VBXE FPGA core "FX"
version 1.0 beta 7

Programmer's Manual

Copyright © 2009 by T.Piórek

Table of contents

THE XDL

The XDL (eXtended Display List) is a list of commands, that controls the OVERLAY display and the attribute map of the VBXE. The XDL is loaded to the VBXE memory through the MEMAC buffers (see the MEMAC description). It may be loaded to any location inside the 512KB VBXE VRAM, and it is pointed to by the registers XDL_ADR0, XDL_ADR1 i XDL_ADR2. The XDL processing starts, when the bit XDL_ENABLED in the VIDEO_CONTROL register is set to 1. There are no limitations on the XDL's size, and its structure is vertical: as it is the case of the ANTIC DL, the XDL processing „starts" at the top of the display.

The XDL is structured as follows:

XDLC (2 bytes)
additional data (0-20 bytes)
XDLC (2 bytes)
additional data (0-20 bytes)
...
...
XDLC with the XDLC_END marker (2 bytes)
additional data (0-20 bytes)

The „XDLC" stands for the XDL Control word. This word always occupies two bytes. Every bit of the control word has different meaning (see the Table 1). A part of the bits enables or disables display functions, and another part of them carries an information, whether the XDL Controller should fetch additional data, and what data it would be. The bits do not depend on each other, any combination of them can be used at any time; it is, for example, possible to load the Overlay video memory address leaving the Overlay switched off. The XDLC word gets processed before the line starts to be displayed.

byte.bit XDLC	bit’s label	meaning	additional data
1.0	XDLC_TMON	enable Overlay Text Mode	-
1.1	XDLC_GMON	enable Overlay Graphic Mode	-
1.2	XDLC_OVOFF	disable Overlay	-
remarks:	Setting more than one of the bits 0.0, 0.1 and 0.2 will cause the Overlay to be switched off. The Overlay is disabled by default (at the top of the screen). Leaving all of these bits zeroed will preserve the current state of the overlay. It can be useful when you only want to change the font or scrolling values.		
1.3	XDLC_MAPON	enable colour attributes	-
1.4	XDLC_MAPOFF	disable colour attributes	-
remarks:	Setting more than one of the bit 0.3 and 0.4 will disable the attributes. It is also disabled by default, i.e. at the top of the screen. Leaving all of these bits zeroed will preserve the current state of the map. It can be useful when you only want to change the font or scrolling values.		
1.5	XDLC_RPTL	no changes in next x scanlines	number of scanlines (x) (1 byte)
remarks:	After the current line was displayed, there will be x consecutive scanlines to use the same settings, and XDL will not be processed for them. For example, if you want to display a line of text, enable the text mode with XDLC_TMON, and set XDLC_RPTL giving 7 as x. As a result 8 scanlines will be produced forming a line of the text mode.		
1.6	XDLC_OVADR	set the address and step of the Overlay display memory	5 bytes (3 byte address and 2 byte step), little endian.
remarks:	The address of the Overlay display memory is a 19-bit value. The *step* parameter defines, how many bytes should be added to the address, so that it would point to the data for the next line. The *step* may be a value from 0 to 4095. The order of the additional data: 1. OVADR[7:0] 2. OVADR[15:8] 3. OVADR[18:16] 4. OVSTEP[7:0] 5. OVSTEP[11:8] In a pixel mode the OVSTEP gets added to the OVADR after every scanline. In the text mode the OVSTEP is added to the OVADR when eight lines of the character have been displayed.		
1.7	XDLC_OVSCRL	Set scrolling values for the text mode	2 bytes: 1. hscroll (1 byte) 2. vscroll (1 byte)

byte.bit XDLC	**bit's label**	**meaning**	**additional data**
remarks:	hscroll is a value ranged 0 ... 7, where 0 is a not scrolled line, and 7 is a line scrolled 7 pixels to the left. vscroll is a value ranged 0 ... 7, where 0 is a not scrolled line, and 7 is a line scrolled 7 pixels up. By default, at the top of the screen, hscroll = vscroll = 0. Scrolling values can be changed in every scanline. Setting the XDLC_OVSCRL does not enable the scroll (which is always on), but only sets the VALUES OF THE SCROLLING REGISTERS. These values will be used for every consecutive scanline until the XDL changes them. The horizontal scrolling unit is 1 pixel VBXE hires (or 0.5 pixel GR.8).		
2.0 (XDLC 2nd byte)	XDLC_CHBASE	set character base	1 byte = font address
remarks:	The font contains 256 characters, 8x8 pixels each, and should be loaded to the VBXE memory. Every font must start at a 2K boundary, therefore up to 256 fonts can be loaded to the 512K VRAM. As everything else, the font is stored in the VBXE memory through the MEMAC buffers.		
2.1	XDLC_MAPADR	set the address and step of the colour attribute map	5 bytes (3 byte address and 2 byte step), little endian.
remarks:	The colour attribute map may start at any location in the VBXE memory. Data order: 1. AMAP[7:0] 2. AMAP[15:8] 3. AMAP[18:16] 4. MAPSTEP[7:0] 5. MAPSTEP[11:8] The MAPSTEP value is automatically added to the AMAP address when the attribute field has been completely displayed vertically (i.e. after displaying the last – bottom – scanline of the field), unless the settings get explicitly changed by the XDL.		

byte.bit XDLC	bit's label	meaning	additional data
2.2	XDLC_MAPPAR	set scrolling values, width and height of a field in the colour attribute map	4 bytes: 1. hscroll (1 byte) 2. vscroll (1 byte) 3. width (1 byte) 4. height (1 byte)
remarks:	hscroll is a value of range <0 ... 31>, where 0 means that the line is not scrolled, and 31 – that the line is scrolled 31 pixels to the left. vscroll is a value of range <0 ... 31>, where 0 means that the line is not scrolled, and 31 – that the line is scrolled 31 pixels up. width – the width of the field in pixels, range <7 ... 31> == 8 to 32 pixels (as in ANTIC GR.8) height – the height of the field in scanlines <0 ... 31> == 1 to 32 lines hscroll and vscroll for the map should never get greater than the respective values of width and height. Default values (at the top of the screen) are: hscroll = vscroll = 0; height = width = 7; (the field size 8x8) The field size and scrolling values may be changed in any scanline. The hscroll unit for the map is 1 pixel GR.8. Setting XDLC_MAPPAR does not enable the map to scroll (this function is always on), it only loads the scrolling registers. The values loaded will be used in consecutive scanlines until they are explicitly changed with XDL.		

byte.bit XDLC	bit’s label	meaning	additional data
2.3	XDLC_OVATT	Setting the display size (both Overlay and Colour map) and Overlay priority to the ANTIC display. And Overlay colour modification.	2 bytes: 1. Overlay / map width + Overlay colour modification 2. main priority
remarks:	BYTE 1: OV_WIDTH: OVERLAY and COLOUR MAP width: 0 = NARROW (256 pixels, as ANTIC narrow) 1 = NORMAL (320 pixels,as ANTIC normal) 2 = WIDE (336 pixels, as ANTIC wide; in this mode the display is 8 pixels wider at both sides, than NORMAL) The default (at the top of the screen) width is NORMAL (320 pixels). OV_COLOR_SHIFT: primary modification of the OVERLAY pixel colour. Valid in all Overlay modes. If the colour is not transparent (see „Transparent Overlay colours”), then: **effective_color = (palette_number * 256) +** **+ ((OV_COLOR_SHIFT * 16) | color_selected_by_Overlay_data);** where the "|" is bitwise OR. This allow to increase the number of displayable colours, especially in the VBXE hires pixel mode. OV_COLOR_SHIFT value is 0 by default. The possible values are from 0 to 15. BAJT 2: Main priority. b0 - 1 = OVERLAY over PM0, 0 = OVERLAY overlaid by PM0 b1 - 1 = OVERLAY over PM1 b2 - 1 = OVERLAY over PM2 b3 - 1 = OVERLAY over PM3 b4 - 1 = OVERLAY over PF0 b5 - 1 = OVERLAY over PF1 b6 - 1 = OVERLAY over PF2 b7 - 1 = OVERLAY over PF3 The default value (at the top of the screen) of the priority is 255. The main priority is not taken into account, when the attribute map is enabled. In this case it is the map, that decides, which one of the 4 predefined priorities P0 ... P3 will be used for the particular part of the screen.		
2.4	XDLC_HR	enable the Hi-Res pixel mode	-

byte.bit XDLC	bit's label	meaning	additional data
remarks:	This bit is only taken into account, when XDLC_GMON == 1. The HR mode (or hires) has a resolution of 640 pixels horizontally for the NORMAL display width and can display 16 colours, from $00 to $0F, in the current Overlay palette (of course, the OV_COLOR_SHIFT can be used as well). Each pixel is represented by a nibble of data (4 bits) in the VBXE memory. Each data byte contains 2 nibbles: the most significant nibble represents the leftmost pixel.		
2.5	XDLC_LR	enable the Low Resolution mode	-
	This bit is only taken into account, when XDLC_GMON == 1. The LR mode has a resolution of 160 pixels horizontally for the NORMAL display width. The number of displayable colours is the same as in the standard display mode (256).		
2.6	-	reserved (=0)	-
2.7	XDLC_END	XDL end (the last XDL record), wait for VSYNC.	-
remarks:	XDLC_END tells the XDL controller, than after processing of this XDLC is finished, it has to wait for the vertical sync pulse, and then start processing the XDL from the beginning.		

OVERLAY MODES

The bit combos that enable the Overlay display modes of the VBXE:

XDLC_TMON	XDLC_GMON	XDLC_HR	XDLC_LR	mode
0	1	0	0	Pixel SR (320 / 256c)
0	1	1	0	Pixel HR (640 / 16c)
0	1	0	1	Pixel LR (160 / 256c)
0	1	1	1	Forbidden
1	0	X	X	80-column text mode
1	1	X	X	Forbidden, works as XDLC_OVOFF

Pixel modes

The SR mode (Standard Resolution)

This is a pixel mode that can display 256/320/336 pixels horizontally (the width is selected via XDL, the vertical resolution is defined by the XDL structure) in 256 colours. Every pixel is represented by a byte in VBXE memory. If the value of this byte is 0, then the pixel is not displayed (it is transparent, unless the no_trans bit in the VIDEO_CONTROL register is set to 1). The other values select the colour from the current Overlay palette, and the value of the byte is the colour number. After displaying a scanline, the VBXE automatically increases the address of the data to fetch from the screen memory adding the *step* value, ranged 0 ... 4095, as defined by the XDL.

The LR mode (Low Resolution)

This is a pixel mode with horizontal resolution of 128/160/168 pixels. All other characteristics are as in the SR mode.

The HR mode (High Resolution)

This is a pixel mode with horizontal resolution of 512/640/672 pixels (the width is selected via XDL, the vertical resolution is defined by the XDL structure) in 16 colours.

A byte of the video memory contains information about 2 pixels, 4 bits each:

b7	b6	b5	b4	b3	b2	b1	b0
Leftmost pixel				Rightmost pixel			

The transparency is selected, when the nibble value is 0.

Each pixel selects the colour 0 ... 15 from the currently selected (locally or globally) Overlay palette.

The text mode

This is a text mode with horizontal resolution of 64/80/84 characters (the width is selected via XDL, the vertical resolution is defined by the XDL structure) in 128 or 16+8 colours. The video memory structure is as follows:

char (1 byte), attribute (1 byte), char, attribute, char, attribute, ... and so on.

The char is a value 0-255 and defines which character of the 256-character font will be displayed.

The attribute has the following structure:

b7 – decides, whether the character's background is transparent or it has a colour.

when b7 = 0:

b7	b6	b5	b4	b3	b2	b1	b0
0	foreground (pixels set to 1) colour = $00 .. $7F						

b0 ... b6 = colour number (0 ... 127) for the character, i.e. colours 0...127 from the active (locally or globally) Overlay palette.

The character's background is transparent, if the no_trans bit in the VIDEO_CONTROL register is cleared (0) – or it has the colour no. 128 otherwise.

if b7 = 1:

b7	b6	b5	b4	b3	b2	b1	b0
1	foreground (pixels set to 1) colour = $00 .. $7F background colour = $80 .. $FF (foreground colour + $80)						

b0 ... b6 = colour number for the character (0...127 for the foreground and 128...255
for the background), i.e. colours 0-255 from the active (locally or globally) Overlay palette.

The background is not transparent.

In other words:

colour of enabled pixels: always 0 ... 127 (attribute value & 127)
colour of disabled pixels:

```
a) when VC bit 2 (no_trans) == 0
    if (attribute < 128) -> transparent background
    otherwise background colour = (attribute & 127) + 128 (i.e. 128 ... 255)
b) when VC bit 2 (no_trans) == 1
    if (attribute < 128) -> background colour = 128
    otherwise background colour = (attribute & 127) + 128 (i.e. 128 ... 255)
```

The full line of the text mode occupies the number of bytes in the memory equal to 2x line width in characters. Additionally the line can be expanded by 1 byte because of the hscroll.

Text mode scroll

See the XDL description.

Special techniques applying to Overlay modes

It is possible to enable additional transparent colours, see the section „**Transparent Overlay colours**”, and the primary colour modification in XDL, see „**Overlay colour modification**”.

Overlay colour modification

The number of colours in any of the Overlay modes may be increased using two mechanisms:

- OV_COLOR_SHIFT

  The XDL-defined register OV_COLOR_SHIFT (values 0 ... 15) allows to change (per

  scanline) the colour resulting from the Overlay data according to the following formula:

  effective_colour = (OV_COLOR_SHIFT * 16) | Overlay_data;

  with the following exception:

  If the (Overlay_data == 0 && VIDEO_CONTROL.no_trans == 0) – i.e. when the colour is transparent, then it remains transparent.

- changing the current palette:

  There are four user-defined palettes, 256 colours each. The effective colour of the Overlay is selected by expanding this formula:

  effective_colour *(see above)*

  with 2 palette-selection bits:

  - when the colour attribute map is disabled, the Overlay always uses the palette no. 1.
  - when the colour attribute map is active, then each field allows to select palette 0 ... 3 independently.

Transparent Overlay colours

I. When the no_trans bit in the VIDEO_CONTROL register is set to 1, then no Overlay colour is transparent, either in the pixel modes or in the text mode (regardless of the trans15 bit state in the VIDEO_CONTROL register).

II. When the no_trans bit in the VIDEO_CONTROL register is set to 0 and the trans15 bit in the same register is cleared (which is the default), then:

- SR / LR pixel modes: colour „0” is transparent.
- HR pixel mode: the colour selected by the nibble that has a value of „0” is transparent
- text mode: the background is transparent, if the bit 7 of the attribute is cleared.

III. When the no_trans bit in the VIDEO_CONTROL register is cleared and the bit trans15 in the VIDEO_CONTROL register is set, then colours are transparent as described in the paragraph II, and additionally:

- SR / LR pixel modes: each colour with palette index $HF (where H = 0...F, i.e. $0F, $1F, $2F and so on up to $FF) is transparent;
- HR pixel mode: colour index $F is transparent;
- text mode: each colour with palette index $HF (where H = 0...F, i.e. $0F, $1F, $2F etc.. up to $FF) is transparent.

The trans15 bit allows to create invisible objects, which can be used, for example, as fields to detect collisions with other Overlay objects.

Priorities OVERLAY modes <-> ANTIC/GTIA modes

Setting priorities between the Overlay display and the ANTIC/GTIA display goes as follows:

Every GTIA colour register (except COLBAK) has a corresponding bit in the VBXE priority register. The state of this bit decides, whether this colour (if it is being displayed by the ANTIC/GTIA) has to be overridden by the Overlay colour, or vice versa. Apart from that, the only colour always overridden by the Overlay is COLBAK.

See also the XDL description (meaning of bits in the main priority register).

The main priority register is loaded by the XDL. Additionally, if the colour attribute map is on, the main priority register state is overridden by one of the four predefined priority registers selected for the particular colour field of the colour attribute map. The predefined registers are set through the MSEL/PRIORMAP (see the description of the colour attribute map and MSEL/PRIORMAP).

In the GTIA modes known as GR.9 and 11 (16 shades or 16 hues) the Overlay is always displayed over the ANTIC/GTIA display (but in turn it may be overlaid by the P/MG).

Detecting collisions between the OVERLAY and ANTIC/GTIA

It is possible to detect collisions between the data displayed by the Overlay (either in pixel or character mode) and any of the GTIA colours COLPM0/1/2/3 and COLPF0/1/2/3.

The collision detection is performed automatically while the VBXE is producing the display.

The collision is detected when the bitwise AND of the local Overlay colour (without taking the OV_COLOR_SHIFT into account) and the mask loaded to the COLMASK register results in a non-zero value, and in the same screen place any of the COLPFx or COLPMx colours is being displayed.

The collision code can be fetched from the COLDETECT register.

Clearing the collision state is accomplished by writing any value to the COLCLR register.

Want to know more? See the VBXE core register description.

NOTE: this mechanism should not be confused with the mechanism of detecting Overlay-Overlay collisions offerred by the Blitter.

The order of fetching data in the XDL

The additional XDL data (addresses, scrolling registers etc.) are fetched or they are not, depending on the states of the corresponding bits in the XDLC (see the XDL description). The order of them in the memory is always the same, data corresponding to the lower bits of the XDLC are fetched before the data corresponding to the higher bytes of the XDLC:

- XDLC_RPTL (1 byte)
- XDLC_OVADR (5 bytes)
- XDLC_OVSCRL (2 bytes)
- XDLC_CHBASE (1 byte)
- XDLC_MAPADR (5 bytes)
- XDLC_MAPPAR (4 bytes)
- XDLC_OVATT (2 bytes)

After the XDLC word there may be maximum 20 bytes of data.

Example: an XDL that creates 16 scanlines (or 2 lines) of the text mode.

```
XDLC equ XDLC_TMON + XDLC_RPTL + XDLC_OVADR+XDLC_CHBASE + XDLC_OVATT + XDLC_END

.word XDLC

.byte 15        ;how many scanlines without a change (xdlc_rptl)
.long adr       ;3-byte screen memory address (xdlc_ovadr)
.word 160       ;automatic step (xdlc_ovadr)
.byte $20       ;CHBASE $20 * $800
.byte 0         ;(xdlc_ovatt) - narrow Overlay
.byte 255       ;(xdlc_ovatt) – the highest priority of the Overlay
```

THE COLOUR ATTRIBUTE MAP

The colour attribute map allows to locally (i.e. within a field of 8x1 up to 32x32 pixels of GR.8) change the colours PF0, PF1 and PF2, override the main Overlay priority over the ANTIC/GTIA to one of four predefined priorities, change the local colour palette for both ANTIC and Overlay screen modes, and change the resolution of the display generated by the ANTIC/GTIA from ANTIC hires (GR.8) to CCR (Colour Clock Resolution = GR.15) or vice versa.

Consequently, the attribute map greatly extends the graphic capabilities of your Atari machine even if the proper Overlay modes are not in use.

Characteristics:

- the field size (X x Y): 8x1 up to 32x32 pixels of GR.8 (or ANTIC hires).
- map address in the VBXE VRAM: no limitations.
- automatic update of the address after displaying a full line of the map: programmable in the range 0 ... 4095 bytes.
- horizontal and vertical scrolling by 1 pixel of ANTIC hires, controlled by XDL. It is possible to change the register values in any scanline.
- every map field is defined by a set of 4 bytes stored consecutively in the VRAM. The bytes define as follows:
  - byte 1: local colour PF0
  - byte 2: local colour PF1
  - byte 3: local colour PF2
  - byte 4:
    - local ANTIC<>OVERLAY priority (1 of 4 predefined ones)
    - local resolution change
    - local colour palette for ANTIC and Overlay display modes (independently). The choice is between 4 palettes. ANTIC and Overlay may use either the same or different palettes in the scope of the map field.

The attribute map is completely controlled by XDL, i.e. its VRAM address, scrolling registers, field size etc. are defined inside the XDL list.

Attribute data

Every field of the map is defined in the VBXE VRAM by four consecutive bytes:

b7	b6	b5	b4	b3	b2	b1	b0
Local substitute of the COLPF0 register (GTIA)							

b7	b6	b5	b4	b3	b2	b1	b0
Local substitute of the COLPF1 register (GTIA)							

b7	b6	b5	b4	b3	b2	b1	b0
Local substitute of the COLPF2 register (GTIA)							

b7	b6	b5	b4	b3	b2	b1	b0
ANTIC/GTIA palette		OVERLAY palette		-	RES	PSEL1	PSEL0

b6, b7 – local palette selection for normal Playfield and P/MG objects. When the attribute map is active, it may be any of the four palettes. When the attribute map is disabled, it will always be the palette 0.

b4, b5 – local palette selection for the Overlay. When the attribute map is active, it may be any of the four palettes. When the attribute map is disabled, it will always be the palette 1.

b3 - reserved (unused) – write 0 here.

b2 (RES) – local change ANTIC HIRES <-> CCR (1 = enabled). This bit „reverses” the resolution selected by the standard ANTIC DL, changing the mode locally (within the particular field of the map) from ANTIC hires into CCR or vice versa.

b0, b1 (PSEL0 / PSEL1) – selection of one of the 4 predefined priorities OVERLAY <-> ANTIC/GTIA (see also MSEL/PRIORMAP).

PSEL1	**PSEL0**	**register selected**
0	0	Priority register P0 (loaded to MB0)
0	1	Priority register P1 (loaded to MB1)
1	0	Priority register P2 (loaded to MB2)
1	1	Priority register P3 (loaded to MB3)

NOTE: Within the active attribute map the global GTIA colour registers: COLPF0, COLPF1 i COLPF2 and the global priority register are not used.

NOTE: The width of the attribute map may be forced to correspond to the wide/normal/narrow ANTIC display. This is accomplished using the XDL, see XDLC_OVATT.

MSEL/RGB

This mechanism allows to change the RGB components of the palette.

The VBXE allows to display up to 1024 colours at one time, out of the 21-bit hardware palette (2097152 colours). There are 4 sets (or palettes), 256 user-defined colours each.

The RGB palette for each of these 1024 (4 x 256) colours may be updated by writing the RGB components (8 bits each, the lowest significant bit = 0) into core registers, MB1, MB2 and MB3 respectively. Done that the colour index in the palette (0 – 255) should be loaded into the MB0 register, and to the MSEL – the value of $C0 + palette number (0 ... 3).

That is:

MSEL = $c0 for palette 0, or
MSEL = $c1 for palette 1, or
MSEL = $c2 for palette 2, or
MSEL = $c3 for palette 3.

All RGB components of the selected colour are updated when the MSEL register is written to.

By default, when the attribute map is inactive, the palette 0 is assigned to the ANTIC/GTIA modes and to the P/MG, and palette 1 is assigned to the Overlay. Palettes 2 and 3 are unused. This changes, when the attribute map gets activated. Every field of the map allows to define, which one of the four palettes will be assigned to Overlay pixels, and which one – to the ANTIC/GTIA and P/MG pixels.

After power-up the palette 0 is loaded by the default, or „factory”, palette, which is a modified version of the laoo.act palette used by the Atari800 emulator. The palettes 1-3 are zeroed (black colours only). Remember, that a program loaded previously could have changed the factory values in any of the palettes!

NOTE: the Overlay modes (pixel and text) and ANTIC/GTIA/P/MG do colour indexing always using 8 bits. Next, depending whether:

- *the colour map is active*
- *the colour is generated by ANTIC/GTIA or Overlay*

the proper palette is assigned (by expanding the colour index with additional two bits).

MSEL/PRIORMAP

This mechanism allows to set the four priority registers that define priorities between ANTIC/GTIA and Overlay displays. These registers are used and selected by the attribute map and override the main priority register selected by the XDL.

The attribute map allows (using bits b0 and b1 in the every fourth byte of the map) to choose one of the redefined priorities for the current field.

- load P0 priority definition to the MB0 register
- load P1 priority definition to the MB1 register
- load P2 priority definition to the MB2 register
- load P3 priority definition to the MB3 register
- load $80 to the MSEL register. On this write the Px registers will get updated.

The bits in each of the priority register is the same as in the main priority value defined by the XDL (see the XDL description), which is in use when the attribute map is inactive (locally or globally). Within the scope of the active attribute map the main priority remains unused.

MEMAC

The MEMAC is a part of the VBXE core responsible for allowing the system (CPU and ANTIC) access to the 512 KB VRAM installed inside the VBXE.

An access to the VRAM made through the MEMAC costs VBXE 1 cycle of the PCLK clock (14.18 MHz) per byte being read or written. Looking from the side of the CPU or ANTIC, there is no difference between this access an any other access to the memory or I/O registers. Technically the VBXE disables the standard RAM and substitutes own memory using the EXTSEL signal of the MMU.

The VBXE memory access can be accomplished through „windows” available in two memory areas:

$2000 - $3FFF - "MEMAC A" (an 8 KB window)
$4000 - $7FFF - "MEMAC B" (a 16 KB window)

Either of these two may be:

- disabled (the standard Atari RAM is visible there then)
- enabled for the CPU (the CPU sees the VRAM, the ANTIC sees the Atari RAM)
- enabled for the ANTIC (the ANTIC sees the VRAM, the CPU sees the Atari RAM)
- enabled for the CPU and for the ANTIC

Additionally, the bank selection is independent for the CPU and for the ANTIC, i.e. the CPU and ANTIC can see different VRAM banks.

The MEMAC A window has 8 KB -> the VRAM is subdivided into 64 banks.
The MEMAC B window has 16 KB -> the VRAM is subdivided into 32 banks.

The special function of the MEMAC is to emulate the standard RAM extension known as 320K RAMBO (64 KB conventional + 256 KB extended RAM, without the separate access for the CPU and ANTIC). As the address range of the RAM extension is the same as the address area of the MEMAC B window ($4000-$7FFF in the Atari memory), the extension gets disabled, when the MEMAC B window is activated. The RAMBO extension is mapped to the upper half of the VBXE VRAM ($40000-$7FFFF in the VBXE VRAM). You should remember, that this area is shared by the RAMBO emulation and VBXE VRAM (but there is also a version of the core, which does not emulate the RAMBO extension).

Controlling registers

MA_CPU

bit 7 - 1 = CPU sees VBXE RAM inside MEMAC A area
        0 = CPU sees the Atari RAM there
bit 6 - reserved
bits 0 - 5 select one of the 64 8 KB VBXE RAM banks.

MA_ANTIC

bit 7 - 1 = ANTIC sees the VBXE RAM inside MEMAC A area
        0 = ANTIC sees the Atari RAM there
bit 6 - reserved
bits 0 - 5 select one of the 64 8 KB VBXE RAM banks

MB_CPU
bit 7 - 1 = CPU sees VBXE RAM inside MEMAC B area
        0 = CPU sees the Atari RAM there
bit 6 - reserved
bit 5 - reserved
bits 0 - 4 select one of the 32 16 KB VBXE RAM banks

MB_ANTIC
bit 7 - 1 = ANTIC sees VBXE RAM inside MEMAC B area
        0 = ANTIC sees the Atari RAM there
bit 6 - reserved
bit 5 - reserved
bits 0 - 4 select one of the 32 16 KB VBXE RAM banks

MAPPING ATARI MEMAC A/B ADDRESSES INTO VBXE VRAM

The VRAM addresses are 19-bit (512 KB). You can calculate the VRAM effective address from the MEMAC bank number and the offset inside the MEMAC window. It is important, because the VBXE core often requires the VRAM effective address to be given, therefore we have to know, where the data has been loaded to within the VRAM address space.

MEMAC A:
VRAM[18:0] = {A_BANK_NUMBER[5:0],A[12:0]}

MEMAC B:
VRAM[18:0] = {B_BANK_NUMBER[4:0],A[13:0]}

Example: write to VRAM starting at $12800 using the MEMAC A window.

A_BANK_NUMBER[5:0] = $12800 / $2000 = 9
ATARI_ADDRESS = $2000 + ($12800 & $1fff) = $2000 + $800 = $2800

Example: write to VRAM starting at $12800 using the MEMAC B window.

B_BANK_NUMBER[4:0] = $12800 / $4000 = 4
ATAR_ADDRESS = $4000 + ($12800 & $3fff) = $4000 + $2800 = $6800

BLITTER

The Blitter built into the VBXE core allows to copy and fill VRAM areas of any size.

The Blitter is controlled by the BlitterList – a sequence of data loaded into the VRAM by the Atari CPU. The general structure of the BlitterList looks as follows:

BCB
BCB
BCB
...
BCB with NEXT-marker cleared

The BCB stands for „Blitter Command Block”. The BlitterList consists of one or more BCBs. The BCB is a set of information for the Blitter. Each BCB defines one blitter operation. The BCB is 19 bytes long.

Byte	Name	Description
1	source_adr0	bits 0 ... 7, source address
2	source_adr1	bits 8 ... 15, source address
3	source_adr2	bits 16 ... 18, source address
4	source_step0	bits 0 ... 7, source step
5	source_step1	bits 8 ... 11, source step
6	dest_adr0	bits 0 ... 7, destination address
7	dest_adr1	bits 8 ... 15, destination address
8	dest_adr2	bits 16 ... 18, destination address
9	dest_step0	bits 0 ... 7, destination step
10	dest_step1	bits 8 ... 11, destination step
11	blt_width0	bits 0 ... 7, object width (in bytes)
12	blt_width1	bit 8, object width (in bytes)
13	blt_height	bits 0 ... 7, object height (in lines)
14	blt_and_mask	AND-mask for source data
15	blt_xor_mask	XOR-mask for source data
16	blt_collision_mask	AND-mask for collision detection
17	blt_zoom	X- and Y- axis zoom of the object being copied
18	pattern_feature	pattern fill
19	blt_control	additional information (see below)

source_adr

The source data for the Blitter operation may be located at any address inside the VBXE memory.

source_step

This parameter defines, how many bytes to add to, or subtract from the source_adr after the horizontal line of the blt_width width has been copied.

source step = 0...4095

dest_adr

The destination data for the Blitter operation may be located at any address inside the VBXE memory.

dest_step

This parameter defines, how many bytes to add to, or subtract from the dest_adr after the horizontal line of the blt_width width has been copied.

dest step = 0...4095

blt_width

The width of the object being copied (measured in BYTES), less 1.

blt_width = 0...511. This corresponds to the width of 1...512 bytes, and in the Overlay modes SR and LR this means 1 ... 512 pixels. In the HR mode this is 2 ... 1024 pixels.

blt_height

The height of the object being copied (measured in lines), less 1.

blt_height = 0 ... 255, i.e. 1 ... 256 lines

blt_and_mask

Clearing bits in the source data:

source' = source AND blt_and_mask

Every byte of the source data undergoes this operation. The „source” in the equation above means the source data byte having been fetched by the Blitter.

blt_xor_mask

Reverting bits in the source data:

source'' = source' XOR blt_xor_mask

Every byte of the source data undergoes this operation.

blt_collision_mask

Collision mask. The collision detection is performed in modes 1, 2, 3, 4, 5 and 6 (see the description of blt_control) according the the following equation:

```
if (source" != 0 && (blt_collision_mask & dest) != 0) BLT_COLLISION_CODE = dest;
```

where „dest” stands for a prefetched destionation data (before it is processed and written back). Mode 6 uses bit 0 ... 3 of the blt_collision_mask, other modes use all bits. BLT_COLLISION_CODE is one of the core registers (see below).

blt_zoom

It is possible to resize the object horizontally and vertically. This is accomplished by multiplying its width and height by a constant 1 ... 8.

b7	b6	b5	b4	b3	b2	b1	b0
	BLT_ZOOMY			INTLVE	BLT_ZOOMX		

The source data remains unchanged (blt_width and blt_height refer to the source object size in bytes), it is the destination area that gets enlarged. The enlargement is equal to:

ZOOMX(Y) = BLT_ZOOMX(Y) + 1

The INTLVE bit controls the DESTINATION address modification, while a „horizontal” data line is being copied:

if the INTLVE bit is 0, then the address is modified (increased or decreased, see DX bit in blt_control) by 1.
if the INTLVE bit is 1, then the address is modified by 2.

As a result, when INTLVE = 1, the blitter writes to each second byte of the destination area leaving the rest unchanged. Thanks to that it is possible to independently manipulate character data and attributes in the text mode.

NOTE: the skipped bytes do not count into blt_width, so, if the INTLVE = 1, to perform the operation on the entire line of the 80-column text mode, one should set the blt_width to 79, instead of 159.

pattern_feature

b7	b6	b5	b4	b3	b2	b1	b0
IN_USE	-	PATTERN_WIDTH					

The pattern_feature allows to „replicate” the the source data within a horizontal line. If the IN_USE bit is cleared, then this function is switched off and the source data is never replicated.

If the IN_USE bit is 1, then, when PATTERN_WIDTH+1 (1 ... 64) bytes have been copied, the source address value is restored to its initial state for the line, and, next to this, the PATTERN_WIDTH+1 bytes will be copied again, and the source address will be restored again etc. until blt_width+1 bytes are copied. The pattern copying will get aborted, if (blt_width+1)%(PATTERN_WIDTH+1) != 0, or in other words, blt_width has a higher priority.

blt_control

The byte controlling the operation and general behaviour of the blitter:

b7	b6	b5	b4	b3	b2	b1	b0
DY	DX	SY	SX	NEXT	MODE		

MODE	Description

0	The so called "COPY MODE". Every byte of the source data is copied to the destination, without any regard to transparency (values of 0) and without collision detection.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source'' = source' ^ blt_xor_mask;
	dest' = source'';
	WriteDest(dest');

1	The main blitter mode. The source” data is copied to the destination area, IF (source” != 0). If the blt_collision_mask is non-zero, then before the copying the blitter will fetch the destination byte and if this byte is not a zero, then collision will occur, and the dest code will be written to the BLT_COLLISION_CODE register. The collision detection slows down the blitter. If the collision detection is not desired, it is better to set the blt_collision_mask to 0, this disables the collision detection and the blitter will work faster.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source'' = source' ^ blt_xor_mask;
	if (source'' != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	  dest' = source'';
	  WriteDest(dest');
	}

MODE	Description

2	The written out data dest’ is an arithmetical sum of source” and the dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	  dest' = dest + source";
	  WriteDest(dest');
	}

3	The written out data dest’ is a result of a bitwise OR of source” and dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	  dest' = dest | source";
	  WriteDest(dest');
	}

4	The written out data dest’ is a result of a bitwise AND of source” and dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	dest = ReadDest();
	if (source" != 0 && (dest & blt_collision_mask))
	{
	  BLT_COLLISION_CODE = dest;
	}
	dest' = dest & source";
	WriteDest(dest');

5	The written out data dest’ is a result of a bitwise XOR of source” and dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	  dest' = dest ^ source";
	  WriteDest(dest');
	}

MODE	Description

6	HR Overlay support. It is basically the mode 1, except that transparency analysis and collision detection is done by nibbles rather than by bytes.
	
	`source = ReadSource();`
	`source' = source & blt_and_mask;`
	`source" = source' ^ blt_xor_mask;`
	`if (source" != 0)`
	`{`
	`  dest = ReadDest();`
	
	`  if (source"[3:0] != 0)`
	`  {`
	`    if (dest[3:0] & blt_collision_mask[3:0]) BLT_COLLISION_CODE[3:0] = dest[3:0];`
	`    dest'[3:0] = source"[3:0];`
	`  }`
	` else dest'[3:0] = dest[3:0];`
	
	`  if (source"[7:4] != 0)`
	`  {`
	`    if (dest[7:4] & blt_collision_mask[3:0]) BLT_COLLISION_CODE[7:4] = dest[7:4];`
	`    dest'[7:4] = source"[7:4];`
	`  }`
	` else dest'[7:4] = dest[7:4];`
	
	`  WriteDest(dest');`
	`}`

7	unused, reserved.

NEXT – if this bit is cleared (0), then te current BCB is the last BCB in the BlitterList, and after its execution the Blitter will end processing, clear the BUSY flag in the BLITTER_BUSY register, and triggering an IRQ, if it was allowed in the IRQ_CONTROL register. If the NEXT bit is set (1), then after finishing with the current BCB, the Blitter will behave as follows:

- clear the BUSY flag in the BLITTER_BUSY register
- at the same time it will set the BCB_LOAD flag in the same register
- fetch the next BCB
- set the BUSY flag in the BLITTER_BUSY register
- perform the next BCB
- after that, clear the BUSY flag
- check the NEXT bit
- etc. (the end or a next BCB)

Bits SX, SY, DX, DY.

At the outset it should be clarified, that before the Blitter starts, it makes the following operation:

```
source_adr' = source_adr;
dest_adr' = dest_adr;
```

Data is fetched from and written to the addresses marked as ‘ (prim).

SX – method of the source address modification within a horizontal data line:

```
SX = 0: source_adr' = source_adr' + 1
```

SX = 1: source_adr' = source_adr' - 1

SY – method of source address modification after the horizontal line is finished:

SY = 0 : source_adr = source_adr + source_step
SY = 1 : source_adr = source_adr - source_step

And:

source_adr' = source_adr;

DX - method of the destination address modification within a horizontal data line:

DX = 0 : dest_adr' = dest_adr' + 1
DX = 1 : dest_adr' = dest_adr' - 1

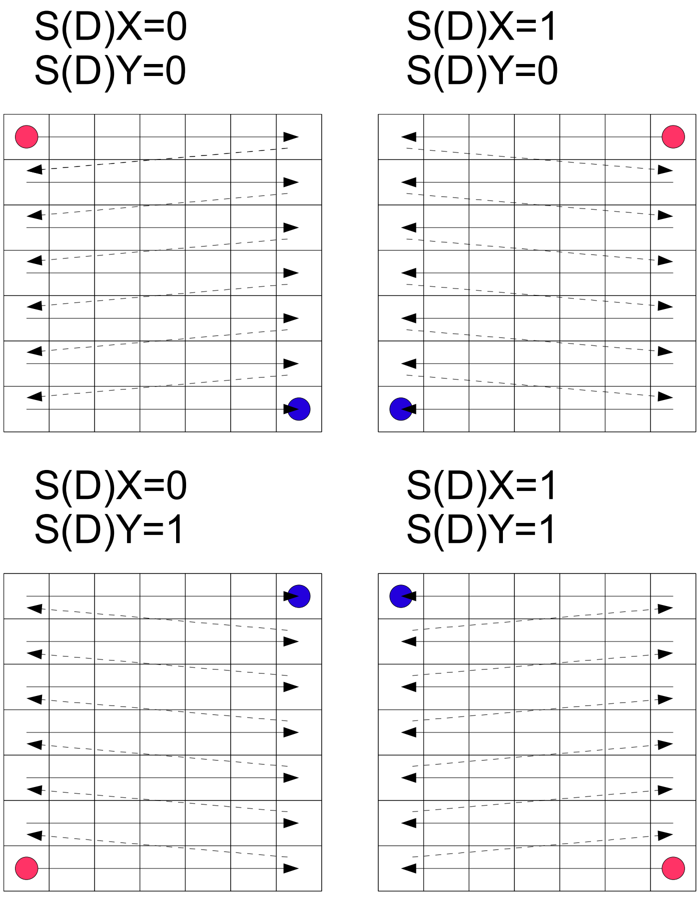
DY - method of destination address modification after the horizontal line is finished:

DY = 0 : dest_adr = dest_adr + dest_step
DY = 1 : dest_adr = dest_adr - dest_step

And:

dest_adr' = dest_adr;

The following figure shows the effects of the source and destination address modification during the Blitter’s work:



● START (= source_adr lub dest_adr w BCB)
● STOP - ostatni bajt danych
blt_width = 6 (7 punktów)
blt_height = 6 (7 linii)
1 kratka = 1 bajt
Adresy kolejnych wierszy oddalone od siebie
o wartość source_step lub dest_step

Changing the bits SX, SY, DX and DY you can invert the object vertically and horizontally, change the copying direction to avoid overlapping etc. The source_adr and dest_adr values in the BCB should always be adjusted accordingly.

The Blitter and constant source data

If the result of the following equation:

(~blt_and_mask ^ blt_xor_mask)

is $FF, then the source data is CONSTANT – it is independent from the source area and its value is equal to blt_xor_mask. The Blitter will skip the phase of fetching the source data, and the entire operation will be performed quicker. Filling VRAM with a constant value is twice as fast as copying.

CORE REGISTERS

Address	Write	Read
Dx40	VIDEO_CONTROL	CORE_VERSION (= $10)
Dx41	XDL_ADR0 bity 0 ... 7	255
Dx42	XDL_ADR1 bity 8 ... 15	255
Dx43	XDL_ADR2 bity 16 ... 18	255
Dx44	MSEL	255
Dx45	MB0	255
Dx46	MB1	255
Dx47	MB2	255
Dx48	MB3	255
Dx49	COLMASK	255
Dx4A	COLCLR	COLDETECT
Dx4B	-	255
Dx4C	MA_CPU	255
Dx4D	MA_ANTIC	255
Dx4E	MB_CPU	255
Dx4F	MB_ANTIC	255
Dx50	BL_ADR0 bits 0 ... 7	BLT_COLLISION_CODE
Dx51	BL_ADR1 bits 8 ... 15	255
Dx52	BL_ADR2 bits 16 ... 18	255
Dx53	BLITTER_START	BLITTER_BUSY
Dx54	IRQ_CONTROL	IRQ_STATUS
Dx55	-	255
Dx56	-	255
Dx57	-	255
Dx58	-	255
Dx59	-	255
Dx5A	-	255
Dx5B	-	255
Dx5C	-	255
Dx5D	-	255
Dx5E	-	255
Dx5F	-	255

x = 6 or 7, depending on where the VBXE is decoded in the Atari memory.

VIDEO_CONTROL

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	trans15	no_trans	xcolor	xdl_enabled
-	-	-	-	w-0	w-0	w-0	w-0

Symbols:

- first line: bit number b0 - b7
- second line: bit function ('-' = unused)
- third line:
    'w' – write only
    'r' – read only
    'rw' – read/write
    "-0" "0" after RESET
    "-1" "1" after RESET
    "-x" undefined after RESET

xcolor:

1 = display the PM0, PM1, PM2, PM3, PF0, PF1, PF2, PF3, BKGND colours taking the bit 0 into account (this makes 16 instead of 8 shades), and in the ANTIC hires (GR.0 and GR.8) display independent colours for the foreground and for the background.

0 = full GTIA compatibility: 8 shades in colour registers (128 colours) and the foreground colour dependent on the background colour in hires modes.

NOTE: the xcolor bit operates so either for global GTIA registers and for colours modified locally by the colour map fields.

xdl_enable:

1 = enable the XDL processing after the nearest VBL pulse.

0 = disable the XDL processing after the nearest VBL pulse.

no_trans:

0 = in the Overlay modes the colour index 0 will be treated as transparent and the ANTIC/GTIA display will be visible in its place.

1 = the Overlay has no transparent colours.

This bit allows to use all 256 palette indices as colours without problems.

NOTE: the no_trans bit has no influence on the Blitter, which in most of its modes will consider colour index 0 as transparent.

trans15:

This bit is only taken into account, when no_trans = 0. This allows to define additional transparent colours: see "Transparent Overlay colours".

XDL_ADR0

b7	b6	b5	b4	b3	b2	b1	b0
xdl_adr[7]	xdl_adr[6]	xdl_adr[5]	xdl_adr[4]	xdl_adr[3]	xdl_adr[2]	xdl_adr[1]	xdl_adr[0]
w-x	w-x	w-x	w-x	w-x	w-x	w-x	w-x

bits 0 ... 7 of the XDL address in the VBXE VRAM.

The XDL address should be set before enabling the XDL processing (xdl_enable in the VIDEO_CONTROL register).

XDL_ADR1

b7	b6	b5	b4	b3	b2	b1	b0
xdl_adr[15]	xdl_adr[14]	xdl_adr[13]	xdl_adr[12]	xdl_adr[11]	xdl_adr[10]	xdl_adr[9]	xdl_adr[8]
w-x	w-x	w-x	w-x	w-x	w-x	w-x	w-x

bits 8 ... 15 of the XDL address in the VBXE VRAM.

XDL_ADR2

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	xdl_adr[18]	xdl_adr[17]	xdl_adr[16]
-	-	-	-	-	w-x	w-x	w-x

bits 16 ... 18 of the XDL address in the VBXE VRAM.

MSEL

b7	b6	b5	b4	b3	b2	b1	b0
SEL1	SEL0	The exact function depends on SEL0 and SEL1					
w-0	w-0	w-x	w-x	w-x	w-x	w-x	w-x

SEL1	SEL0	działanie
0	0	do nothing
0	1	reserved / forbidden
1	0	execute MSEL / PRIORMAP
1	1	execute MSEL / RGB

See the description of MSEL/RGB and MSEL/PRIORMAP.

MB0, MB1, MB2, MB3

See the description of MSEL/RGB and MSEL/PRIORMAP.

MA_CPU

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	bank number: 0 ... 63					
w-0	-	w-x	w-x	w-x	w-x	w-x	w-x

MEMAC. The number of the MEMAC bank of VRAM in the window of $2000-$3FFF available to the CPU, when ENA = 1. When ENA = 0, then the CPU accesses the internal Atari memory in this area.

MA_ANTIC

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	bank number: 0 ... 63					
w-0	-	w-x	w-x	w-x	w-x	w-x	w-x

MEMAC. The number of the MEMAC bank of VRAM in the window of $2000-$3FFF available to the ANTIC, when ENA = 1. When ENA = 0, then the ANTIC accesses the internal Atari memory in this area.

MB_CPU

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	-	bank number: 0 ... 31				
w-0	-	-	w-x	w-x	w-x	w-x	w-x

MEMAC. The number of the MEMAC bank of VRAM in the window of $4000-$7FFF available to the CPU, when ENA = 1. When ENA = 0, then the CPU accesses the internal Atari memory in this area.

MB_ANTIC

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	-	bank number: 0 ... 31				
w-0	-	-	w-x	w-x	w-x	w-x	w-x

MEMAC. The number of the MEMAC bank of VRAM in the window of $4000-$7FFF available to the ANTIC, when ENA = 1. When ENA = 0, then the ANTIC accesses the internal Atari memory in this area.

BL_ADR0

b7	b6	b5	b4	b3	b2	b1	b0
blt_adr[7]	blt_adr[6]	blt_adr[5]	blt_adr[4]	blt_adr[3]	blt_adr[2]	blt_adr[1]	blt_adr[0]
w-x	w-x	w-x	w-x	w-x	w-x	w-x	w-x

bits 0 ... 7 of the BlitterList address in the VBXE VRAM.

The BlitterList address in the VRAM consists of 19 bits. The BlitterList can be located anywhere in the VRAM and start at any byte. There are no limits to the length of the BlitterList. When the address of the BlitterList has been written to the BL_ADR, the Blitter may be started. After it has finished, the contents of the BL_ADR remains unchanged.

The BL_ADR has to be loaded before starting the Blitter.

BL_ADR1

b7	b6	b5	b4	b3	b2	b1	b0
blt_adr[15]	blt_adr[14]	blt_adr[13]	blt_adr[12]	blt_adr[11]	blt_adr[10]	blt_adr[9]	blt_adr[8]
w-x	w-x	w-x	w-x	w-x	w-x	w-x	w-x

bits 8 ... 15 of the BlitterList address in the VBXE VRAM.

BL_ADR2

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	blt_adr[18]	blt_adr[17]	blt_adr[16]
-	-	-	-	-	w-x	w-x	w-x

bits 16 ... 19 of the BlitterList address in the VBXE VRAM.

BLITTER_START

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	-	1=START 0=STOP
-	-	-	-	-	-	-	w-0

Setting b0 bit to 1 causes the Blitter to start. It will read the BlitterList, then it will perform according to the instructions found in the BlitterList.

While the Blitter is working, it is possible to write 0 to the b0 bit. It will cause the Blitter to be stopped immediately. This mechanism allows to abort the Blitter, if it is looping infinitely (this can happen, when the Blitter has been started, and the VRAM is filled with a value of $FF – the BlitterList will then always contain the NEXT-marker). This function should not be used normally.

IRQ_CONTROL

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	-	IRQE
-	-	-	-	-	-	-	w-0

Enable the IRQ triggered after the Blitter has finished its work (i.e. after the transition from the BUSY state to IDLE state).
IRQE = 0 – Blitter IRQ disabled.
IRQE = 1 – Blitter IRQ allowed.

Writing any value of IRQE acknowledges and disables the Blitter IRQ, when it has been triggered.

CORE_VERSION

The version of the core, in BCD. $14 = version 1, revision 4.

BLT_COLLISION_CODE

The code of the collision detected, when the Blitter was running. A collision was detected, when the BLT_COLLISION_CODE != 0. The code corresponds to the non-zero value of the pixel overwritten by the Blitter.

BLITTER_BUSY

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	BUSY	BCB_LOAD
-	-	-	-	-	-	r-0	r-0

This register contains a non-zero value, while the Blitter is running, i.e. is processing the BlitterList (BCB_LOAD = 1) or it performs the actual operation (BUSY = 1). In IDLE state this register contains a value of 0 and the Blitter may be prepared for another task.

IRQ_STATUS

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	0	0	0	IRQF
-	-	-	-	-	-	-	r-0

IRQF = 0 – no IRQ has been requested by the VBXE.
IRQF = 1 – the Blitter-Done IRQ has been requested. Acknowledge by a write to IRQ_CONTROL.

COLMASK

The AND mask that allows to detect collisions between the Overlay and the Playfield / P/MG of the ANTIC/GTIA. The collision may be detected, if:

(Overlay_colour & COLMASK) != 0.

(& is a bitwise AND)

The Overlay colour corresponds to the Overlay colour index (0 ... 255) currently being displayed, without taking the OV_COLOR_SHIFT into account, and independently of the selected palette. This way it is possible to limit the collision detection to a part of the objects / colours of the Overlay.

COLCLR

Writing any value here will clear the COLDETECT register.

COLDETECT

The latch register of detected collisions. As the display is being generated, the collisions are detected in the process. A bit set to 1 means that a collision has been detected. For a collision to be detected, the condition must be met first, as specified in the description of the COLMASK register. The priority of the Overlay to the ANTIC/GTIA display does not have influence on the collision detection. Clearing bits (all at once) is accomplished by writing any value to the COLCLR register.

bit	Meaning (when set to 1)
b0	OVERLAY collides with COLPM0
b1	OVERLAY collides with COLPM1
b2	OVERLAY collides with COLPM2
b3	OVERLAY collides with COLPM3
b4	OVERLAY collides with COLPF0
b5	OVERLAY collides with COLPF1
b6	OVERLAY collides with COLPF2
b7	OVERLAY collides with COLPF3