THE MOS 6567/6569 VIDEO CONTROLLER (VIC-II) AND ITS APPLICATION IN THE COMMODORE 64

by Christian Bauer
bauec002@goofy.zdv.uni-mainz.de
28. August 1996

Post processing by Jan Klingel, 2020
https://janklingel.de

CONTENTS

1 INTRODUCTION

This paper is an attempt to summarize the results of various people's examinations of the graphics chip "6567/6569 Video Interface Controller[1] (VIC-II)" (simply called "VIC" in the following) used in the legendary Commodore 64, and to provide a complete reference to its specified and unspecified properties. It is primarily intended for C64 programmers and authors of C64 emulators, but should also be interesting to "outsiders" interested in hardware design and programming and hacking a computer up to the last bits. For this purpose, some general information (e.g. the C64 memory map) already known to experienced C64 programmers has been included as well.

The description of the unspecified properties is based on tests done by Marko Mäkelä, Andreas Boose, Pasi Ojala, Wolfgang Lorenz and myself (not to mention numerous others) during the last years. It also covers internal registers and workings of the VIC. As no schematics of the VIC are available it can of course only be speculative, but in all cases a model has been chosen that explains the observed phenomena with the minimally required circuitry. E.g. for the video matrix counter (VC), a model with two simple counters was given preference to a more elaborate one with a +40 adder.

Although some measurements have been done with an oscilloscope directly on the chip, most insights are based on test programs on the C64 and by comparing them with the implementation in single cycle emulations like "Frodo SC".

[1] In an internal document from Commodore Semiconductor Group, dated 1973, the 6567 is called a "video interface chip". [6]

2 THE ARCHITECTURE OF THE COMMODORE 64

This chapter gives an overview of the basic hardware architecture of the C64 and the integration of the VIC into the system.

2.1 OVERVIEW

The C64 basically consists of the following units:

- 6510 8-bit microprocessor
- 6567/6569 VIC-II graphics chip
- 6581 SID sound chip
- Two 6526 CIA I/O chips
- 64KB DRAM (64K*8 bit) main memory
- 0.5KB SRAM (1K*4 bit) Color RAM
- 16KB ROM (16K*8 bit) for operating system and BASIC interpreter
- 4KB ROM (4K*8 bit) character generator

Most chips are manufactured in NMOS[2] technology.

2.2 6510 PROCESSOR

The 6510 microprocessor [1] has an 8-bit data bus and a 16-bit address bus and is object code compatible with the famous 6502. It has two external interrupt inputs (one maskable (IRQ) and one non-maskable (NMI)) and as a special feature a 6-bit wide bidirectional I/O port. It is clocked at 1MHz in the C64.

Important signals:

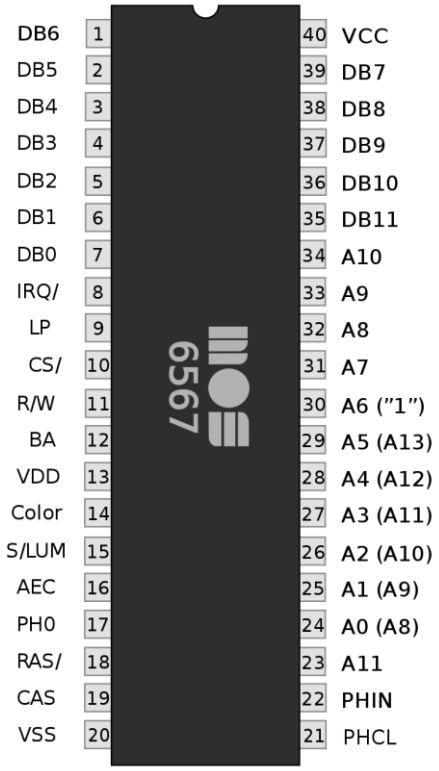
ø2	Processor clock output This clock signal is the reference for the complete bus timing. Its frequency is 1022.7 kHz (NTSC models) or 985.248 kHz (PAL models). One period of this signal corresponds to one clock cycle consisting of two phases: ø2 is low in the first phase and high in the second phase (hence the name 'ø2' for

[2] N-type metal-oxide-semiconductor

	"phase 2"). The 6510 only accesses the bus in the second clock phase, the VIC normally only in the first phase.
R/W	This signal flags a read (R/W high) or write (R/W low) access.
IRQ	If this input is held on low level, an interrupt sequence is triggered unless interrupts are masked with the interrupt mask bit in the status register. The interrupt sequence begins after two or more clock cycles at the start of the next instruction. With this pin, the VIC can trigger an interrupt in the processor. Interrupts are only recognized if the RDY line is high.
RDY	If this line is low during a read access, the processor stops with the address lines reflecting the current address being fetched. It is ignored during write accesses. In the C64, RDY is used to stop the processor if the VIC needs additional bus cycles for character pointer and sprite data accesses. It is connected to the BA signal on the VIC.
AEC	This pin tri-states the address lines. This is used for making the processor address bus inactive during VIC accesses. The signal is connected to the AEC output on the VIC.
P0-P5	This is the built-in 6 bit I/O port. Each line can be individually programmed as input or output. A data direction register and a data register are internally mapped to addresses 0 and 1, respectively. You may therefore expect that the processor cannot access the RAM addresses 0 and 1 (as they are overlayed by the I/O port), but more on this later...

2.3 6567/6569 GRAPHICS CHIP

The 656* series graphics chip by MOS Technologies were originally designed to be used in video games and graphics terminals. But as the sales in these markets have been rather poor, Commodore decided to use the chips when they were planning to make their own home computers.



In the C64, the "Video Interface Controller II (VIC-II)" [2] has been used, featuring 3 text based (40x25 characters with 8x8 pixels each) and 2 bitmap based (320x200 pixels) video modes, 8 hardware sprites and a fixed palette of 16 colors. It can manage up to 16KB of dynamic RAM (including the generation of RAS and CAS and the RAM refresh) and also has a light pen input and interrupt possibilities. Two VIC types appear in the C64: The 6567 in NTSC machines and the 6569 in PAL machines. There are several mask steppings of both types, but the differences are mostly neglectable with the exception of the 6567R56A. Newer C64 versions are bearing the functionally equivalent chips 8562 (NTSC) and 8565 (PAL). In the following, only 6567/6569 will be mentioned, but all statements are applicable for the 856* chips. There is also a 6566 designed to be connected to static RAM but this one was never used in C64s.

Important signals[3]:

A0-A13	The 14 bit video address bus used by the VIC to address 16KB of memory. The address bits A0-A5 and A8-A13 are multiplexed in pairs (i.e. A0/A8, A1/A9 etc.) on one pin each. The bits A6-A11 are (additionally) available on separate lines.
D0-D11	A 12 bit wide data bus over which the VIC accesses the memory. The lower 8 bits are connected to the main memory and the processor data bus, the upper 4 bits are connected to a special 4 bit wide static memory (1024 addresses, A0-A9) used for storing color information, the Color RAM.
IRQ	This output is wired to the IRQ input on the processor and makes it possible for the VIC to trigger interrupts. The VIC has four interrupt sources: On reaching a certain raster line (raster interrupt), on the collision of two or more sprites, on the collision of sprites with graphics data and on a negative edge on the light pen input.
BA	With this signal, the VIC indicated that the bus is available to the processor during the second clock phase (ø2 high). BA is normally high as the VIC accesses the bus mostly during the first phase. But for the character pointer and sprite data accesses, the VIC also needs the bus sometimes during the second phase. In this case, BA goes low three cycles before the VIC access. After that, AEC remains low during the second phase and the VIC performs the accesses. Why three cycles? BA is connected to the RDY line of the processor as mentioned, but this line is ignored on write accesses (the CPU can only be interrupted on reads), and the 6510 never does more than three writes in sequence (see [5]).

[3] Picture by Bill Bertram. Common creative license.

AEC	This pin is wired to the processor signal with the same name (see there). It reflects the state of the data and address line drivers of the VIC. If AEC is high, they are in tri-state. AEC is normally low during the first clock phase (ø2 low) and high during the second phase so that the VIC can access the bus during the first phase and the 6510 during the second phase. If the VIC also needs the bus in the second phase, AEC remains low.
LP	This input is intended for connecting a light pen. On a negative edge, the current position of the raster beam is latched to the registers LPX and LPY. As this pin shares a line with the keyboard matrix, it can also be accessed by software.
øIN PHIN	This is the feed for the pixel clock of 8.18 MHz (NTSC) or 7.88 MHz (PAL) that is generated from the crystal frequency. Eight pixels are displayed per bus clock cycle (ø2).
ø0 PHCL	From the pixel clock on øIN, the VIC generates the system (color) clock of 1.023 MHz (NTSC) or 0.985 MHz (PAL) by dividing øIN by eight. It is available on this pin and fed into the processor which in turn generated the signal ø2 from it.

2.4 MEMORY

Three memory areas in the C64 are involved with the graphics:

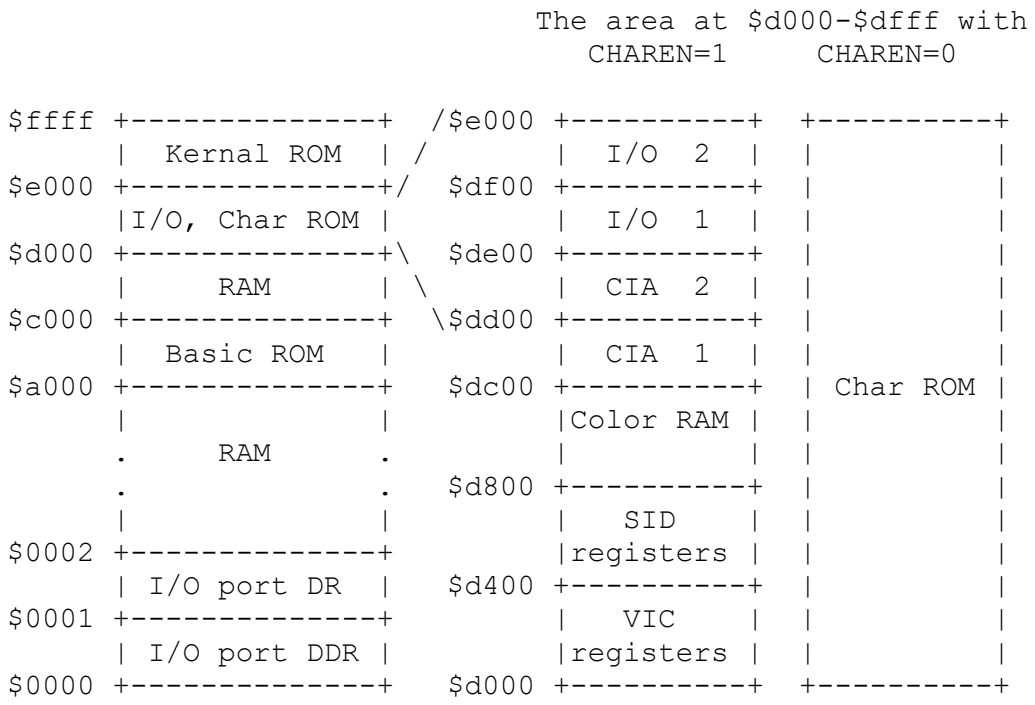
· The 64 KB main memory
· The 1K*4 bit Color RAM
· The 4KB character generator ROM (Char ROM)

In the following two sections it is explained how these memory areas share the address space as seen by the CPU and the VIC. After that, the basics of memory access and DRAM handling are mentioned.

2.4.1 MEMORY MAP AS SEEN BY THE 6510

The 6510 can address 64KB linearly with its 16 address lines. With the aid of a special PAL chip in the C64, many different memory configurations can be used via the 6510 I/O port lines and control lines on the expansion port (see [3]). Only the standard configuration will be discussed here as the other configurations don't change the position of the different areas. They only map in additional areas of the main memory.

This is the memory map as seen by the 6510:



Basically, the 64KB main memory can be accessed in a linear fashion, but they are overlaid by ROM and register areas at several positions. A write access to a ROM area will store the byte in the RAM lying "under" the ROM. The 6510 I/O port is mapped to addresses $0000 (for the data direction register) and $0001 (for the data register).

In the area at $d000-$dfff you can switch between the I/O chip registers and the Color RAM, or the character generator ROM, with the signal CHAREN (which is bit 2 of the 6510 I/O port). The Color RAM is mapped at $d800-$dbff and connected to the lower 4 data bits. The upper 4 bits are open and have "random" values on reading. The two areas named "I/O 1" and "I/O 2" are reserved for expansion cards and also open under normal circumstances. Hence, a read access will fetch "random" values here too (it will be explained in chapter 4 that these values are not really random. Reading from open addresses fetches the last byte read by the VIC on many C64s).

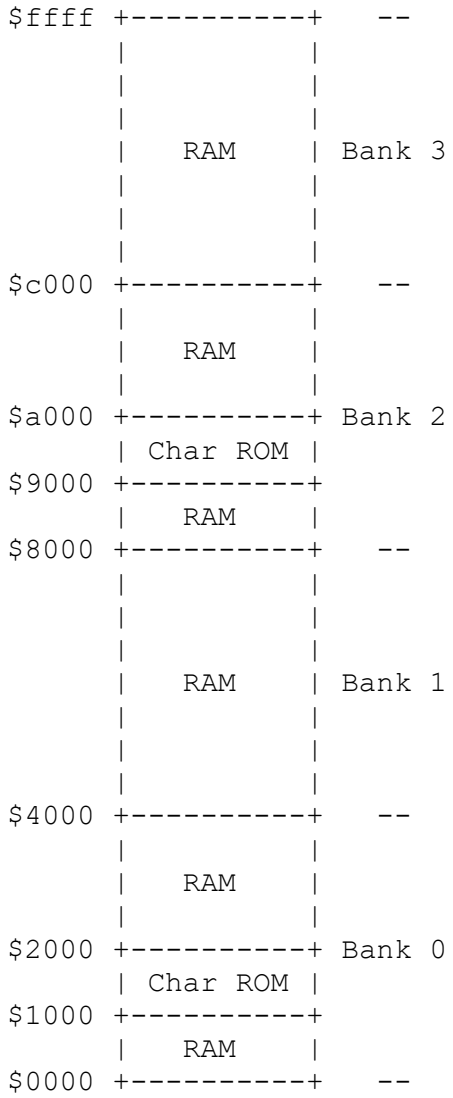
The 47 registers of the VIC are mapped in at $d000. Due to the incomplete address decoding, they are repeated every 64 bytes in the area $d000-$d3ff.

2.4.2 MEMORY MAP AS SEEN BY THE VIC

The VIC has only 14 address lines, so it can only address 16KB of memory. It can access the complete 64KB main memory all the same because the 2 missing address bits are provided

by one of the CIA I/O chips (they are the inverted bits 0 and 1 of port A of CIA 2). With that you can select one of 4 16KB banks for the VIC at a time.

The (extended) memory map as seen by the VIC looks like this:



The Char ROM is mapped in at the VIC addresses $1000-$1fff in banks 0 and 2 (it appears at $9000 in the above diagram, but remember that the VIC doesn't know about the two address bits generated by the CIA. From the VIC's point of view, the Char ROM is at $1000-$1fff also in bank 2).

The attentive reader will already have noticed that the Color RAM doesn't appear anywhere. But as explained earlier, the VIC has a 12-bit data bus of which the upper 4 bits are connected with the Color RAM. Generally speaking, the sole purpose of the upper 4 bits of the VIC data bus is to read from the Color RAM. The Color RAM is addressed by the lower 10 bits of the VIC address bus and is therefore available in all banks at all addresses.

2.4.3 MEMORY ACCESS OF THE 6510 AND VIC

6510 and VIC are both based on a relatively simple hard-wired design. Both chips make a memory access in EVERY clock cycle, even if that is not necessary at all. E.g. if the processor is busy executing an internal operation like indexed addressing in one clock cycle, that really doesn't require an access to memory, it nevertheless performs a read and discards the read byte. The VIC only performs read accesses, while the 6510 performs both reads and writes.

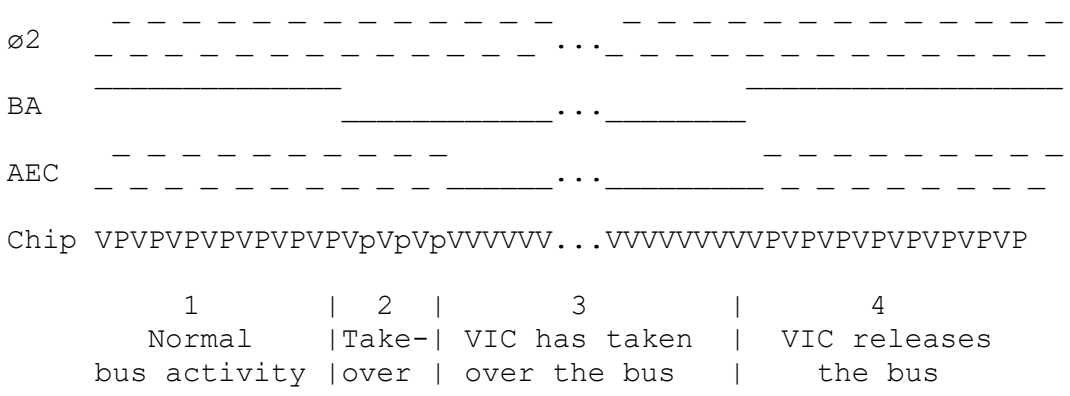
There are no wait states, no internal caches and no sophisticated access protocols for the bus as seen with more modern processors. Every access is done in a single cycle.

The VIC generates the clock frequencies for the system bus and the RAS and CAS signals for accessing the dynamic RAM (for both the processor and the VIC). So it has primary control over the bus and may "stun" the processor sometime or another when it needs additional cycles for memory accesses. Besides this, the VIC takes care of the DRAM refresh by reading from 5 refresh addresses in each raster line.

The division of accesses between 6510 and VIC is basically static: Each clock cycle (one period of the ø2 signal) consists of two phases. The VIC accesses in the first phase (ø2 low), the processor in the second phase (ø2 high). The AEC signal closely follows ø2. That way the 6510 and VIC can both use the memory alternatively without disturbing each other.

However, the VIC sometimes needs more cycles than made available to it by this scheme. This is the case when the VIC accesses the character pointers and the sprite data. In the first case it needs 40 additional cycles, in the second case it needs 2 cycles per sprite. BA will then go low 3 cycles before the VIC takes over the bus completely (3 cycles is the maximum number of successive write accesses of the 6510). After 3 cycles, AEC stays low during the second clock phase so that the VIC can output its addresses.

The following diagram illustrates the process of the take-over:



The line "Chip" designates which chip is just accessing the bus (as said before, there is an access in every cycle). "V" stands for the VIC, "P" for the 6510. The cycles designated with "p" are accesses of the 6510 that are only performed if they are write accesses. The first "p" read access stops the 6510, at least after the third "p" as the 6510 never does more than 3 write accesses in succession. On a "p" read access the processor addresses are still output on the bus because AEC is still high.

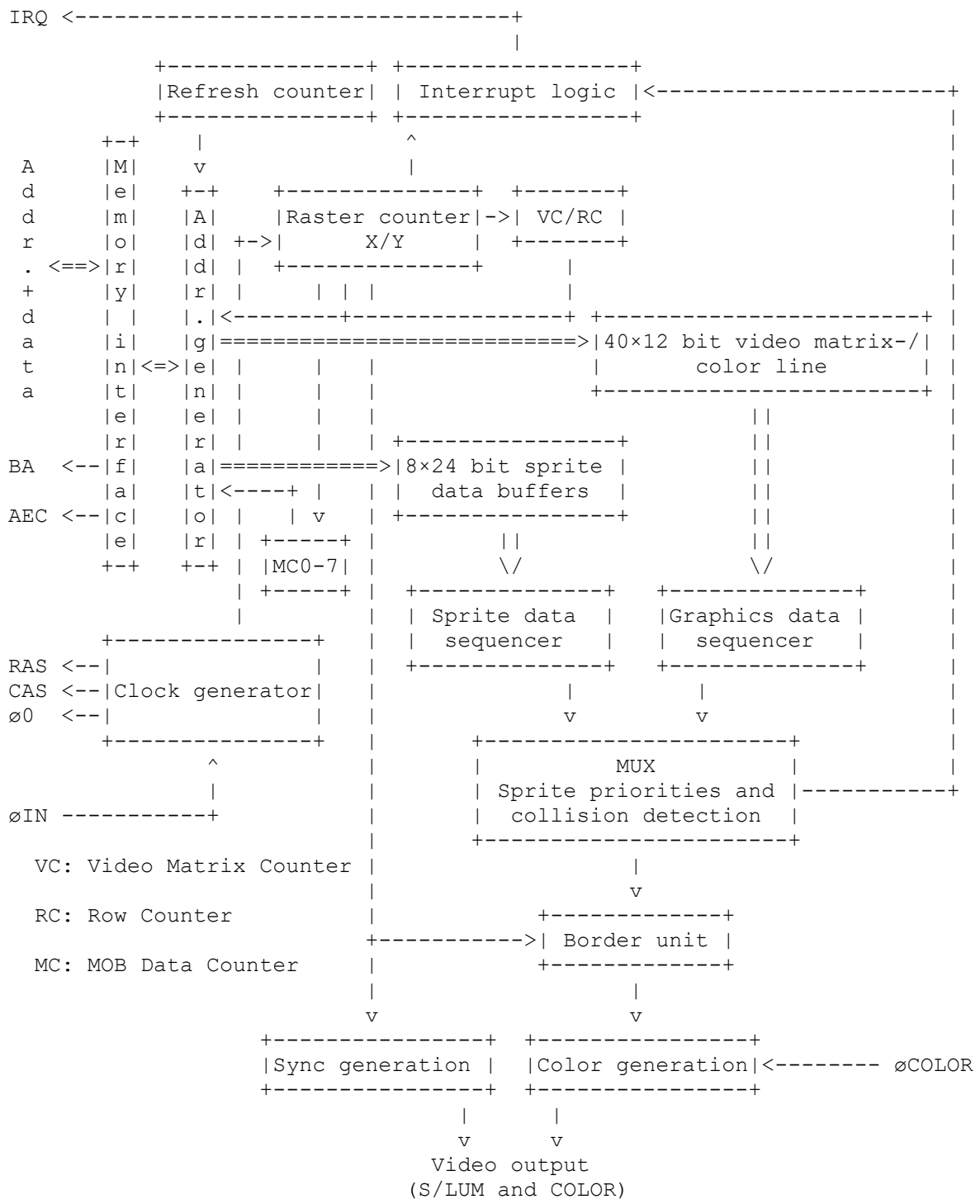
The diagram describes the normal process of a bus take-over. By appropriately modifying the VIC register $d011, it is possible to force a bus take-over at extraordinary times. This is explained in chapter 3 as well as the complete bus timing of a VIC raster line.

3 DESCRIPTION OF THE VIC

This chapter is about the single function units in the VIC, their way of working and their unspecified behavior, and the insights into the internal functions of the VIC that can be gained by that.

3.1 BLOCK DIAGRAM

The following block diagram gives an overview over the internal structure of the VIC and the independently working function units:



The light pen[4] unit is not shown.

As you can see, the "Raster counter X/Y" plays a central role. This is no surprise as the complete screen display and all bus accesses are synchronized by it. It is important to note that the units for display and for the needed memory accesses are separate from each other for the sprites as well as for the graphics. There is a data buffer between the two units that holds the read graphics data and buffers it for the display circuits. In the normal operation of the VIC, the functions of the two units are so closely tied to each other that they appear like a single function block. By appropriate programming, however, you can decouple the circuits and e.g. display graphics without previously having read data (in this case, the data which are still in the buffer are displayed).

3.2 REGISTERS

The VIC has 47 read/write registers for the processor to control its functions:

#	Addr.	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Function
0	$d000	M0X								X coordinate sprite 0
1	$d001	M0Y								Y coordinate sprite 0
2	$d002	M1X								X coordinate sprite 1
3	$d003	M1Y								Y coordinate sprite 1
4	$d004	M2X								X coordinate sprite 2
5	$d005	M2Y								Y coordinate sprite 2
6	$d006	M3X								X coordinate sprite 3
7	$d007	M3Y								Y coordinate sprite 3
8	$d008	M4X								X coordinate sprite 4
9	$d009	M4Y								Y coordinate sprite 4
10	$d00a	M5X								X coordinate sprite 5
11	$d00b	M5Y								Y coordinate sprite 5

[4] The light pen has an optical sensor at its tip that scans the electron beam of raster screens. If the electron beam comes to the point where the light pen touches the screen during line-by-line image formation, the raster signal is converted into an electrical signal by the sensor and the position of the light pen is determined from this signal. With the C64, the light pen must be connected to joystick port 1.

12	$d00c	M6X								X coordinate sprite 6
13	$d00d	M6Y								Y coordinate sprite 6
14	$d00e	M7X								X coordinate sprite 7
15	$d00f	M7Y								Y coordinate sprite 7
16	$d010	M7X8	M6X8	M5X8	M4X8	M3X8	M2X8	M1X8	M0X8	MSBs of X coordinates
17	$d011	RST8	ECM	BMM	DEN	RSEL	YSCROLL			Control register 1
18	$d012	RASTER								Raster counter
19	$d013	LPX								Light pen X
20	$d014	LPY								Light pen Y
21	$d015	M7E	M6E	M5E	M4E	M3E	M2E	M1E	M0E	Sprite enabled
22	$d016	-	-	RES	MCM	CSEL	XSCROLL			Control register 2
23	$d017	M7YE	M6YE	M5YE	M4YE	M3YE	M2YE	M1YE	M0YE	Sprite Y expansion
24	$d018	VM13	VM12	VM11	VM10	CB13	CB12	CB11	-	Memory pointers
25	$d019	IRQ	-	-	-	ILP	IMMC	IMBC	IRST	Interrupt register
26	$d01a	-	-	-	-	ELP	EMMC	EMBC	ERST	Interrupt enabled
27	$d01b	M7DP	M6DP	M5DP	M4DP	M3DP	M2DP	M1DP	M0DP	Sprite data priority
28	$d01c	M7MC	M6MC	M5MC	M4MC	M3MC	M2MC	M1MC	M0MC	Sprite multicolor
29	$d01d	M7XE	M6XE	M5XE	M4XE	M3XE	M2XE	M1XE	M0XE	Sprite X expansion
30	$d01e	M7M	M6M	M5M	M4M	M3M	M2M	M1M	M0M	Sprite-sprite collision
31	$d01f	M7D	M6D	M5D	M4D	M3D	M2D	M1D	M0D	Sprite-data collision
32	$d020	-	-	-	-	EC				Border color
33	$d021	-	-	-	-	B0C				Background color 0
34	$d022	-	-	-	-	B1C				Background color 1
35	$d023	-	-	-	-	B2C				Background color 2
36	$d024	-	-	-	-	B3C				Background color 3
37	$d025	-	-	-	-	MM0				Sprite multicolor 0
38	$d026	-	-	-	-	MM1				Sprite multicolor 1
39	$d027	-	-	-	-	M0C				Color sprite 0
40	$d028	-	-	-	-	M1C				Color sprite 1
41	$d029	-	-	-	-	M2C				Color sprite 2
42	$d02a	-	-	-	-	M3C				Color sprite 3

43	$d02b	-	-	-	-	M4C	Color sprite 4
44	$d02c	-	-	-	-	M5C	Color sprite 5
45	$d02d	-	-	-	-	M6C	Color sprite 6
46	$d02e	-	-	-	-	M7C	Color sprite 7

Notes:

- The bits marked with '-' are not connected and give "1" on reading
- The VIC registers are repeated each 64 bytes in the area $d000-$d3ff, i.e. register 0 appears on addresses $d000, $d040, $d080 etc.
- The unused addresses $d02f-$d03f give $ff on reading, a write access is ignored
- The registers $d01e and $d01f cannot be written and are automatically cleared on reading
- The RES bit (bit 5) of register $d016 has no function on the VIC 6567/6569 examined as yet. On the 6566, this bit is used to stop the VIC.
- Bit 7 in register $d011 (RST8) is bit 8 of register $d012. Together they are called "RASTER" in the following. A write access to these bits sets the comparison line for the raster interrupt (see section 3.12.).

3.3 COLOR PALETTE

The VIC has a hard-wired palette of 16 colors that are encoded with 4 bits:

0	black
1	white
2	red
3	cyan
4	pink
5	green
6	blue
7	yellow
8	orange
9	brown
10	light red
11	dark gray
12	medium gray
13	light green
14	light blue
15	light gray

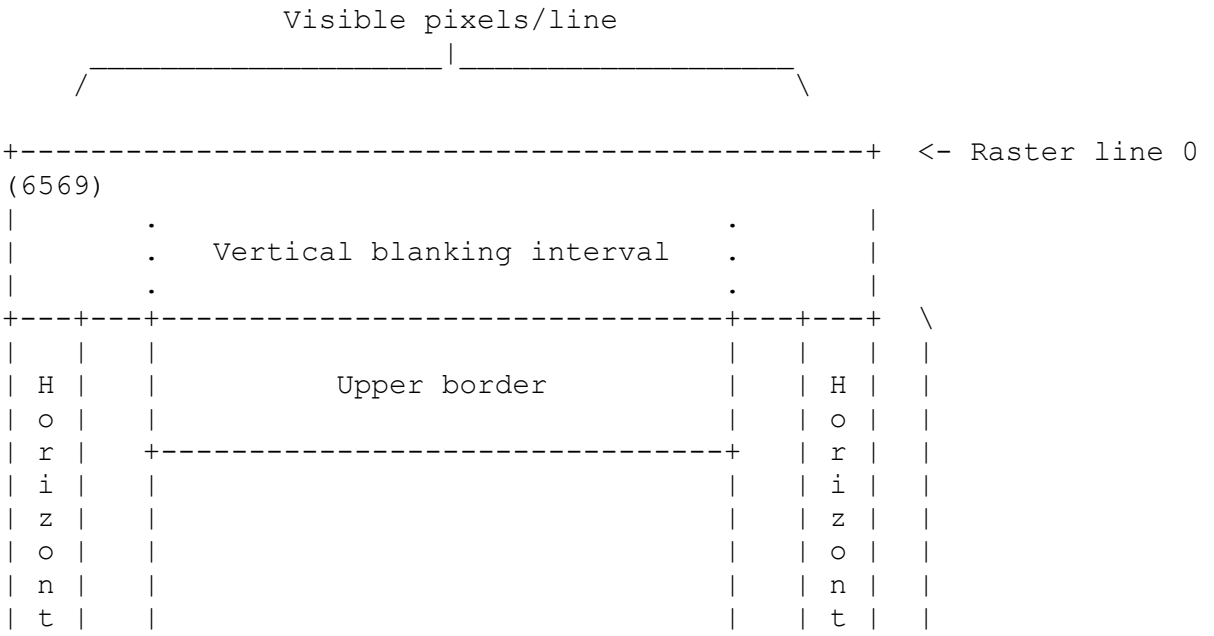
3.4 DISPLAY GENERATION AND DISPLAY WINDOW DIMENSIONS

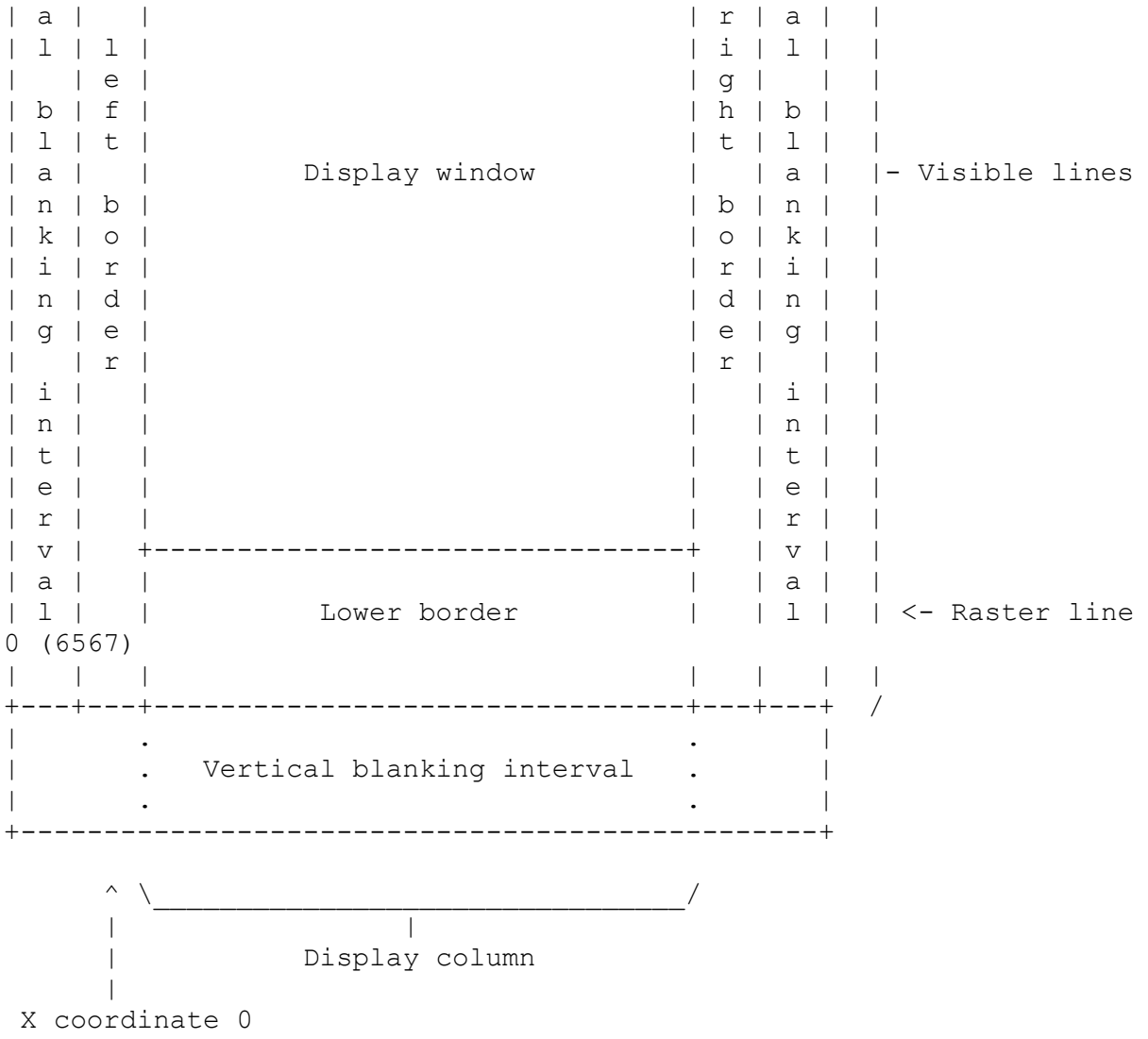
As usual for controlling CRTs, the VIC builds the video frame line by line. The line number and the number of clock cycles per line are constant for every VIC type. The VIC works character-based, every character consists of a matrix of 8×8 pixels, so a text line is made up of 8-pixel lines. 40×25 text characters are displayed in the text-based modes, 320×200 or 160×200 pixels in the bitmap modes.

In this article, the specification of a position on the screen is done with the raster line number as the Y coordinate (RASTER, register $d011/$d012) and a X coordinate that corresponds to the sprite coordinate system. When specifying the time of a VIC memory access or an internal operation in the VIC, the raster line number is used as Y coordinate and the number of the clock cycle within the line as X coordinate. As previously mentioned, 8 pixels make a clock cycle, so the specification of a sprite X coordinate is eight times more precise than that of a cycle number.

The graphics are displayed in an unmovable window in the middle of the visible screen area, the "display window". The area outside the display window is covered by the screen border and is displayed in the border color (EC, register $d020). You can also turn off the border partially or completely with some little tweaking; then you see that the display window is part of a "display column" that is made up by the linear extension of the display window to the top and bottom. With that you can divide the border in an upper/lower border and a left/right border. The visible screen area is surrounded by blanking intervals in which the video signal is turned off and in which the raster beam returns to the start of the next line or the start of the frame, respectively.

The following figure (not in scale) illustrates the last paragraph:





The height and width of the display window can each be set to two different values with the bits RSEL and CSEL in the registers $d011 and $d016:

RSEL	Display window height	First line	Last line
0	24 text lines/192 pixels	55 ($37)	246 ($f6)
1	25 text lines/200 pixels	51 ($33)	250 ($fa)

CSEL	Display window width	First X coo.	Last X coo.
0	38 characters/304 pixels	31 ($1f)	334 ($14e)
1	40 characters/320 pixels	24 ($18)	343 ($157)

If RSEL=0 the upper and lower border are each extended by 4 pixels into the display window, if CSEL=0 the left border is extended by 7 pixels and the right one by 9 pixels. The position of the display window and its resolution do not change, RSEL/CSEL only switch the starting and

ending position of the border display. The size of the video matrix also stays constantly at 40×25 characters.

With XSCROLL/YSCROLL (bits 0-2 of registers $d011 (XSCROLL) and $d016 (YSCROLL)), the position of the graphics inside the display window can be scrolled in single-pixel units up to 7 pixels to the right and to the bottom. This can be used for soft scrolling. The position of the display window itself doesn't change. To keep the graphics aligned with the window, X/YSCROLL have to be 0 and 3 for 25 lines/40 columns and both 7 for 24 lines/38 columns.

The dimensions of the video display for the different VIC types are as follows:

Type	Video system	# of lines	Visible lines	Cycles/ line	Visible pixels/line
6567R56A	NTSC-M	262	234	64	411
6567R8	NTSC-M	263	235	65	418
6569	PAL-B	312	284	63	403

Type	First vblank line	Last vblank line	First X coo. of a line	First visible X coo.	Last visible X coo.
6567R56A	13	40	412 ($19c)	488 ($1e8)	388 ($184)
6567R8	13	40	412 ($19c)	489 ($1e9)	396 ($18c)
6569	300	15	404 ($194)	480 ($1e0)	380 ($17c)

If you are wondering why the first visible X coordinates seem to come after the last visible ones: This is because for the reference point to mark the beginning of a raster line, the occurrence of the raster IRQ has been chosen, which doesn't coincide with X coordinate 0 but with the coordinate given as "First X coo. of a line". The X coordinates run up to $1ff (only

$1f7 on the 6569) within a line, then comes X coordinate 0. This is explained in more detail in the explanation of the structure of a raster line.

3.5 BAD LINES

As already mentioned, the VIC needs 40 additional bus cycles when fetching the character pointers (i.e. the character codes of one text line from the video matrix), because the 63-65 bus cycles available for transparent (unnoticed by the processor) access for the VIC during the first clock phases within a line are not sufficient to read both the character pointers and the pixel data for the characters from memory.

For this reason, the VIC uses the mechanism described in section 2.4.3. to "stun" the processor for 40-43 cycles during the first pixel line of each text line to read the character

pointers. The raster lines in which this happens are usually called "Bad Lines" ("bad" because they stop the processor and thus slow down the computer and lead to problems if the

precise timing of a program is essential, e.g. for the transmission of data to/from a floppy drive).

The character pointer access is also done in the bitmap modes, because the video matrix data is then used for color information.

Normally, every eighth line inside the display window, starting with the very first line of the graphics, is a Bad Line, i.e the first raster lines of each text line. So the position of the Bad Lines depends on the YSCROLL. As you will see later, the whole graphics display and memory access scheme depend completely on the position of the Bad Lines.

It is therefore necessary to introduce a more general definition, namely that of a "Bad Line Condition":

> A Bad Line Condition is given at any arbitrary clock cycle, if at the negative edge of ø0 at the beginning of the cycle RASTER >= $30 and RASTER <= $f7 and the lower three bits of RASTER are equal to YSCROLL and if the DEN bit was set during an arbitrary cycle of raster line $30.

This definition has to be taken literally. You can generate and take away a Bad Line condition multiple times within an arbitrary raster line in the range of $30-$f7 by modifying YSCROLL, and thus make every raster line within the display window completely or partially a Bad Line, or trigger or suppress all the other functions that are connected with a Bad Line Condition. If YSCROLL=0, a Bad Line Condition occurs in raster line $30 as soon as the DEN bit (register $d011, bit 4) is set (for more about the DEN bit, see section 3.10.).

The following three sections describe the function units that are used for displaying the graphics. Section 3.6. explains the memory interface that is used to read the graphics data and the timing of the accesses within a raster line. Section 3.7. is about the display unit that converts the text and bitmap graphics data into colors and generates the addresses for the memory access. Section 3.8. covers the sprites and their address generation.

3.6 MEMORY ACCESS

3.6.1 THE X COORDINATES

Before explaining the timing of memory accesses within a raster line, we will quickly explain how to obtain the X coordinates. This is necessary because the VIC doesn't have a counterpart to the RASTER register (which gives the current Y coordinate) to hold the X coordinates, so you cannot simply read them with the processor. But the VIC surely keeps

track of the X coordinates internally as the horizontal sprite positions are based on them, and a pulse at the light pen input LP latches the current X position in the register LPX ($d013).

Determining the absolute X coordinates of events within a raster line is not trivial as you cannot e.g. simply put a sprite to a well-defined X coordinate and conclude from the text characters displayed at the same X position to the X coordinates of the memory accesses belonging to these characters. The memory access and the display are separate function units and the read graphics data is not immediately displayed on the screen (there is a delay of 12 pixels).

So a different approach has been taken: The absolute position of a single X coordinate within the raster line was measured with the LPX register and the other X coordinates were determined relative to this. To do that, the IRQ output of the VIC has been connected to the LP input and the VIC has been programmed for a raster line interrupt. As the negative edge of IRQ was defined to be the start of a raster line, the absolute X position of the line start could be determined. The position of the negative edge of BA during a Bad Line was also measured with this method and the result was consistent with the relative distance of IRQ and BA to each other. Based on these two measurements, the X coordinates of all other events within a raster line have been determined (see [4]). Not until now the sprite X coordinates were used to be able to determine the moment of the display generation of the text characters.

This of course implicitly assumes that the LPX coordinates are the same as the sprite X coordinates. There is, however, no indication and thus no reason to suppose that they don't (a direct correlation would also be the simplest solution in terms of circuit design).

3.6.2 ACCESS TYPES

The VIC generates two kinds of graphics that require access to memory: The text/bitmap graphics (also often called "background graphics" or simply "graphics") and the sprite graphics. Both require accesses to two separated memory areas:

For the text/bitmap graphics:

- The video matrix; an area of 1000 video addresses (40×25, 12 bits each) that can be moved in 1KB steps within the 16KB address space of the VIC with the bits VM10-VM13 of register $d018. It stores the character codes and their color for the text modes and some of the color information of 8×8 pixel blocks for the bitmap modes. The Color RAM is part of the video matrix, it delivers the upper 4 bits of the 12-bit matrix. The data read from the video matrix is stored in an internal buffer in the VIC, the 40×12 bit video matrix/color line.

- The character generator resp. the bitmap; an area of 2048 bytes (bitmap: 8192 bytes) that can be moved in 2KB steps (bitmap: 8KB steps) within the VIC address space with the bits CB11-CB13 (bitmap: only CB13) of register $d018. It stores the pixel data of the characters for the text modes and the bitmap for the bitmap modes. The character generator has basically nothing to do with the Char ROM. The Char ROM only contains prepared bit patterns that can be used as character generator, but you can also store the character generator in normal RAM to define your own character images.

For the sprites:

- The sprite data pointers; 8 bytes after the end of the video matrix, that select one out of 256 blocks of 64 bytes within the VIC address space for each sprite.
- The sprite data; an area of 63 bytes containing the pixel data of the sprites which can be moved in steps of 64 bytes with the sprite data pointers independently for each sprite.

Corresponding to that, the VIC does 4 different kinds of graphics accesses:

1. To the video matrix ("c-access", 12 bits wide).
2. To the pixel data, i.e. character generator or bitmap ("g-access", 8 bits wide).
3. To the sprite data pointers ("p-access", 8 bits wide).
4. To the sprite data ("s-access", 8 bits wide).

Moreover, the VIC does two additional types of accesses:

5. Accesses for refreshing the dynamic RAM, 5 read accesses per raster line.
6. Idle accesses. As described, the VIC accesses in every first clock phase although there are some cycles in which no other of the above mentioned accesses is pending. In this case, the VIC does an idle access; a read access to video address $3fff (i.e. to $3fff, $7fff, $bfff or $ffff depending on the VIC bank) of which the result is discarded.

3.6.3 TIMING OF A RASTER LINE

The sequence of VIC memory accesses within a raster line is hard-wired, independent of the graphics mode and the same for every raster line. The negative edge of IRQ on a raster interrupt has been used to define the beginning of a line (this is also the moment in which the RASTER register is incremented). Raster line 0 is, however, an exception: In this line, IRQ and incrementing (resp. resetting) of RASTER are performed one cycle later than in the other lines. But for simplicity we assume equal line lengths and define the beginning of raster line 0 to be one cycle before the occurrence of the IRQ.

First the timing diagrams, the explanation follows:

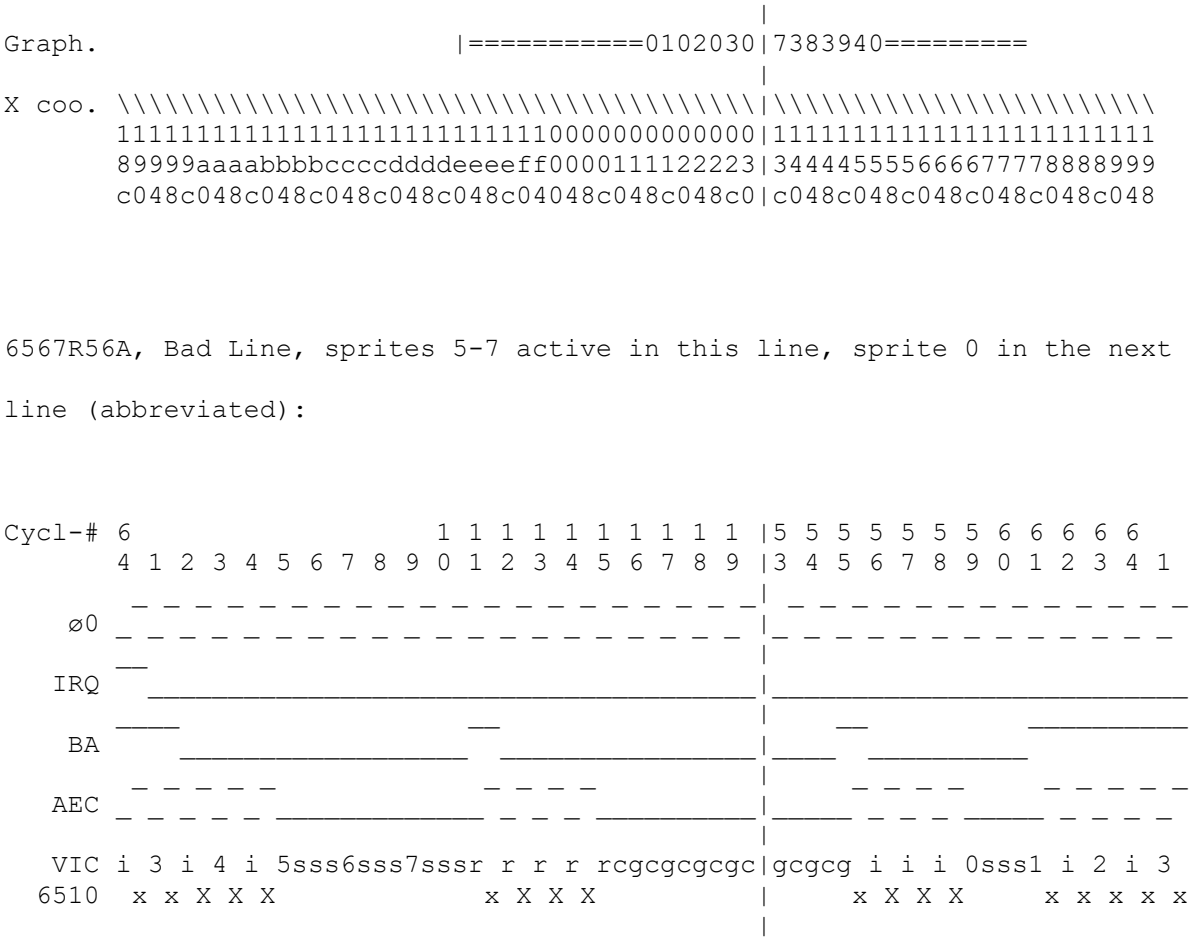
6569, Bad Line, no sprites:

```
Cycl-# 6                   1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3
3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 6 6 6 6
```

```
       3 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 1
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    ø0 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
       __
   IRQ
___________________________________________________________________________
_____________________________________________________
       ________________________
____________________
    BA
___________________________________________________________________________
___________
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _
   AEC _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
___________________________________________________________________________
______ _ _ _ _ _ _ _ _ _

   VIC i 3 i 4 i 5 i 6 i 7 i r r r r
rcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcgcg
cgcgcg i i 0 i 1 i 2 i 3
  6510  x x x x x x x x x x x x X X X
x x x x x x x x x x

Graph.
|===========010203040506070809101112131415161718192021222324252627282930313
2333435363738394O=========

X coo.
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

111111111111111111111111110000000000000000000000000000000000000000000000000
00000000000000011111111111111111111111111111111111111

89999aaaabbbbccccddddeeeeff0000111122223333444455556666777788889999aaaabbbb
ccccddddeeeeffff000011112222333344445555666677778888999

c048c048c048c048c048c048c04048c048c048c048c048c048c048c048c048c048c048c048c
048c048c048c048c048c048c048c048c048c048c048c048c048
```

6569, no Bad Line, no sprites (abbreviated):

```
Cycl-# 6                   1 1 1 1 1 1 1 1 1 1 |5 5 5 5 5 5 5 6 6 6 6
       3 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 |3 4 5 6 7 8 9 0 1 2 3 1
                                               |
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _| _ _ _ _ _ _ _ _ _ _ _ _
    ø0 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ |_ _ _ _ _ _ _ _ _ _ _ _
       __                                      |
   IRQ   ______________________________________|________________________
       ________________________________________|________________________
    BA                                         |
                                               |
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _| _ _ _ _ _ _ _ _ _ _ _ _
   AEC _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ |_ _ _ _ _ _ _ _ _ _ _ _
                                               |
   VIC i 3 i 4 i 5 i 6 i 7 i r r r r r g g g g |g g g i i 0 i 1 i 2 i 3
  6510  x x x x x x x x x x x x x x x x x x x x| x x x x x x x x x x x x
```


Graph. |==========0102030|7383940=========
X coo. \\|\\\\\\\\\\\\\\\\\\\\\\\\
1111111111111111111111111110000000000000|111111111111111111111111
89999aaaabbbbccccddddeeeeff0000111122223|344445555666677778888999
c048c048c048c048c048c048c04048c048c048c0|c048c048c048c048c048c048
6567R56A, Bad Line, sprites 5-7 active in this line, sprite 0 in the next
line (abbreviated):
Cycl-# 6 1 1 1 1 1 1 1 1 1 1 |5 5 5 5 5 5 5 6 6 6 6 6
4 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 |3 4 5 6 7 8 9 0 1 2 3 4 1
ø0
IRQ
BA
AEC
VIC i 3 i 4 i 5sss6sss7sssr r r r rcgcgcgcgc|gcgcg i i i 0sss1 i 2 i 3
6510 x x X X X x X X X | x X X X x x x x x
Graph. |==========0102030|7383940==========
X coo. \\|\\\\\\\\\\\\\\\\\\\\\\\\
1111111111111111111111111110000000000000|111111111111111111111111
999aaaabbbbccccddddeeeeffff0000111122223|344445555666677778888999a
48c048c048c048c048c048c048c048c048c048c0|c048c048c048c048c048c048c0
6567R8, no Bad Line, sprites 2-7 active in this line, sprites 0-4 in the
next line (abbreviated):
Cycl-# 6 1 1 1 1 1 1 1 1 1 1 |5 5 5 5 5 5 5 6 6 6 6 6 6
5 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 |3 4 5 6 7 8 9 0 1 2 3 4 5 1
ø0
IRQ
BA
AEC

```
   VIC ss3sss4sss5sss6sss7sssr r r r r g g g g |g g g i i i i
0sss1sss2sss3s
  6510                         x x x x x x x x| x x x x X X X
                                              |
Graph.                        |===========0102030|7383940===========
                                              |
X coo.
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\|\\\\\\\\\\\\\\\\\\\\\\\\\\\\\

1111111111111111111111111110000000000000|1111111111111111111111111111

999aaaabbbbccccddddeeeefffff0000111122223|3444455556666777788888899999a

48c048c048c048c048c048c048c048c048c048c0|c048c048c048c048c04cccc04c80
```

The line "Cycl-#" show the number of the clock cycle within the raster line. The line starts with cycle 1 and consists of 63 cycles on the 6569, of 64 cycles on the 6567R56A and of 65 cycles on the 6567R8. The last cycle of the previous line and the first cycle of the next line have also been included in the diagrams to make things clearer.

The lines "ø0", "IRQ", "BA" and "AEC" reflect the state of the bus signals with the same names. ø0 is low in the first phase and high in the second phase.

The symbols in the lines "VIC" and "6510" show what kind of access VIC and 6510 do in the corresponding clock phase (for an explanation of the different access types of the VIC see section 3.6.2.):

- c Access to video matrix and Color RAM (c-access)
- g Access to character generator or bitmap (g-access)
- 0-7 Reading the sprite data pointer for sprite 0-7 (p-access)
- s Reading the sprite data (s-access)
- r DRAM refresh
- i Idle access
- x Read or write access of the processor
- X Processor may do write accesses, stops on first read (BA is low and so is RDY)

The line "X coo." contains the X coordinates of the beginning of each clock phase (thus the "\\\" as a reminder) and the line "Graph." is a projection of the 40-column display window and the border to these coordinates, for positioning sprites. However, this doesn't correspond to the signal on the VIC video output. Also, you cannot see from the "Graph." line when the border unit generates the border. This happens approx. 8 pixels later than shown in the "Graph." line.

To time the accesses of the processor within a raster line when programming, it's best to use the VIC g-accesses for orientation by changing a byte in graphics memory with the 6510 and watching on the screen on which character the change is first visible. The write access of the processor must then have occurred in the clock phase immediately before. Then you can use

the diagrams to determine the clock cycle in which the access took place and count the other accesses relative to it.

3.7 TEXT/BITMAP DISPLAY

3.7.1 IDLE STATE/DISPLAY STATE

The text/bitmap display logic in the VIC is in one of two states at any time: The idle state and the display state.

- In display state, c- and g-accesses take place, the addresses and interpretation of the data depend on the selected display mode.
- In idle state, only g-accesses occur. The access is always to address $3fff ($39ff when the ECM bit in register $d016 is set). The graphics are displayed by the sequencer exactly as in display state, but with the video matrix data treated as "0" bits.

The transition from idle to display state occurs as soon as there is a Bad Line Condition (see section 3.5.). The transition from display to idle state occurs in cycle 58 of a line if the RC (see next section) contains the value 7 and there is no Bad Line Condition.

As long as register $d011 is not modified in the middle of a frame, the display logic is in display state within the display window and in idle state outside of it. If you set a YSCROLL other than 3 in a 25-line display window and store a value not equal to zero in $3fff you can see the stripes generated by the sequencer in idle state on the upper or lower side of the window.

In [4], idle accesses as well as g-accesses in idle state are called "idle bus cycle". But the two phenomena are not the same. The accesses marked with "+" in the diagrams of [4] are normal g-accesses. In this article, the term "idle access" is only used for the accesses marked with "i" in the diagrams in section 3.6.3., and not for the g-accesses during idle state.

3.7.2 VC AND RC

Probably the most important result of the VIC examinations is the discovery of the function of the internal registers "VC" and "RC" of the VIC. They are used to generate the addresses for accessing the video matrix and the character generator/bitmap.

Strictly speaking there are three registers:

- "VC" (video counter) is a 10-bit counter that can be loaded with the value from VCBASE.
- "VCBASE" (video counter base) is a 10-bit data register with reset input that can be loaded with the value from VC.
- "RC" (row counter) is a 3-bit counter with reset input.

Besides this, there is a 6-bit counter with reset input that keeps track of the position within the internal 40×12 bit video matrix/color line where read character pointers are stored resp. read again. I will call this "VMLI" (video matrix line index) here.

These four registers behave according to the following rules:

1. Once somewhere outside of the range of raster lines $30-$f7 (i.e. outside of the Bad Line range), VCBASE is reset to zero. This is presumably done in raster line 0, the exact moment cannot be determined and is irrelevant.
2. In the first phase of cycle 14 of each line, VC is loaded from VCBASE (VCBASE->VC) and VMLI is cleared. If there is a Bad Line Condition in this phase, RC is also reset to zero.
3. If there is a Bad Line Condition in cycles 12-54, BA is set low and the c-accesses are started. Once started, one c-access is done in the second phase of every clock cycle in the range 15-54. The read data is stored in the video matrix/color line at the position specified by VMLI. This data is internally read from the position specified by VMLI as well on each g-access in display state.
4. VC and VMLI are incremented after each g-access in display state.
5. In the first phase of cycle 58, the VIC checks if RC=7. If so, the video logic goes to idle state and VCBASE is loaded from VC (VC->VCBASE). If the video logic is in display state afterwards (this is always the case if there is a Bad Line Condition), RC is incremented.

These rules normally see that VC counts all 1000 addresses of the video matrix within the display frame and that RC counts the 8 pixel lines of each text line. The behavior of VC and RC is largely determined by Bad Line Conditions which you can control with the processor via YSCROLL, giving you control of the VC and RC within certain limits.

3.7.3 GRAPHICS MODES

The graphics data sequencer is capable of 8 different graphics modes that are selected by the bits ECM, BMM and MCM (Extended Color Mode, Bit Map Mode and Multi Color Mode) in the registers $d011 and $d016 (of the 8 possible bit combinations, 3 are "invalid" and generate the same output, the color black). The idle state is a bit special in that no c-accesses occur in it and the sequencer uses "0" bits for the video matrix data.

The sequencer outputs the graphics data in every raster line in the area of the display column as long as the vertical border flip-flop is reset (see section 3.9.). Outside of the display column and if the flip-flop is set, the last current background color is displayed (this area is normally covered by the border). The heart of the sequencer is an 8-bit shift register that is shifted by 1 bit every pixel and reloaded with new graphics data after every g-access. With XSCROLL from register $d016 the reloading can be delayed by 0-7 pixels, thus shifting the display up to 7 pixels to the right.

The address generator for the text/bitmap accesses (c- and g-accesses) has basically 3 modes for the g-accesses (the c-accesses always follow the same address scheme). In display state, the BMM bit selects either character generator accesses (BMM=0) or bitmap accesses (BMM=1). In idle state, the g-accesses are always done at video address $3fff. If the ECM bit is set, the address generator always holds the address lines 9 and 10 low without any other changes to the addressing scheme (e.g. the g-accesses in idle state then occur at address $39ff).

The 8 graphics modes are now covered separately and the generated addresses and the interpretation of the read data on c- and g-accesses is described.

This is followed by a description of the peculiarities of the idle state. For easy reference, the addresses are always given explicitly for every mode although e.g. the c-accesses are always the same.

3.7.3.1 STANDARD TEXT MODE (ECM/BMM/MCM=0/0/0)

In this mode (as in all text modes), the VIC reads 8-bit character pointers from the video matrix that specify the address of the dot matrix of the character in the character generator. A character set of 256 characters is available, each consisting of 8×8 pixels which are stored in 8 successive bytes in the character generator. Video matrix and character generator can be moved in memory with the bits VM10-VM13 and CB11-CB13 of register $d018.

In standard text mode, every bit in the character generator directly corresponds to one pixel on the screen. The foreground color is given by the color nibble from the video matrix for each character, the background color is set globally with register $d021.

c-access

Addresses

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Data

11	10	9	8	7	6	5	4	3	2	1	0
Color of "1" pixels				D7	D6	D5	D4	D3	D2	D1	D0

g-access

Addresses

```
| 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|CB13|CB12|CB11| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | RC2| RC1| RC0|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 |         8 pixels (1 bit/pixel)        |
 |                                       |
 | "0": Background color 0 ($d021)       |
 | "1": Color from bits 8-11 of c-data   |
 +---------------------------------------+
```

3.7.3.2 MULTICOLOR TEXT MODE (ECM/BMM/MCM=0/0/1)

This mode allows for displaying four-colored characters at the cost of horizontal resolution. If bit 11 of the c-data is zero, the character is displayed as in standard text mode with only the colors 0-7 available for the foreground. If bit 11 is set, each two adjacent bits of the dot matrix form one pixel. By this means, the resolution of a character of reduced to 4×8 (the pixels are twice as wide, so the total width of the characters doesn't change).

It is interesting that not only the bit combination "00" but also "01" is regarded as "background" for the sprite priority and collision detection.

```
c-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |VM13|VM12|VM11|VM10| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+----+----+----+----+
 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+
 | MC |   Color of   | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
 |flag|  "11" pixels |    |    |    |    |    |    |    |    |
 +----+--------------+----+----+----+----+----+----+----+----+

g-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |CB13|CB12|CB11| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | RC2| RC1| RC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

```
Data

+----+----+----+----+----+----+----+----+
|  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+
|         8 pixels (1 bit/pixel)         |
|                                        | MC flag = 0
| "0": Background color 0 ($d021)        |
| "1": Color from bits 8-10 of c-data    |
+----------------------------------------+
|         4 pixels (2 bits/pixel)        |
|                                        |
| "00": Background color 0 ($d021)       | MC flag = 1
| "01": Background color 1 ($d022)       |
| "10": Background color 2 ($d023)       |
| "11": Color from bits 8-10 of c-data   |
+----------------------------------------+
```

3.7.3.3 STANDARD BITMAP MODE (ECM/BMM/MCM=0/1/0)

In this mode (as in all bitmap modes), the VIC reads the graphics data from a 320×200 bitmap in which every bit corresponds to one pixel on the screen. The data from the video matrix is used for color information. As the video matrix is still only a 40×25 matrix, you can only specify the colors for blocks of 8×8 pixels individually (sort of a YC 8:1 format). As the designers of the VIC wanted to realize the bitmap mode with as little additional circuitry as possible (the VIC-I didn't have a bitmap mode), the arrangement of the bitmap in memory is somewhat weird: In contrast to modern video chips that read the bitmap in a linear fashion from memory, the VIC forms an 8×8 pixel block on the screen from 8 successive bytes of the bitmap. The video matrix and the bitmap can be moved in memory with the bits VM10-VM13 and CB13 of register $d018.

In standard bitmap mode, every bit in the bitmap directly corresponds to one pixel on the screen. Foreground and background color can be arbitrarily set for every 8×8 block.

```
c-access

Addresses

+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|VM13|VM12|VM11|VM10| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+

Data

+----+----+----+----+----+----+----+----+----+----+----+----+
| 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+----+----+----+----+
|       unused      |     Color of      |     Color of      |
|                   |    "1" pixels     |    "0" pixels     |
+-------------------+-------------------+-------------------+
```

```
g-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |CB13| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0| RC2| RC1| RC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 |         8 pixels (1 bit/pixel)        |
 |                                       |
 | "0": Color from bits 0-3 of c-data    |
 | "1": Color from bits 4-7 of c-data    |
 +---------------------------------------+
```

3.7.3.4 MULTICOLOR BITMAP MODE (ECM/BMM/MCM=0/1/1)

Similar to the multicolor text mode, this mode also forms (twice as wide) pixels by combining two adjacent bits. So the resolution is reduced to 160×200 pixels.

The bit combination "01" is also treated as "background" for the sprite priority and collision detection, as in multicolor text mode.

```
c-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |VM13|VM12|VM11|VM10| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+----+----+----+----+
 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+
 |     Color of      |     Color of      |     Color of      |
 |    "11 pixels"    |    "01" pixels    |    "10" pixels    |
 +-------------------+-------------------+-------------------+

g-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |CB13| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0| RC2| RC1| RC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

Data

```
+----+----+----+----+----+----+----+----+
|  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+
|         4 pixels (2 bits/pixel)       |
|                                       |
| "00": Background color 0 ($d021)      |
| "01": Color from bits 4-7 of c-data   |
| "10": Color from bits 0-3 of c-data   |
| "11": Color from bits 8-11 of c-data  |
+---------------------------------------+
```

3.7.3.5 ECM TEXT MODE (ECM/BMM/MCM=1/0/0)

This text mode is the same as the standard text mode, but it allows the selection of one of four background colors for every single character. The selection is done with the upper two bits of the character pointer. This, however, reduces the character set from 256 to 64 characters.

c-access

Addresses

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|VM13|VM12|VM11|VM10| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

Data

```
+----+----+----+----+----+----+----+----+----+----+----+----+
| 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+----+----+----+----+
|     Color of      |Back.col.| D5 | D4 | D3 | D2 | D1 | D0 |
|    "1" pixels     |selection|    |    |    |    |    |    |
+-------------------+---------+----+----+----+----+----+----+
```

g-access

Addresses

```
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|CB13|CB12|CB11|  0 |  0 | D5 | D4 | D3 | D2 | D1 | D0 | RC2| RC1| RC0|
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

Data

```
+----+----+----+----+----+----+----+----+
|  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
+----+----+----+----+----+----+----+----+
|         8 pixels (1 bit/pixel)        |
|                                       |
```

```
| "0": Depending on bits 6/7 of c-data  |
|      00: Background color 0 ($d021)   |
|      01: Background color 1 ($d022)   |
|      10: Background color 2 ($d023)   |
|      11: Background color 3 ($d024)   |
| "1": Color from bits 8-11 of c-data   |
+---------------------------------------+
```

3.7.3.6 INVALID TEXT MODE (ECM/BMM/MCM=1/0/1)

Setting the ECM and MCM bits simultaneously doesn't select one of the "official" graphics modes of the VIC but creates only black pixels. Nevertheless, the graphics data sequencer internally generates valid graphics data that can trigger sprite collisions even in this mode. By using sprite collisions, you can also read out the generated data (but you cannot see anything, the screen is black). You can, however, only distinguish foreground and background pixels as you cannot get color information from sprite collisions.

The generated graphics is similar to that of the multicolor text mode, but the character set is limited to 64 characters as in ECM mode.

```
c-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |VM13|VM12|VM11|VM10| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+----+----+----+----+
 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+
 | MC |          unused         | D5 | D4 | D3 | D2 | D1 | D0 |
 |flag|                         |    |    |    |    |    |    |
 +----+-------------------------+----+----+----+----+----+----+

g-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |CB13|CB12|CB11|  0 |  0 | D5 | D4 | D3 | D2 | D1 | D0 | RC2| RC1| RC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 |         8 pixels (1 bit/pixel)        |
```

```
|                                       | MC flag = 0
| "0": Black (background)               |
| "1": Black (foreground)               |
+---------------------------------------+
|         4 pixels (2 bits/pixel)       |
|                                       |
| "00": Black (background)              | MC flag = 1
| "01": Black (background)              |
| "10": Black (foreground)              |
| "11": Black (foreground)              |
+---------------------------------------+
```

3.7.3.7 INVALID BITMAP MODE 1 (ECM/BMM/MCM=1/1/0)

This mode also only displays a black screen, but the pixels can also be read out with the sprite collision trick.

The structure of the graphics is basically as in standard bitmap mode, but the bits 9 and 10 of the g-addresses are always zero due to the set ECM bit and so the graphics is - roughly said - made up of four "sections" that are each repeated four times.

c-access

Addresses

13	12	11	10	9	8	7	6	5	4	3	2	1	0
VM13	VM12	VM11	VM10	VC9	VC8	VC7	VC6	VC5	VC4	VC3	VC2	VC1	VC0

Data

11	10	9	8	7	6	5	4	3	2	1	0
unused											

g-access

Addresses

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CB13	VC9	VC8	0	0	VC5	VC4	VC3	VC2	VC1	VC0	RC2	RC1	RC0

Data

7	6	5	4	3	2	1	0
8 pixels (1 bit/pixel)							
"0": Black (background)							

```
| "1": Black (foreground)                |
+----------------------------------------+
```

3.7.3.8 INVALID BITMAP MODE 2 (ECM/BMM/MCM=1/1/1)

The last invalid mode also creates a black screen but it can also be "scanned" with sprite-graphics collisions.

The structure of the graphics is basically as in multicolor bitmap mode, but the bits 9 and 10 of the g-addresses are always zero due to the set ECM bit, with the same results as in the first invalid bitmap mode. As usual, the bit combination "01" is part of the background.

```
c-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |VM13|VM12|VM11|VM10| VC9| VC8| VC7| VC6| VC5| VC4| VC3| VC2| VC1| VC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+----+----+----+----+
 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+
 |                          unused                           |
 +-----------------------------------------------------------+

g-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |CB13| VC9| VC8|  0 |  0 | VC5| VC4| VC3| VC2| VC1| VC0| RC2| RC1| RC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 |         4 pixels (2 bits/pixel)       |
 |                                       |
 | "00": Black (background)              |
 | "01": Black (background)              |
 | "10": Black (foreground)              |
 | "11": Black (foreground)              |
 +---------------------------------------+
```

3.7.3.9 IDLE STATE

In idle state, the VIC reads the graphics data from address $3fff (resp. $39ff if the ECM bit is set) and displays it in the selected graphics mode, but with the video matrix data (normally read in the c-accesses) being all "0" bits. So the byte at address $3fff/$39ff is output repeatedly.

```
c-access

 No c-accesses occur.

 Data

 +----+----+----+----+----+----+----+----+----+----+----+----+
 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+
 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+

g-access

 Addresses (ECM=0)

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Addresses (ECM=1)

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |  1 |  1 |  1 |  0 |  0 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |  1 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 |         8 pixels (1 bit/pixel)        | Standard text mode/
 |                                       | Multicolor text mode/
 | "0": Background color 0 ($d021)       | ECM text mode
 | "1": Black                            |
 +---------------------------------------+
 |         8 pixels (1 bit/pixel)        | Standard bitmap mode/
 |                                       | Invalid text mode/
 | "0": Black (background)               | Invalid bitmap mode 1
 | "1": Black (foreground)               |
 +---------------------------------------+
 |         4 pixels (2 bits/pixel)       | Multicolor bitmap mode
 |                                       |
 | "00": Background color 0 ($d021)      |
 | "01": Black (background)              |
 | "10": Black (foreground)              |
 | "11": Black (foreground)              |
 +---------------------------------------+
 |         4 pixels (2 bits/pixel)       | Invalid bitmap mode 2
```

```
|                                       |
| "00": Black (background)              |
| "01": Black (background)              |
| "10": Black (foreground)              |
| "11": Black (foreground)              |
+---------------------------------------+
```

3.8 SPRITES

Apart from the text/bitmap graphics, the VIC can display eight independent 24×21 pixels large, freely movable objects, the "sprites" (called "MOBs" (Movable Object Blocks) in [2]).

The sprites can have an arbitrary position on the screen, you can switch them on and off one at a time with the bits of register $d015 (MxE), expand them by the factor 2 in X and/or Y direction with registers $d017/$d01d (with the resolution still being 24×21 pixels), choose between standard and multicolor mode with register $d01c (MxMC), set the display priority with respect to the text/bitmap graphics with register $d01b (MxDP) and assign a different color to each sprite (registers $d027-$d02e). Besides, the VIC has the ability to detect collisions between sprites among themselves or between sprites and text/bitmap graphics and to trigger an interrupt on such collisions (see 3.11.).

The position of the top left corner of a sprite is specified with the coordinate registers (MxX, MxY) belonging to it. There are 8 bits for the Y coordinate and 9 bits for the X coordinate (the most significant bits of all sprites are collected in register $d010).

3.8.1 MEMORY ACCESS AND DISPLAY

The 63 bytes of sprite data necessary for displaying 24×21 pixels are stored in memory in a linear fashion: 3 adjacent bytes form one line of the sprite.

These 63 bytes can be moved in steps of 64 bytes within the 16KB address space of the VIC. For this, the VIC reads a sprite data pointer for each sprite in every raster line from the very last 8 bytes of the video matrix (p-access) that is used as the upper 8 bits of the address for sprite data accesses (s-accesses). The lower 6 bits come from a sprite data counter (MC0-MC7, one for each sprite) that plays a similar role for the sprites as VC does for the video matrix. As the p-accesses are done in every raster line and not only when the belonging sprite is just displayed, you can change the appearance of a sprite in the middle of its display by changing the sprite data pointer.

When s-accesses are necessary for a sprite, they are done in the three half-cycles directly after the p-access belonging to the sprite within the raster line. The VIC also uses the BA and AEC signals (as in the Bad Lines) to access the bus in the second clock phase. BA will also go

low three cycles before the proper access in this case. The s-accesses are done in every raster line in which the sprite is visible (for the sprites 0-2, it is always in the line before, see the timing diagrams in section 3.6.3.), for every sprite in statically assigned cycles within the line.

Like the text and bitmap graphics, the sprites also have a standard mode and a multicolor mode. In standard mode, every bit directly corresponds to one pixel on the screen. A "0" pixel is transparent and the underlying graphics are visible below it, a "1" pixel is displayed in the sprite color from registers $d027-$d02e belonging to the sprite in question. In multicolor mode, two adjacent bits form one pixel, thus reducing the resolution of the sprite to 12×21 (the pixels are twice as wide).

Moreover, the sprites can be doubled in their size on the screen in X and/or Y direction (X/Y expansion). For that, every sprite pixel simply becomes twice as wide/tall, the resolution doesn't change. So a pixel of an x-expanded multicolor sprite is four times as wide as a pixel of an unexpanded standard sprite. Although both expansions look similar, they are implemented completely differently in the VIC. The X expansion simply instructs the sprite data sequencer to output pixels with half frequency. But the Y expansion makes the sprite address generator read from the same addresses in each two lines in sequence so that every sprite line is output twice.

Every sprite has its own sprite data sequencer whose core is a 24-bit shift register. Apart from that, there are two internal registers for every sprite:

- "MC" (MOB Data Counter) is a 6-bit counter that can be loaded from MCBASE.
- "MCBASE" (MOB Data Counter Base) is a 6-bit counter with reset input.

Besides, there is one expansion flip-flop per sprite that controls the Y expansion.

The display of a sprite is done after the following rules (the cycle numbers are only valid for the 6569):

1. The expansion flip-flop is set as long as the bit in MxYE in register $d017 corresponding to the sprite is cleared.
2. If the MxYE bit is set in the first phase of cycle 55, the expansion flip-flop is inverted.
3. In the first phases of cycle 55 and 56, the VIC checks for every sprite if the corresponding MxE bit in register $d015 is set and the Y coordinate of the sprite (odd registers $d001-$d00f) match the lower 8 bits of RASTER. If this is the case and the DMA for the sprite is still off, the DMA is switched on, MCBASE is cleared, and if the MxYE bit is set the expansion flip-flop is reset.
4. In the first phase of cycle 58, the MC of every sprite is loaded from its belonging MCBASE (MCBASE->MC) and it is checked if the DMA for the sprite is turned on and the Y coordinate of the sprite matches the lower 8 bits of RASTER. If this is the case, the display of the sprite is turned on.
5. If the DMA for a sprite is turned on, three s-accesses are done in sequence in the corresponding cycles assigned to the sprite (see the diagrams in section 3.6.3.). The p-accesses are always done, even if the sprite is turned off. The read data of the first

access is stored in the upper 8 bits of the shift register, that of the second one in the middle 8 bits and that of the third one in the lower 8 bits. MC is incremented by one after each s-access.

6. If the sprite display for a sprite is turned on, the shift register is shifted left by one bit with every pixel as soon as the current X coordinate of the raster beam matches the X coordinate of the sprite (even registers $d000-$d00e), and the bits that "fall off" are displayed. If the MxXE bit belonging to the sprite in register $d01d is set, the shift is done only every second pixel and the sprite appears twice as wide. If the sprite is in multicolor mode, every two adjacent bits form one pixel.
7. In the first phase of cycle 15, it is checked if the expansion flip-flop is set. If so, MCBASE is incremented by 2.
8. In the first phase of cycle 16, it is checked if the expansion flip-flop is set. If so, MCBASE is incremented by 1. After that, the VIC checks if MCBASE is equal to 63 and turns of the DMA and the display of the sprite if it is.

As the test in rule 3 is done at the end of a raster line, the sprite Y coordinates stored in the registers must be 1 less than the desired Y position of the first sprite line, as the sprite display will not start until the following line, after the first sprite data has been read (as long as the sprite is not positioned to the right of sprite X coordinate $164 (cycle 58, see rule 4)).

Sprites can be "reused" vertically: If you change the Y coordinate of a sprite to a later raster line during or after its display has completed, so that the comparisons mentioned in rules 1 and 2 will match again, the sprite is displayed again at that Y coordinate (you may then of course freely set a new X coordinate and sprite data pointer). It is therefore possible to display more than 8 sprites on the screen.

This is not possible in the horizontal direction. After 24 displayed pixels, the shift register has run empty and even if you change the X coordinate within a line so that the comparison in rule 4 will match again, no sprite data is displayed any more. So you can only display up to 8 sprites within one raster line at a time.

Once again, an overview of the scheme of p- and s-accesses:

```
p-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 |VM13|VM12|VM11|VM10|  1 |  1 |  1 |  1 |  1 |  1 |  1 |Sprite number |
 +----+----+----+----+----+----+----+----+----+----+----+--------------+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 | MP7| MP6| MP5| MP4| MP3| MP2| MP1| MP0|
 +----+----+----+----+----+----+----+----+
```

```
s-access

 Addresses

 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+
 | MP7| MP6| MP5| MP4| MP3| MP2| MP1| MP0| MC5| MC4| MC3| MC2| MC1| MC0|
 +----+----+----+----+----+----+----+----+----+----+----+----+----+----+

 Data

 +----+----+----+----+----+----+----+----+
 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
 +----+----+----+----+----+----+----+----+
 |         8 pixels (1 bit/pixel)        |
 |                                       | MxMC = 0
 | "0": Transparent                      |
 | "1": Sprite color ($d027-$d02e)       |
 +---------------------------------------+
 |         4 pixels (2 bits/pixel)       |
 |                                       |
 | "00": Transparent                     | MxMC = 1
 | "01": Sprite multicolor 0 ($d025)     |
 | "10": Sprite color ($d027-$d02e)      |
 | "11": Sprite multicolor 1 ($d026)     |
 +---------------------------------------+
```

3.8.2 PRIORITY AND COLLISION DETECTION

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

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

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

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

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

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

```
MxDP=0:

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

MxDP=1:

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

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

If you choose one of the invalid video modes only the sprites will be visible (fore- and background graphics will all become black, see sections 3.7.3.6.-3.7.3.8.), but by setting the sprites to appear behind the foreground graphics, the foreground graphics will actually become visible as black pixels overlaying the sprite pixels.

Together with the priority management, the VIC has the ability to detect collisions of sprites among themselves and of sprites and foreground pixels of the text/bitmap graphics.

A collision of sprites among themselves is detected as soon as two or more sprite data sequencers output a non-transparent pixel in the course of display generation (this can also happen somewhere outside of the visible screen area). In this case, the MxM bits of all

affected sprites are set in register $d01e and (if allowed, see section 3.12.), an interrupt is generated. The bits remain set until the register is read by the processor and are cleared automatically by the read access.

A collision of sprites and other graphics data is detected as soon as one or more sprite data sequencers output a non-transparent pixel and the graphics data sequencer outputs a foreground pixel in the course of display generation. In this case, the MxD bits of the affected sprites are set in register $d01f and (if allowed, see section 3.12.), an interrupt is generated. As with the sprite-sprite collision, the bits remain set until the register is read by the processor.

If the vertical border flip-flop is set (normally within the upper/lower border, see next section), the output of the graphics data sequencer is turned off and there are no collisions.

3.9 THE BORDER UNIT

The VIC uses two flip-flops to generate the border around the display window: A main border flip-flop and a vertical border flip-flop.

The main border flip-flop controls the border display. If it is set, the VIC displays the color stored in register $d020, otherwise it displays the color that the priority multiplexer switches through from the graphics or sprite data sequencer. So the border overlays the text/bitmap graphics as well as the sprites. It has the highest display priority.

The vertical border flip-flop is for auxiliary control of the upper/lower border. If it is set, the main border flip-flop cannot be reset. Apart from that, the vertical border flip-flop controls the output of the graphics data sequencer. The sequencer only outputs data if the flip-flop is

not set, otherwise it displays the background color. This was probably done to prevent sprite-graphics collisions in the border area.

There are 2×2 comparators belonging to each of the two flip-flops. There comparators compare the X/Y position of the raster beam with one of two hardwired values (depending on the state of the CSEL/RSEL bits) to control the flip-flops. The comparisons only match if the values are reached precisely. There is no comparison with an interval.

The horizontal comparison values:

	CSEL=0	CSEL=1
Left	31 ($1f)	24 ($18)
Right	335 ($14f)	344 ($158)

And the vertical ones:

```
        |   RSEL=0   |  RSEL=1
 -------+------------+----------
 Top    |  55 ($37)  |  51 ($33)
 Bottom | 247 ($f7)  | 251 ($fb)
```

The flip-flops are switched according to the following rules:

1. If the X coordinate reaches the right comparison value, the main border flip-flop is set.
2. If the Y coordinate reaches the bottom comparison value in cycle 63, the vertical border flip-flop is set.
3. If the Y coordinate reaches the top comparison value in cycle 63 and the DEN bit in register $d011 is set, the vertical border flip-flop is reset.
4. If the X coordinate reaches the left comparison value and the Y coordinate reaches the bottom one, the vertical border flip-flop is set.
5. If the X coordinate reaches the left comparison value and the Y coordinate reaches the top one and the DEN bit in register $d011 is set, the vertical border flip-flop is reset.
6. If the X coordinate reaches the left comparison value and the vertical border flip-flop is not set, the main flip-flop is reset.

So the Y coordinate is checked once or twice within each raster line: In cycle 63 and if the X coordinate reaches the left comparison value.

By appropriate switching of the CSEL/RSEL bits you can prevent the comparison values from being reached and thus turn off the border partly or completely (see 3.14.1.).

3.10 DISPLAY ENABLE

The DEN bit (Display Enable, register $d011, bit 4) serves for switching on and off the text/bitmap graphics. It is normally set. The bit affects two functions of the VIC: The Bad Lines and the vertical border unit.

- A Bad Line Condition can only occur if the DEN bit has been set for at least one cycle somewhere in raster line $30 (see section 3.5.).
- If the DEN bit is cleared, the reset input of the vertical border flip-flop is deactivated (see section 3.9.). Then the upper/lower border is not turned off.

Clearing the DEN bit will normally prevent Bad Lines (and thus c- and g-accesses) from occurring and make the whole screen display the border color.

3.11 LIGHT PEN

On a negative edge on the LP input, the current position of the raster beam is latched in the registers LPX ($d013) and LPY ($d014). LPX contains the upper 8 bits (of 9) of the X position and LPY the lower 8 bits (likewise of 9) of the Y position. So the horizontal resolution of the light pen is limited to 2 pixels.

Only one negative edge on LP is recognized per frame. If multiple edges occur on LP, all following ones are ignored. The trigger is not released until the next vertical blanking interval.

As the LP input of the VIC is connected to the keyboard matrix as are all lines of the joystick ports, it can also be controlled by software. This is done with bit 4 of port B of CIA A ($dc01/$dc03). This allows to determine the current X position of the raster beam by triggering an LP edge and reading from LPX afterwards (the VIC has no register that would allow reading the X position directly). This can e.g. be used to synchronize raster interrupt routines on exact cycles.

The values you get from the LPX register can be calculated from the sprite coordinates of the timing diagrams in section 3.6.3. The reference point is the end of the cycle in which the LP line is triggered. E.g. if you trigger LP in cycle 20, you get the value $1e in LPX, corresponding to the sprite coordinate $03c (LPX contains the upper 8 bits of the 9-bit X coordinate).

The VIC can also additionally trigger an interrupt on a negative edge on the LP pin (see next section), likewise only once per frame.

3.12 VIC INTERRUPTS

The VIC has the possibility to generate interrupts for the processor when certain events occur. This is done with the IRQ output that is directly connected to the IRQ input of the 6510. The VIC interrupts are therefore maskable with the I flag in the processor status register.

There are four interrupt sources in the VIC. Every source has a corresponding bit in the interrupt latch (register $d019) and a bit in the interrupt enable register ($d01a). When an interrupt occurs, the corresponding bit in the latch is set. To clear it, the processor has to write a "1" there "by hand". The VIC doesn't clear the latch on its own.

If at least one latch bit and the belonging bit in the enable register is set, the IRQ line is held low and so the interrupt is triggered in the processor. So the four interrupt sources can be independently enabled and disabled with the enable bits. As the VIC - as described - doesn't clear the interrupt latch by itself, the processor has to do it before the I flag is cleared resp.

before the processor returns from the interrupt routine. Otherwise the interrupt will be triggered again immediately (the IRQ input of the 6510 is state-sensitive).

The following table describes the four interrupt sources and their bits in the latch and enable registers:

Bit	Name	Trigger condition
0	RST	Reaching a certain raster line. The line is specified by writing to register $d012 and bit 7 of $d011 and internally stored by the VIC for the raster compare. The test for reaching the interrupt raster line is done in cycle 0 of every line (for line 0, in cycle 1).
1	MBC	Collision of at least one sprite with the text/bitmap graphics (one sprite data sequencer outputs non-transparent pixel at the same time at which the graphics data sequencer outputs a foreground pixel)
2	MMC	Collision of two or more sprites (two sprite data sequencers output a non-transparent pixel at the same time)
3	LP	Negative edge on the LP input (light pen)

For the MBC and MMC interrupts, only the first collision will trigger an interrupt (i.e. if the collision registers $d01e resp. $d01f contained the value zero before the collision). To trigger further interrupts after a collision, the concerning register has to be cleared first by reading from it.

The bit 7 in the latch $d019 reflects the inverted state of the IRQ output of the VIC.

3.13 DRAM REFRESH

The VIC does five read accesses in every raster line for the refresh of the dynamic RAM. An 8-bit refresh counter (REF) is used to generate 256 DRAM row addresses. The counter is reset to $ff in raster line 0 and decremented by 1 after each refresh access.

So the VIC will access addresses $3fff, $3ffe, $3ffd, $3ffc and $3ffb in line 0, addresses $3ffa, $3ff9, $3ff8, $3ff7 and $3ff6 in line 1 etc.

Refresh addresses

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	REF7	REF6	REF5	REF4	REF3	REF2	REF1	REF0

3.14 EFFECTS/APPLICATIONS

The following sections will describe some graphical effects that can be achieved by applying the rules and mechanisms of the VIC described in the previous sections.

3.14.1 HYPERSCREEN

As explained in section 3.9., the VIC generates the screen border by comparing the beam coordinates with start and stop positions selected by the CSEL/RSEL bits. So the border is basically not displayed within a certain range of coordinates, but rather turned on and off at certain coordinates. If you now make sure by appropriately switching CSEL/RSEL that the coordinate comparison never matches, the border is e.g. never turned on and you can see the graphics in the border area that is normally covered by the border. The technique is called "hyperscreen" or "opening the border".

However, the graphics displayable in the border area is mainly limited to sprites, as the graphics data sequencer is in idle state in this area as no Bad Lines can occur outside of Y coordinates $30-$f7 (see section 3.5.). But you can also do something sensible with the graphics generated in idle state.

To turn off the upper/lower border, you proceed as follows:

1. Somewhere in the upper part of the screen, you switch to 25-line-border by setting the RSEL bit.
2. Now you wait until RASTER has reached a value in the range of 248-250. The vertical border flip-flop is still cleared as the comparison value for RSEL=1 is raster line 251.
3. Then you clear the RSEL bit. The comparator is switched and now sets the vertical flip-flop at line 247. But this line is already passed and thus the VIC "forgets" to turn on the vertical border.
4. After raster line 251 you set the RSEL bit again and repeat from step 2.

If you open the upper/lower border with this method, the left/right border still remains active in the "opened up" area. If you switch from RSEL=0 to RSEL=1 in the raster line range 52-54, the border is never turned off and covers the whole screen (this is the same when the DEN bit is cleared, but Bad Lines still occur). But this is not very sensible.

The left/right border can be turned off with the CSEL bit in a similar way. However, the timing is much more critical. Whereas for the vertical border, you have 4 raster lines time for the switch, for the horizontal border the change from CSEL=1 to CSEL=0 has to be exactly in cycle 56. Likewise, the horizontal border can be prevented from turning off by switching from

CSEL=0 to CSEL=1 in cycle 17.

If you want to open the left/right border in the upper/lower border area, you must either start with it before the vertical border flip-flop is set (i.e. outside of the upper/lower border), or also open the upper/lower border, because the main border flip-flop can only be reset if the vertical flip-flop is not set. If you compare both methods, you can verify that the vertical flip-flop controls the graphics data sequencer output: With the first method, only the background color is visible in the opened up upper/lower border area, whereas the second method displays the idle state graphics there.

3.14.2 FLD

When building the graphics out of text lines, the VIC orientates itself exclusively by the occurrence of Bad Lines: A Bad Line gives the "start signal" for the display of one text line. By appropriately changing YSCROLL (in register $d011) you can suppress and arbitrarily delay the Bad Line Condition (see 3.5.). So you can exactly control in which raster lines Bad Lines should occur and thus from which raster lines the VIC should start to display one text line each. The distance between two text lines can be arbitrarily increased if only you hold back the next Bad Line long enough. This effect is called "Flexible Line Distance" (FLD).

E.g. if you only allow three Bad Lines on the screen at raster lines $50, $78 and $a0, the VIC will also only display three text lines at these positions. The sequencer is in idle state in the lines between.

If you only delay the occurrence of the first Bad Line, you can scroll down the complete graphics display by large distances without moving a single byte in display memory.

3.14.3 FLI

Instead of delaying the occurrence of Bad Lines as for the FLD effect, you may also artificially create additional Bad Lines before the VIC has completed the current text line. This is especially interesting for the bitmap modes, as these use the data from the video matrix (which is read in the Bad Lines) for color information, so normally only single 8×8 pixel blocks can be colored individually in bitmap modes. But if you make every raster line a Bad Line by appropriately modifying YSCROLL, the VIC will read from the video matrix in every line and so it will also read new color information for every line.

This way, each of the 4×8 pixels of a block in multicolor mode can have a different color. This software-generated new graphics mode is called "Flexible Line Interpretation" (FLI) and is probably the most outstanding example of "unconventional" VIC programming.

There is however one problem: If you create a new Bad Line before the current text line has been finished, VCBASE is not incremented (see 3.7.2.). So the VIC reads from the same addresses in the video matrix as in the previous line. As you cannot change the video matrix fast enough with the processor, you have to switch the base address of the video matrix with the bits VM10-VM13 of register $d018 (unfortunately the Color RAM cannot be switched, so the color selection of the pixels is not completely free).

Besides, the access to $d011 to create the Bad Line may not happen until cycle 14 of each raster line, or else the RC would be cleared in every line and the bitmap display would not be as desired. But this also implies that the first three c-accesses of the VIC in each line do not read valid data, because the first c-access in cycle 15 requires that BA should already have

gone low in cycle 12 so that AEC can stay low in cycle 15 (AEC doesn't stay low until three cycles after the negative edge of BA, there is no way around that). But as the Bad Line was first created in cycle 14, it's true that BA is low in cycle 15 on the first c-access, but AEC is high and so the internal data bus drivers D0-D7 of the VIC are closed and as the chip is manufactured in NMOS technology, it reads the value $ff and not the video matrix data (the data bus drivers D8-D11 are indeed open, but this is explained in section 3.14.6. in more detail) which is visible as 24-pixel wide stripes on the left side of the screen.

Practically you store eight video matrices in memory that are used in the following way: In the first raster line the first line of the first matrix, in the second line the first line of the second matrix, etc..., in the eighth line the first line of the eighth matrix, in the ninth line the second line of the first matrix, etc. With these eight matrices you can line-wise cover a complete bitmap.

There are several flavors of the FLI mode, such as AFLI (Advanced FLI) which uses the standard bitmap mode and simulates color blends by similarly colored adjacent pixels, and IFLI (Interlaced FLI) that alternates between two frames in a sort of interlace mode.

3.14.4 LINECRUNCH

By manipulating YSCROLL, you have even more possibilities to control the Bad Lines. You may also abort a Bad Line before its correct completion by negating the Bad Line Condition within an already begun Bad Line before cycle 14. This his several consequences:

- The graphics data is in display state, so graphics are displayed.
- The RC is not reset. If you abort the very first line of a frame this way, the RC is still at 7 from the last line of the previous frame.
- In cycle 58 of the line the RC is still 7, so the sequencer goes to idle state and VCBASE is loaded from VC. But as the sequencer has been in display state within the line, VC has

been incremented after every g-access, so VCBASE has now been effectively increased by 40. The RC doesn't overflow, it stays at 7.

With this procedure you have reduced the display of a text line to its last raster line, because as VCBASE has been incremented by 40, the VIC continues with the next line. This effect if therefore called "Linecrunch": You can "crunch" single text lines with it.

If you now do this in every raster line, the RC will always stay at 7 and there will be no c-accesses, but VCBASE is incremented by 40 in every line. This eventually makes VCBASE cross the 1000-byte limit of the video matrix and the VIC displays the last, normally invisible, 24 bytes of the matrix (where also the sprite data pointers are stored). VCBASE wraps around to zero when reaching 1024.

By crunching whole text lines to one raster line each you have the possibility to quickly scroll the screen contents up by large distances without moving bytes in the graphics memory, in a similar way as you can scroll it down with FLD. The only disturbing side effect is that the crunched lines pile up at the upper screen border, looking awkward. But here you can use one of the invalid graphics modes to blank out these lines.

3.14.5 DOUBLED TEXT LINES

The display of a text line is normally finished after 8 raster lines, because then RC=7 and in cycle 58 of the last line the sequencer goes to idle state (see section 3.7.2.). But if you now assert a Bad Line Condition between cycles 54-57 of the last line, the sequencer stays in display state and the RC is incremented again (and thus overflows to zero). The VIC will then in the next line start again with the display of the previous text line. But as no new video matrix data has been read, the previous text line is simply displayed twice.

3.14.6 DMA DELAY

The most sophisticated Bad Line manipulation is to create a Bad Line Condition within cycles 15-53 of a raster line in the display window in which the graphics data sequencer is in idle state, e.g. by modifying register $d011 so that YSCROLL is equal to the lower three bits of RASTER.

The VIC will then set BA to low immediately in the next cycle, switch to display state and start reading from the video matrix (the processor is now stopped because BA is low and it wants to read the next opcode). However, AEC still follows ø2 for three cycles before also staying at low state. This behavior (AEC not until three cycles after BA) is hardwired in the VIC and cannot be avoided.

Nevertheless, the VIC accesses the video matrix, or at least it tries, because as long as AEC is still high in the second clock phase, the address and data bus drivers D0-D7 of the VIC are in tri-state and the VIC reads the value $ff from D0-D7 instead of the data from the video matrix in the first three cycles. The data lines D8-D13 of the VIC however don't have tri-state drivers and are always set to input. But the VIC doesn't get valid Color RAM data from there either, because as AEC is high, the 6510 is still considered the bus master and unless it doesn't by chance want to read the next opcode from the Color RAM, the chip select input of the Color RAM is not active. Instead, a 4-bit analog (!) switch, U16, connects the data bits D0-D3 of the processor with the data bits D8-D13. This connection is always in place when AEC is high and should allow the processor to access the Color RAM if desired. To make a long story short: In the first three cycles after BA went low, the VIC reads $ff as character pointers and as color information the lower 4 bits of the opcode after the access to $d011. Not until then, regular video matrix data is read.

These data are stored just as usual at the start of the internal video matrix/color line and VC is incremented after each following g-access (with the generation of the Bad Line, the sequencer has also been put into display state). The c- and g-accesses are continued until cycle 54. But as the accesses started in the middle of a line, less than 40 accesses took place so VC has been incremented by a total of less than 40 in this raster line and no longer is a multiple of 40 as it normally always is at the end of a raster line. Because of the working of VC (see section 3.7.2.), this "misalignment" is continued for all following lines. So the whole screen seems to be scrolled to the right by as much characters as the number of cycles the $d011 access was done after cycle 14. As the c-accesses within the line start later than in a normal Bad Line, this procedure is called "DMA Delay".

With this, it is possible to scroll the complete screen sideways by large distances (this also works with bitmap graphics as with text screens as the VC is also used for accessing the bitmap data) without having to move the graphics memory with the processor. If you now combine DMA Delay with FLD and Linecrunch, you can scroll complete graphics screens without using much computing time by almost arbitrarily large distances in all directions.

Experimenting with the DMA Delay (and with Bad Line effects in general) is also the best method to discover the internal functions of the VIC, especially of RC and VC, and to determine in which cycles certain things are done inside the VIC.

It should also be mentioned that DMA Delay can not only be achieved by manipulating YSCROLL but also with the DEN bit of register $d011. To do this, you have to set YSCROLL to zero so that raster line $30 becomes a bad line and switch DEN from reset to set in the middle of the line. This is because Bad Line can only occur if the DEN bit has been set for at least one cycle in line $30, and if YSCROLL is zero a Bad Line Condition will occur in line $30 as soon as DEN is set.

3.14.7 SPRITE STRETCHING

As the sprite circuitry is simpler than that for the text graphics, there are not as many special effects possible with sprites, but among them is a very interesting effect that takes advantage of the way the sprite Y expansion works: By modifying the MxYE bits in register $d017 it is not only possible to decide for every single sprite line if it should be doubled, but you can also have single lines repeated three or more times and so expand a sprite by arbitrary scaling factors.

This effect can be understood as follows (see section 3.8.1.):

Let's say that we are in cycle 55 of a raster line in which sprite 0 is turned on and whose Y coordinate matches the Y coordinate of the sprite, so we are in the line before the sprite is displayed. Suppose that the M0YE bit is set. The VIC will then turn on the DMA for sprite 0 and clear MCBASE and the expansion flip-flop. BA goes to low state so that the VIC is able to access in the second clock phases of cycles 58 and 59. In cycle 58, MC is loaded from MCBASE and so cleared as well, and the p-access for the sprite is done. Afterwards, the three s-accesses are carried out and MC is incremented after each access so it now has the value 3.

Now you wait for cycle 16 of the following line. As the expansion flip-flop is reset, MCBASE still stays at zero. Then you first clear the M0YE bit and thereby set the flip-flop, but immediately set the M0YE again. The flip-flop is now inverted in cycle 55, because M0YE is set, and is thus reset again (if the M0YE hadn't been cleared, the flip-flop would now be set).

But this is exactly the same state in which the VIC was also in cycle 55 of the previous line. So the VIC "thinks" that it is still in the first raster line of an expanded sprite line and (as MC is still zero) and it will read the first sprite line twice more from memory, three times in total: The first sprite line has been tripled.

Another interesting effect can be achieved by proceeding exactly as outlines above and not clearing the M0YE bit after cycle 16 but in the second phase of cycle 15. MCBASE will then only be incremented by 1 and the next sprite line is read from memory with MC=1..3, that is one byte higher than normal. This "misalignment" is continued in the complete display of the sprite. Therefore, the condition MC=63 for turning off the sprite DMA in cycle 16 is also not met and the sprite is effectively displayed twice in sequence. Not until the end of the second display, the DMA is turned off when MC is 63.

4 THE ADDRESSES 0 AND 1 AND THE $DE00 AREA

The address range $de00-$dfff of the 6510 (see 2.4.1.) is reserved for external expansions of the C64 and normally not connected with any other unit (RAM, I/O). A read access will fetch data that looks random at first sight. The same is true for the upper nibbles of the addresses $d800-$dbff (the Color RAM).

But on some C64, this data is not "random" at all but rather identical to the data that the VIC has read from memory in the first phase of the clock cycle. This effect is however not reproducible on all machines and not always precise.

Apart from the fact that this gives the possibility to measure the VIC timing completely by software (the timing diagrams in [4] on which the diagrams in this article are based, have e.g. been created with this method), you can also make the 6510 execute programs in the $de00-area or in the Color RAM if the VIC displays a graphics in such a way that the 6510 gets valid opcodes from the graphics data read by the VIC.

With a similar effect you can also write to RAM addresses 0 and 1 from the processor. They are normally not available as the internal data direction register and data register of the 6510 I/O port are mapped to these addresses, and the data bus drivers stay in tri-state on a write access.

But the R/W line is set to low state (this can be explained as the I/O port has been integrated afterwards into the existing design of the 6502) and so the byte read by the VIC in the first clock phase is written to RAM. If you want to write a certain value to addresses 0 or 1 you only have to write an arbitrary value to these addresses and take care that the VIC read the desired value from RAM in the clock phase before.

The addresses 0 and 1 can of course also be read by the processor. Either with via the $de00-area or with the aid of sprite collisions. For this, you make the VIC display a bitmap starting at address 0 and move a sprite consisting only of one pixel over the single bits of the first two bytes of the bitmap. Depending on whether a collision has been detected or not, you can find out the state of the single bits and put them together to one byte.

APPENDIX A: BIBLIOGRAPHY

[1] Commodore Business Machines, "C64 Programmers Reference Guide", Appendix L: "6510 microprocessor data sheet", 1984

[2] dto., Appendix N: "6566/6567 Video Interface Controller (VIC-II) Chip Specifications"

[3] dto., Chapter 5, Section "Memory management on the Commodore 64"

[4] Marko Mäkelä, "The memory accesses of the MOS 6569 VIC-II and MOS 8566 VIC-IIe Video Interface Controller" (AKA: Pal timing), 15.07.1994

[5] John West, Marko Mäkelä, "Documentation for the NMOS 65xx/85xx Instruction Set" (AKA: 64doc), 03.06.1994

[6] Commodore Semiconductor Group, "6567 Video Interface Chip Specification", 1973

APPENDIX B: ACKNOWLEDGMENTS

I want to thank

- Marko Mäkelä, Andreas Boose, Pasi Ojala and Wolfgang Lorenz for the amount of work they put into the examination of the VIC

- Kaspar Jensen for proof-reading the English version of this document and for his suggestions

- Adam Vardy <abe0084@InfoNET.st-johns.nf.ca> for pointing out that my description of idle state graphics display was wrong