3.4.2 COMPARTMENT

The COMPARTMENT register of a VDS1 controller is a 8-bit read/write register used by the VDS1 controller to track the currently selected compartment. Reading from this register yields the address of the current compartment; writing to this register causes a new compartment to be selected as current.

Regardless of the actual number of available compartments, a VDS1 controller has only one COMPARTMENT register.

3.4.3 MODE

The MODE register of a VDS1 controller's compartment is a 8-bit read/write register that specifies the current video mode used by the compartment in question.

Each compartment within the VDS1 controller has its own, private, copy of this register.

3.4.4 PAGE

The PAGE register of a VDS1 controller's compartment is a 8-bit read/write register that specifies the current video page of the by the compartment in question.

Each compartment within the VDS1 controller has its own, private, copy of this register.

3.4.5 SOURCE

The SOURCE register of a VDS1 controller's compartment is a 16-bit read/write register used by the VDS1 controller to specify the source address of a data transfer operation within the compartment's video memory.

Each compartment within the VDS1 controller has its own, private, copy of this register.

3.4.6 DESTINATION

The DESTINATION register of a VDS1 controller's compartment is a 16-bit read/write register used by the VDS1 controller to specify the destination address of a data transfer operation within the compartment's video memory.

Each compartment within the VDS1 controller has its own, private, copy of this register.

3.4.7 LENGTH

The LENGTH register of a VDS1 controller's compartment is a 16-bit read/write register used by the VDS1 controller to specify the length of the data block transferred to/from/within the compartment's video memory.

Each compartment within the VDS1 controller has its own, private, copy of this register.

3.5 Controller commands

This section describes commands that can be issued to a VDS1 controller.

3.5.1 Command description format

A description of each VDS1 controller command consists of:

- Command – a byte that needs to be written to the `COMMAND` register (after making sure `COMMAND_READY` is 1, i.e. the `COMMAND` register is ready for the next command).
- Arguments – a sequence of bytes that needs to be written to the DATA register (after making sure `DATA_OUT_READY` is 1 before writing each byte, i.e. the `DATA` register is ready for the next argument byte).
- Results – a sequence of bytes that needs to be read from the DATA register (after making sure `DATA_IN_READY` is 1 before reading each byte, i.e. the `DATA` register is ready with the next argument byte).

3.5.2 Address wraparound

Many controller commands cause controller register(s) to be incremented; other commands operant on continuous blocks of the controller's video memory. In all cases, video memory addresses are calculated modulo 2^{16}. Specifically:

- If a controller's address register (e.g. `SOURCE` or `DESTINATION`) is 0xFFFF before increment by 1, its value will be 0x0000 after the increment.
- If a block of video memory is specified in a way such that `<block's start address>` + `<block size>` > 0x10000, the block wraps around to address 0x0000 after address 0xFFFF.

3.5.3 Reset

Command:	`0x00`
Arguments:	None
Results:	None

This command, when issued, causes the full reset of a VDS1 controller. Specifically:

- All video memory of all existing compartments is initialized with 0x00 bytes.
- Compartment 0x00 is selected as active.
- Video mode 0 (textual, 32*24) is set for all compartments and video page 0 selected as active.
- `SOURCE`, `DESTINATION` and `LENGTH` registers of each compartment are set to 0.

3.5.4 ResetCompartment

Command:	`0x01`
Arguments:	None
Results:	None

This command, when issued, causes the full reset of the current compartment of a VDS1 controller. Specifically:

- All video memory of the current compartment is initialized with 0x00 bytes.
- Video mode 0 (textual, 32*24) is set for the current compartment and video page 0 selected as active.

- SOURCE, DESTINATION and LENGTH registers of the current compartment are set to 0.

3.5.5 GetCapacity

Command:	0x02
Arguments:	None
Results:	<compartment capacity>

This command, when issued, returns the VDS1 controller's compartment capacity (i.e. the number of compartments supported by the controller, e.g. 0x00 = 1 compartment only; 0x01 = 2 compartments, etc.)

3.5.6 GetInterruptMask

Command:	0x03
Arguments:	None
Results:	<controller's interrupt mask>

This command, when issued, returns the value of the VDS1 controller INTERRUPT_MASK register.

3.5.7 SetInterruptMask

Command:	0x04
Arguments:	<controller's interrupt mask>
Results:	None

This command, when issued, sets the value of the VDS1 controller INTERRUPT_MASK register. The new interrupt mask takes effect once the command has been fully processed (i.e. from the moment the COMMAND_IN_PROGRESS bit of the STATE register goes from 1 to 0, thus signifying that interrupt mask modification has been completed) or once a new command is issued, whichever happens first.

3.5.8 SenseDisplay

Command:	0x05
Arguments:	<compartment number>
Results:	<display flags>

This command, when issued, checks if a display is attached to the specified compartment of the VDS1 controller. The <display flags> byte returned by the command is a combination of the following bit flags:

Bit	Mask	Meaning
0	0x01	1 if a display device is connected to the specified

		compartment, 0 otherwise.
1	0x02	1 if a display device connected to the specified compartment is powered on, 0 if it is powered off or does not exist (i.e. is not connected).
2..7	0xFC	Reserved for future use, always zero.

3.5.9 GetCurrentCompartment

Command:	0x10
Arguments:	None
Results:	<current compartment number>

This command, when issued, returns the address of the compartment selected as current.

3.5.10 SetCurrentCompartment

Command:	0x11
Arguments:	<new current compartment>
Results:	None

This command, when issued, causes the compartment with the specified address to become current. If the compartment with the specified address does not exist, the command is ignored. There is no error return; to check whether compartment change was successful the `GetCurrentCompartment` command can be issued after `SetCurrentCompartment` in order to check whether the current compartment has actually been changed.

3.5.11 GetVideoMode

Command:	0x20
Arguments:	None
Results:	<current compartment's video mode>

This command, when issued, returns the `MODE` register of the current compartment.

3.5.12 SetVideoMode

Command:	0x21
Arguments:	<new video mode for the current compartment>
Results:	None

This command, when issued, causes the current compartment to switch to a new video mode (by changing its `MODE` register) and select video page 0 as the current video

page. The command is ignored if the new video mode code is unsupported (i.e. is not in range 0..6).

3.5.13 GetVideoPage

Command:	0x22
Arguments:	None
Results:	<current compartment's video mode>

This command, when issued, returns the PAGE register of the current compartment.

3.5.14 SetVideoPage

Command:	0x23
Arguments:	<new video page for the current compartment>
Results:	None

This command, when issued, causes the current compartment to select a new video page as the current video page (by changing its PAGE register). The command is ignored if the new video page number is unsupported (i.e. is not in the supported range of video pages for the compartment's current video mode).

3.5.15 GetSourceAddress

Command:	0x30
Arguments:	None
Results:	<high byte of the source address> <low byte of the source address>

This command, when issued, causes the high and low bytes of the current compartment's SOURCE register to be returned in this order.

3.5.16 SetSourceAddress

Command:	0x31
Arguments:	<new high byte of the source address> <new low byte of the source address>
Results:	None

This command, when issued, causes the contents of the SOURCE register of the current compartment to be replaced with the two specified bytes.

3.5.17 GetDestinationAddress

Command:	0x32
Arguments:	None
Results:	<high byte of the destination address>

	<low byte of the destination address>

This command, when issued, causes the high and low bytes of the current compartment's `DESTINATION` register to be returned in this order.

3.5.18 SetDestinationAddress

Command:	`0x33`
Arguments:	<new high byte of the destination address> <new low byte of the destination address>
Results:	None

This command, when issued, causes the contents of the `DESTINATION` register of the current compartment to be replaced with the two specified bytes.

3.5.19 GetLength

Command:	`0x34`
Arguments:	None
Results:	<high byte of the length> <low byte of the length>

This command, when issued, causes the high and low bytes of the current compartment's `LENGTH` register to be returned in this order.

3.5.20 SetLength

Command:	`0x35`
Arguments:	<new high byte of the length> <new low byte of the length>
Results:	None

This command, when issued, causes the contents of the `LENGTH` register of the current compartment to be replaced with the two specified bytes.

3.5.21 Read

Command:	`0x40`
Arguments:	None
Results:	<byte read from the video memory>

This command, when issued, causes a byte to be read from the address `SOURCE` of the current compartment's video memory and returned. The `SOURCE` register remains unaffected.

3.5.22 ReadAndAdvance

Command:	`0x41`
Arguments:	None
Results:	<byte read from the video memory>

This command, when issued, causes a byte to be read from the address `SOURCE` of the current compartment's video memory and returned. The `SOURCE` register is incremented by 1 when the command is executed, regardless of whether the actual result byte is read by the main processing core or not.

3.5.23 Write

Command:	`0x50`
Arguments:	<byte to write to the video memory>
Results:	None

This command, when issued, causes a byte to be written to the address `DESTINATION` of the current compartment's video memory. The `DESTINATION` register remains unaffected.

3.5.24 WriteAndAdvance

Command:	`0x51`
Arguments:	<byte to write to the video memory>
Results:	None

This command, when issued, causes a byte to be written to the address `DESTINATION` of the current compartment's video memory. The `DESTINATION` register is incremented by 1 after all command arguments have been received.

3.5.25 WriteWithMask

Command:	`0x52`
Arguments:	<mask> <byte to write to the video memory>
Results:	None

This command, when issued, causes a masked byte to be written to the address `DESTINATION` of the current compartment's video memory. The new value of the video memory's byte at the `DESTINATION` address is `(<old value> & ~<mask>) | (<byte written> & <mask>)`, i.e. all bits of the written byte for which the corresponding bit of the mask is 1 are written to the video memory, whereas all remaining bits of the same byte in video memory remain the same. For example, in order to replace the color, but not the character, of a position in one of the textual modes, the mask 0x0F shall be used. The `DESTINATION` register remains unaffected.

3.5.26 WriteWithMaskAndAdvance

Command:	`0x53`
Arguments:	<mask> <byte to write to the video memory>
Results:	None

This command, when issued, causes a masked byte to be written to the address `DESTINATION` of the current compartment's video memory. The new value of the video memory's byte at the `DESTINATION` address is `(<old value> & ~<mask>) | (<byte written> & <mask>)`, i.e. all bits of the written byte for which the corresponding bit of the mask is 1 are written to the video memory, whereas all remaining bits of the same byte in video memory remain the same. For example, in order to replace the color, but not the character, of a position in one of the textual modes, the mask 0x0F shall be used. The `DESTINATION` register is incremented by 1 after all command arguments have been received.

3.5.27 Fill

Command:	`0x54`
Arguments:	<byte to fill the video memory with>
Results:	None

This command, when issued, causes `LENGTH` sequential bytes to be written to the current compartment's video memory starting at the `DESTINATION` address. The `DESTINATION` register remains unaffected.

3.5.28 FillWithMask

Command:	`0x55`
Arguments:	<mask> <byte to fill the video memory with>
Results:	None

This command, when issued, causes `LENGTH` sequential bytes to be written to the current compartment's video memory starting at the `DESTINATION` address. For each byte stored, only those bits of the source byte for which the corresponding mask bits are 1 replace the corresponding bits in the destination bytes; all destination bits for which the corresponding bit of the mask is 0 remain unaffected. The `DESTINATION` register remains unaffected.

3.5.29 Move

Command:	`0x60`
Arguments:	None
Results:	None

This command, when issued, causes a block of LENGTH video memory bytes to be moved from SOURCE to DESTINATION video memory address. If the two blocks overlap, the effect is as if the blocks were moved byte-by-byte starting with the lower addresses. The SOURCE, DESTINATION and LENGTH registers all remain unaffected.

3.5.30 MoveBackward

Command:	0x61
Arguments:	None
Results:	None

This command, when issued, causes a block of LENGTH video memory bytes to be moved from SOURCE to DESTINATION video memory address. If the two blocks overlap, the effect is as if the blocks were moved byte-by-byte starting with the higher addresses. The SOURCE, DESTINATION and LENGTH registers all remain unaffected.

3.5.31 MoveWithMask

Command:	0x62
Arguments:	<mask>
Results:	None

This command, when issued, causes a block of LENGTH video memory bytes to be moved from SOURCE to DESTINATION video memory address using the specified mask. If the two blocks overlap, the effect is as if the blocks were moved byte-by-byte starting with the lower addresses. For each byte moved, only those bits of the source byte for which the corresponding mask bits are 1 replace the corresponding bits in the destination bytes; all destination bits for which the corresponding bit of the mask is 0 remain unaffected. The SOURCE, DESTINATION and LENGTH registers all remain unaffected.

3.5.32 MoveBackwardWithMask

Command:	0x63
Arguments:	<mask>
Results:	None

This command, when issued, causes a block of LENGTH video memory bytes to be moved from SOURCE to DESTINATION video memory address using the specified mask. If the two blocks overlap, the effect is as if the blocks were moved byte-by-byte starting with the upper addresses. For each byte moved, only those bits of the source byte for which the corresponding mask bits are 1 replace the corresponding bits in the destination bytes; all destination bits for which the corresponding bit of the mask is 0 remain unaffected. The SOURCE, DESTINATION and LENGTH registers all remain unaffected.

3.6 Interrupts

The following table summarizes I/O interrupts that can be raised by a VDS1 controller:

Name	Code	Occurs when…
COMMAND_READY_ON	0x01	The COMMAND_READY_ON I/O bit of the INTERRUPT_MASK register is 1 and the COMMAND_READY bit of the STATE register changes from 0 to 1.
DATA_IN_READY_ON	0x02	The DATA_IN_READY_ON bit of the INTERRUPT_MASK register is 1 and the DATA_IN_READY bit of the STATE register changes from 0 to 1.
DATA_OUT_READY_ON	0x03	The DATA_OUT_READY_ON bit of the INTERRUPT_MASK register is 1 and the DATA_OUT_READY bit of the STATE register changes from 0 to 1.
COMMAND_IN_PROGRESS_OFF	0x04	The COMMAND_IN_PROGRESS_OFF bit of the INTERRUPT_MASK register is 1 and the COMMAND_IN_PROGRESS bit of the STATE register changes from 1 to 0.

3.6.1 Pending interrupts

A situation may occur when a VDS1 controller is ready to issue an I/O interrupt before the previous I/O interrupt issued by the same VDS1 controller has been processed by the main processing core.

In this situation, the interrupt remains pending within the I/O controller. As soon as the previous interrupt has been processed, the pending interrupt is issued.

4 Appendix A: GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in

part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- **A.** Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- **B.** List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- **C.** State on the Title page the name of the publisher of the Modified Version, as the publisher.
- **D.** Preserve all the copyright notices of the Document.
- **E.** Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- **F.** Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- **G.** Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- **H.** Include an unaltered copy of this License.
- **I.** Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- **J.** Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- **K.** For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

- **L.** Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- **M.** Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- **N.** Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- **O.** Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.