VideoBoard XE FX Core

version 1.26

Programmer's Manual

Table of contents

Introduction....4
VGA Cores....5
GTIA emulation core....6
Cores for Atari 5200....6
Detection of core version....7
FX CORE....8
THE XDL....8
The order of fetching data in the XDL....14
OVERLAY MODES....15
Pixel modes....15
The text mode....16
Text mode scroll....17
Transparent Overlay colours....18
Priorities of displayed data of OVERLAY vs PLAYFIELD/PMG....19
OVERLAY-PLAYFIELD/PMG collision detection. (raster detection)....20
THE COLOUR ATTRIBUTE MAP....22
Attribute Map in ANTIC CCR mode....24
Attribute Map in ANTIC HIRES mode....24
RGB PALETTE MODIFICATION....26
MEMAC....27
MEMAC-A....28
MEMAC-B....29
BLITTER....30
The Blitter and constant source data....37
CORE REGISTERS....38
HISTORY....47

Introduction

This document is a programmer's guide for FPGA cores written for VBXE 1.x/2.x extension boards.

Avaivable cores:

Nazwa rdzenia	VBXE 1.x	VBXE 2.x
FX v1.26	v1fx126a.xbf	v2fx126a.xbf
FX v1.26 5200	v1fx126g.xbf	v2fx126g.xbf
FX v1.26 RAMBO	v1fx126r.xbf	v2fx126r.xbf
GTIA v1.06	v1g106a.xbf	v2g106a.xbf
GTIA v1.06 5200	v1g106g.xbf	v2g106g.xbf
GTIA v1.06 RAMBO	v1g106r.xbf	v2g106r.xbf

Cores with a name ending in "r" contain emulation for RAMBO 256K memory expansions and are intended for use in XL/XE machines without any other standard (PORTB based) memory expansions built in. In computers equipped with some memory expansions (RAMBO, COMPY SHOP, ABBUC), users should only use cores with names ending in "a". Cores for Atari 5200 console (file mane ending in 'g') are only available without RAMBO memory expansion.

Cores named "GTIA emu" contain only pure emulation of GTIA chip without any extended "FX" capabilities, with the only exception of newly added palette changing capabilities. A short description of this core family can be found in the section "GTIA emulation core".

To recognise the type of core that is actually running on the FPGA please refer to the section "Detection of core version".

The differences between core versions for XL/XE family and Atari 5200 are described in the section "Cores for Atari 5200 console".

VGA Cores

VGA cores are currently deprecated, and not actively developed. Limitations they had were too severe to justify their existence.

GTIA emulation core

The "GTIA emu" core provides only emulation for GTIA chip and RGB output. Only additional registers present in this core are CORE_VERSION and MINOR_VERSION used for core detection. Starting from version 1.06 user is also able to change RGB palette by altering CSEL, CR, CG and CB registers. Please refer to core register map.

Starting from version 1.06 GTIA core emulates so called “PAL blending” used is some programs to extend color capabilities, this feature howewer limits vertical colour resolution as in the case of real PAL TV. In opposition, the FX core provides “clean” output without such effect.

CORE_VERSION for GTIA emu core has value of 0x11.

MINOR_REVISION for GTIA emu v1.06a has value of 0x06, and for GTIA emu v1.06r 0x86.

Cores for Atari 5200

For the Atari 5200 console the only cores available are "FX" and "GTIA emu" without RAMBO 256k emulation. Differences from the XL/XE series are:

- GTIA is mapped at 0xC000-0xCFFF,
- (only FX core): there is no MEMAC-B register,
- (only FX core): MEMAC-A window is fixed at 0xD800 address and its size is fixed to 4KB (0xD800-0xE7FF), and it can't be altered regardless of MEMAC_CONTROL bit settings (bits are ignored).

Detection of core version

This detection scheme is valid starting from FX v1.21 and GTIA emu v1.01 and will stay valid for any futher core versions and core types.

The type of the core depends on value of CORE_VERSION:

When CORE_VERSION = 0x10 - FX v1.xx is running
When CORE_VERSION = 0x11 - GTIA emu v1.xx is running
Other values are RESERVED for futher core types. The value if VBXE is not present is undefined.

code example:

```
        lda     CORE_VERSION
        cmp     #$10
        beq     core_fx
        cmp     #$11
        beq     core_gtia_emu
; no VBXE or unknown core !
        jmp     error
```

Next thing to do is to read MINOR_REVISION. Bit 7 of this register determines whether its a "a" core (bit cleared), or "r" core (bit set).
In most cases this information can be discarded, thus 0x7F AND mask should be applied.
Bits 6-0 contain BCD coded revision of the core (ie: 0x26 = core v1.26, 0x06 = core v1.06). Compatible cores have bits 6-4 equal, so software for core FX core v1.20 can should run at core v1.24.

code example:

```
core_fx:
        lda     MINOR_REVISION
        and     #$7F
        sta     exact_core_revision  ; #$26 = FX v1.26 a/r
        and     #$70
        cmp     #$20  ;#$20-#$2F - compatible cores (bugfixes only)
        beq     fx_compatible_revision

fx_incompatible_revision:
        jmp     error

core_gtia_emu:
        lda     MINOR_REVISION
        and     #$7F
        sta     exact_core_revision  ; #$06 = GTIA emu v1.06 a/r
        jmp     end
```

FX CORE

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.		

<table>
<tr><th>byte.bit
XDLC</th><th>bit's label</th><th>meaning</th><th>additional data</th></tr>
<tr><td>2.3</td><td>XDLC_ATT</td><td>Setting the display size (both Overlay and Colour map) and Overlay priority to the ANTIC display. And Overlay colour modification.</td><td>2 bytes:
1. Overlay / map width + palette change
2. main priority</td></tr>
<tr><td>remarks:</td><td colspan="3">BYTE 1:

<table>
<tr><td>b7</td><td>b6</td><td>b5</td><td>b4</td><td>b3</td><td>b2</td><td>b1</td><td>b0</td></tr>
<tr><td colspan="2">XDL PF PALETTE</td><td colspan="2">XDL OV PALETTE</td><td>-</td><td>-</td><td colspan="2">OV_WIDTH</td></tr>
</table>

OV_WIDTH: OVERLAY and ATTRIBUTES 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).

XDL OV PALETTE: the palette for OVERLAY active from this screen line. By default, OVERLAY uses palette nr.1 from the top of the screen.

XDL PF PALETTE: the palette for PLAYFIELD and PMG active from this screen line. By default it's the nr.0 palette from the top of the screen.

CAUTION: if the attribute map is turned on, the palettes for OVERLAY and PLAYFIELD/PMG are selected by the corresponding attribute maps.

BYTE 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 and PF3
b7 - 1 = OVERLAY over COLBAK

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.</td></tr>
<tr><td>2.4</td><td>XDLC_HR</td><td>enable the Hi-Res pixel mode</td><td>-</td></tr>
</table>

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.		

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
```

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.

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 of displayed data of OVERLAY vs PLAYFIELD/PMG.

There are several priority registers defining the order data of PLAYFIELD/PMG and OVERLAY content are drawn:

- main priority register (set by XDL) - by default set to 255 which means that the OVERLAY data has higher priority than any of PF/PMG colours.

- P0-P3 registers - used INSTEAD main priority register set by XDL when the Attribute Map is enabled. Their meaning is the same as for the main priority register. The content of the 4th byte of AM cell decides which (P0, P1, P2, or P3) register is active for given cell. See the register description for the further details.

The meaning of the bits in the main priority register (and in P0-P3 registers) is as follow:

b0 - 1 = OVERLAY over PM0, 0 = OVERLAY covered by PM0
b1 - 1 = OVERLAY over PM1, 0 = OVERLAY covered by PM1
b2 - 1 = OVERLAY over PM2, 0 = OVERLAY covered by PM2
b3 - 1 = OVERLAY over PM3, 0 = OVERLAY covered by PM3
b4 - 1 = OVERLAY over PF0, 0 = OVERLAY covered by PF0
b5 - 1 = OVERLAY over PF1, 0 = OVERLAY covered by PF1
b6 - 1 = OVERLAY over PF2 and PF3, 0 = OVERLAY covered by PF2 and PF3
b7 - 1 = OVERLAY over COLBAK, 0 = OVERLAY covered by COLBAK

In GTIA modes (9,11) (16 shades / 16 colours) the ANTIC / GTIA generated pixels are all treated as COLBAK colour, so, if not using the PMG, only b7 priority bit controls if OVERLAY is visible or not.

OVERLAY-PLAYFIELD/PMG collision detection. (raster detection)

It's possible to detect a collision between the displayed OVERLAY data (both text mode and graphics mode) and any of the COLPM0,1,2,3,COLPF0,1,2,3 and attribute map field with CATT bit set.

Collision detection is automatic, and takes place while VBXE draws the display.

The configuration of the raster collision detector 'sensitivity' is done by writing to the COLMASK register. The following table describes the COLMASK bits and their meanings.

COLMASK bit	If set, the detected collision applies to OVERLAY-PLAYFIELD/PMG collision, if the OVERLAY colour is in the following range: (the palette number is insignificant)
0	0-31
1	32-63
2	64-95
3	96-127
4	128-159
5	160-191
6	192-223
7	224-255

Any combination of COLMASK bits is allowed.

The collision code can be read from the COLDETECT register:

COLDETECT bit	If set, the collision is detected between OVERLAY objects and:
0	PM0 colour
1	PM1 colour
2	PM2 colour
3	PM3 colour
4	PF0 colour
5	PF1 colour
6	Colours PF2 or PF3
7	Attribute map field with CATT bit set

Any combination of COLDETECT bits is possible – software collision checking with COLDETECT register should be done with the appropriate AND mask.

Invalidating the detected collision (zeroing out the COLDETECT register) is done by

writing any value to the COLCLR register.

CAUTION: one can only detect the collision with the 'non-transparent' OVERLAY colors.
CAUTION: display priorities do not affect the collision detection process.
CAUTION: one should not confuse the above mentioned mechanism with the OVERLAY-OVERLAY collision detection which is done by the blitter (see blt_collision_mask)

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 in CCR modes, pattern fill in Antic Hi-Res mode
    - 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.
        - support for raster collision detection of the Attribute map with OVERLAY objects (bit CATT)
        -

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
MAP PF palette		**MAP OV palette**		**CATT**	**RES**	**PS**	

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.

WARNING: The Colour Palette assigned here has priority over the one selected in XDL (XDL PF/OV PALETTE), that means, if the Attribute Map is active, then on the area covered by pixels are coloured with palettes assigned to the Attribute Map cells versus default palette assigned by the XDL.

b3 (CATT) - collision detection auxiliary bit. If set, then there is possible to detect collisions with the Attribute Map cells during the scanline, regardless of the data displayed by ANTIC/GTIA chip. See also: Raster collision detection.

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 (PS) – selection of one of the 4 predefined priorities OVERLAY <-> ANTIC/GTIA.

PSEL1	PSEL0	register selected
0	0	Priority register P0
0	1	Priority register P1
1	0	Priority register P2
1	1	Priority register P3

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.

Attribute Map in ANTIC CCR mode

In ANTIC CCR mode (native, or forced with the RES bit - see below) if Attribute Map (AM) is enabled, standard GTIA colour registers (COLPF0, COLPF1 and COLPF2) are replaced by values from the PF0, PF1 and PF2 fields of AM cell. COLPF3 has no matching field in the AM cell and is the same as set in standard register.

Attribute Map in ANTIC HIRES mode

In ANTIC HIRES mode (native, or forced with the RES bit) first byte of AM cell definition has alternative meaning - each bit of this byte select the background colour in each pixel of the cell:

bit = 0 then background colour is assigned as in PF2 field (3rd byte of the AM cell)
bit = 1 then background colour is assigned using COLPF3 (from the GTIA register set)

It can be thought of as one bit depth "overlay". This enables HIRES mode to use 3 colours instead of two.

For cells of width starting from 9 to 16 pixels each bit of this "overlay" sets the colour for 2 adjacent pixels, and if the cell width >16 pixels, each bit sets the colour for 4 adjacent pixels.

Second byte selects the colour for lit pixel. You need to remember, that if the XCOLOR bit in the VIDEO_CONTROL register is cleared, then 4 higher bits of the colour value are replaced with these from COLBGK register. In the other case pixel colour is fully independent from the background colour, and colour registers have 8 bits each (this equals to full 16 shades instead of 8 shades of one of standard colours).

RES bit description

RES bit decides if the data send currently by ANTIC will be treated as CCR mode or HIRES mode.

Normally ANTIC decides (at the beginning of the scanline) how GTIA (VBXE) should interpret following data:

- as ANTIC HIRES mode (modes 2, 3 and F from ANTIC's DL), or
- as ANTIC CCR mode (remaining text and graphics modes)

The RES bit, if set, allows to force local change (reverse) of interpretation of this data by VBXE - part of the image can be displayed in altered resolution.

Conversion of CCR to artificial ANTIC HIRES mode:

Each pixel in CCR resolution has width of 2 pixels of ANTIC HIRES mode, and is converted into two pixels according to the rules in the table below:

colour of the pixel in CCR mode	**HIRES, left pixel**	**HIRES, right pixel**
BACKGROUND	HIRES background	HIRES background
COLPF0	HIRES background	PF1 colour (2nd byte of AM cell)
COLPF1	PF1 colour (2nd byte of AM cell)	HIRES background
COLPF2	PF1 colour (2nd byte of AM cell)	PF1 colour (2nd byte of AM cell)
COLPF3	PF1 colour (2nd byte of AM cell)	PF1 colour (2nd byte of AM cell)

Where "HIRES background" is the color generated by the rules described previously in "Attribute Map in ANTIC HIRES mode". Depending on value of 1st byte of the AM cell it can be colour PF2 (from 3rd byte of the cell), or COLPF3.

Conversion of ANTIC HIRES mode into artificial CCR mode:

Pairs of adjacent bits are converted to pixels according to the rules in the table below:

HIRES, left pixel	**HIRES, right pixel**	**Resulting color**
0	0	PF0 (1st byte of AM cell)
0	1	PF1 (2nd byte of AM cell)
1	0	PF2 (3rd byte of AM cell)
1	1	COLPF3

RGB PALETTE MODIFICATION

VBXE allows for simultaneous display of 1024 colours (out of 21bit hardware palette of 2097152 colours, the least significant bit of the component colour registers is discarded) There are 4 sets (palettes) with 256 user-predefined colours each.

The RGB palette for each of those 1024 (4 x 256) colors is updated as follows:

- write palette number (0-3) into PSEL register,
- write palette colour number (0-255) into CSEL register
- write R component colour value into CR register
- write G component colour value into CG register
- write B component colour value into CB register

CAUTION: Write to CB (but not to CR nor CG) will automatically increase CSEL. One can write next color value without direct CSEL modification.

CAUTION: If CSEL overflows from 255 to 0, PSEL will be unaffected. So, if you want to modify next palette, you will have to write the proper value into PSEL. Otherwise, you will be modificating the same palette, only starting with 0 colour index.

This behaviour allows for palette-rotation based effects with minimal CPU load. Setting up the palette itself is also faster than in the previous core.

This change, together with the change in the palette switching, allows for easier and visually proper implementation of the FADE type effects (smooth fading out or fading in of images) Implementing these kind of effects only with a part of the palette allows a programmer to display images in a more interesting way: e.g credit subtitles with a static image on the side of the screen.

Writes to each of the CR, CB and CG registers make an instant change on the screen, in a form of changing the colour. (if the given palette and colour are currently being displayed) – the component colour values are not buffered.

In GTIA emu cores user can change palette in the same way. There is no PSEL register in these cores howewer, GTIA cores use only palette 0.

By default, after power-on, the palette no #0 contains colours for PLAYFIELD/PMG which are modified colors from „laoo.act" palette. Palettes 1-3 have all components zeroed out. (all colors black)

Please remember, that after modifying no #0 palette, the programmer is responsible for resetting it back to default values when the program exits. Also, remember that the palette registers are read-only

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 (14.18/14.31 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. Because of the "movable" window introduced in FX v1.20 the core is checking CASINH signal of the MMU (it has to be connected to VBXE - see the installation guide, or history version at the end of this document) so the conflicts with OS, BASIC or the CARTRIDGE ROM memory could be excluded.

The VBXE memory access can be accomplished through two independent „windows”:

MEMAC A - window of definable base address and size.
MEMAC B - fixed size window at $4000 - $7FFF (a 16 KB window)

WARNING: If MEMAC A window is defined so it overlaps the MEMAC B windows, then MEMAC A has priority over the MEMAC B. There is no conflict with the priorities if MEMAC A or B is defined as ANTIC accessible, and the other one 6502 accessible - in this case windows can overlap partially or in whole.

WARNING: In case of window in $4000-$7FFF address space programmer should disable extended memory writing 0xFF or 0xFE to the PORTB ($D301) if want to access the VBXE memory - some memory extensions that utilises EXTSEL signal have priority above VBXE. As a result, data that should go into VRAM would go into EXTRAM.

Warning: Cores with 'r' suffix (ie "fx v1.26r"):

Special function of MEMAC is emulation of standard 320KB expansion (RAMBO - 256kB of extended ram with no separate access for ANTIC and CPU). The extension memory is mapped from upper half of VBXE memory. One have to remember that this memory is shared with VRAM, and can be altered by VBXE oriented application!

FX cores with 'a' suffix (ie "fx v1.20a") have no extended memory expansion emulation, and should be used if standard memory expansion unit is already installed in the computer.

MEMAC-A

MEMAC-A window can be placed in arbitrary base address of 6502 with granitulaty of 0x1000 bytes, so it can be places on 0xY000 address where Y=0..F. Size of this window is configurable from one of following sizes:
$1000 (4kB)
$2000 (8kB)
$4000 (16kB)
$8000 (32kB)

WARNING: window set at starting address $F000 (for example) and of size exceeding $1000 will be truncated to $1000 regardless of it size - there will be no address wrap back to $0000 address.

WARNING: if in the selected address space ROM memory or the hardware registers exists part of VRAM will be not accessible, because ROM and hardware registers have priority above VRAM.

MEMAC A window can be:
- off (standard RAM will be there)
- on for CPU (CPU "sees" VRAM, ANTIC "sees" Atari RAM)
- on for ANTIC
- on for both ANTIC and the CPU

ANTIC and the CPU sees the same fragment of VRAM (they have the same bank register - MEMS)

MEMAC-A is controlled by 2 registers (see also description near the end of this document):

MEMMAC_CONTROL (MEMC) - this register selects base address and window size. After the RESET 0 is written to the MCE and MAE bits so the window is disabled, remaining bits are left untouched.

MEMAC_BANK_SEL (MEMS) - this register decides of which part of the VRAM is available through the window. Additionally there is MGE bit which globally enables the window. This bit is also reset when the RESET button is pressed.

For further details on these registers please refer to REGISTER DESCRIPTION.

MEMAC-B

MEMAC-B window is located at 0x4000 address and its size is fixed to 0x4000 (16kB).

MEMAC-B window can be:

- off (standard RAM will be there)
- on for CPU (CPU "sees" VRAM, ANTIC "sees" Atari RAM)
- on for ANTIC
- on for both ANTIC and the CPU

MEMAC-B window configuration and bank selection is done through MEMAC_B_CONTROL register (MEMB). This is write only register, and was left as a legacy from the previous version of the core.

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 21 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_step_y_0	bits 0 ... 7, source step y (signed)
5	source_step_y_1	bits 8 ... 11, source step y (signed)
6	source_step_x	bits 0 ... 7, source step (signed)
7	dest_adr0	bits 0 ... 7, destination address
8	dest_adr1	bits 8 ... 15, destination address
9	dest_adr2	bits 16 ... 18, destination address
10	dest_step_y_0	bits 0 ... 7, destination step y (signed)
11	dest_step_y_1	bits 8 ... 11, destination step y (signed)
12	dest_step_x	bits 0 ... 7, destination step x (signed)
13	blt_width0	bits 0 ... 7, object width (in bytes)
14	blt_width1	bit 8, object width (in bytes)
15	blt_height	bits 0 ... 7, object height (in lines)
16	blt_and_mask	AND-mask for source data
17	blt_xor_mask	XOR-mask for source data
18	blt_collision_mask	AND-mask for collision detection
19	blt_zoom	X- and Y- axis zoom of the object being copied
20	pattern_feature	pattern fill
21	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_y

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 y = -4096...4095

source_step_x

source step x = -128 ... 127

dest_adr

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

dest_step_y

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 = -4096...4095

dest_step_x

dest step x = -128...127

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 detection mask. Collision detection is available in blitter modes 1,2,3,4,5 and 6 (see blt_control)

Blitter modes 1,2,3,4 and 5:

Each collision mask bit corresponds to one segment of the palette. Each segment of the palette has 32 colors (0-31, 32-63,64-95 and so forth) Thus, by using palette in a proper way, we can allow for collision detection in different groups of objects on the screen.

Collision is detected if the following condition is true:
(source" != 0 && collision_sr()), where :

source" – source data (0-255) after AND and XOR operations

```
char collision_sr(void)
{
  return (((blt_collision_mask & 1) && dest >=1 && dest <= 31) ||
  ((blt_collision_mask & 2) && dest >=32 && dest <= 63) ||
  ((blt_collision_mask & 4) && dest >=64 && dest <= 95) ||
  ((blt_collision_mask & 8) && dest >=96 && dest <= 127) ||
  ((blt_collision_mask & 16) && dest >=128 && dest <= 159) ||
  ((blt_collision_mask & 32) && dest >=160 && dest <= 191) ||
  ((blt_collision_mask & 64) && dest >=192 && dest <= 223) ||
  ((blt_collision_mask & 128) && dest >=224 && dest <= 255)) ? 1 : 0;
}
```

"dest" is the retrieved target data (before blitter writes).

Example: giving colors 0-31 (bit 0) to background image, and colors 32-63 and 96-127 (bits 1 and 3) to static screen elements, allows for easy check for the collision of a given object with a static screen element or the lack of the collision thereof, thus freeing the cpu from the collision check. Using this technique we can easily avoid using ANTIC, and generate the whole screen layout with VBXE, freeing the bus for the CPU (no DMA cycles, the only bus activity are the REFRESH cycles)

Blitter mode 6 (VBXE in HR mode) :

Palette is only 16 colors long, so each mask bit controls 2 consecutive colors. Both nibbles are treated separately.

Collision is detected if the following condition is met:
(source" != 0 && collision_hr()), where:

source" – source data with value of 0 to 15 (more significant nibble or less significant nibble – treated separately) after AND and XOR operations .

```
char collision_hr(void)
{
  return (((blt_collision_mask & 1) && dest ==1) ||
  ((blt_collision_mask & 2) && dest >=2 && dest <= 3) ||
  ((blt_collision_mask & 4) && dest >=4 && dest <= 5) ||
  ((blt_collision_mask & 8) && dest >=6 && dest <= 7) ||
  ((blt_collision_mask & 16) && dest >=8 && dest <= 9) ||
  ((blt_collision_mask & 32) && dest >=10 && dest <= 11) ||
  ((blt_collision_mask & 64) && dest >=12 && dest <= 13) ||
  ((blt_collision_mask & 128) && dest >=14 && dest <= 15)) ? 1 : 0;
}
```

"dest" – retrieved target data (before blitter writes). More significant nibble or less significant nibble (treated separately)

CAUTION: any combination of blt_collision_mask is allowed.

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			-	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

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
-	-	-	-	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');
	}

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');
	}

MODE	Description

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)

The Blitter and constant source data

If the result of the following equation:

(blt_and_mask==0)

is true, 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 (FX CORE)

Address	Write	Read
Dx40	VIDEO_CONTROL	CORE_VERSION
Dx41	XDL_ADR0 bity 0 ... 7	MINOR_REVISION
Dx42	XDL_ADR1 bity 8 ... 15	255
Dx43	XDL_ADR2 bity 16 ... 18	255
Dx44	CSEL	255
Dx45	PSEL	255
Dx46	CR	255
Dx47	CG	255
Dx48	CB	255
Dx49	COLMASK	255
Dx4A	COLCLR	COLDETECT
Dx4B	-	255
Dx4C	-	255
Dx4D	-	255
Dx4E	-	255
Dx4F	-	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	P0	255
Dx56	P1	255
Dx57	P2	255
Dx58	P3	255
Dx59	-	255
Dx5A	-	255
Dx5B	-	255
Dx5C	-	255
Dx5D	MEMAC_B_CONTROL	255
Dx5E	MEMAC_CONTROL	MEMAC_CONTROL
Dx5F	MEMAC_BANK_SEL	MEMAC_BANK_SEL

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

CORE REGISTERS (GTIA CORE)

Address	Write	Read
Dx40	-	CORE_VERSION
Dx41	-	MINOR_REVISION
Dx42	-	255
Dx43	-	255
Dx44	CSEL	255
Dx45	-	255
Dx46	CR	255
Dx47	CG	255
Dx48	CB	255
Dx49	-	255
Dx4A	-	255
Dx4B	-	255
Dx4C	-	255
Dx4D	-	255
Dx4E	-	255
Dx4F	-	255
Dx50	-	255
Dx51	-	255
Dx52	-	255
Dx53	-	255
Dx54	-	255
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 (the color of lit pixel is then fetched from PF1 register).

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.

CSEL (Color SELect)

b7	b6	b5	b4	b3	b2	b1	b0
Starting color number for RGB components modification							
w-x	w-x	w-x	w-x	w-x	w-x	w-x	w-x

Before modifying RGB color components, one needs to write the color number (the one to modify, or the one from which to start modification) to CSEL register.
CAUTION: CSEL is automatically incremented after CB write.

PSEL (Palette SELect)

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	Palette number to modify	
-	-	-	-	-	-	w-x	w-x

Before modifying RGB color components, one need to write the palette number to PSEL register. PSEL, unlike CSEL, never increments automatically.

CR (Component Red)

b7	b6	b5	b4	b3	b2	b1	b0
7-bit R (red) component							-
w-x	w-x	w-x	w-x	w-x	w-x	w-x	-

R component value change is done instantly after CR write.

CG (Component Green)

b7	b6	b5	b4	b3	b2	b1	b0
7-bit G (green) component							-
w-x	w-x	w-x	w-x	w-x	w-x	w-x	-

G component value change is done instantly after CG write.

CB (Component Blue)

b7	b6	b5	b4	b3	b2	b1	b0
7-bit B(blue) component							-
w-x	w-x	w-x	w-x	w-x	w-x	w-x	-

B component value change is done instantly after CB write.
Moreover, CB write forces CSEL to increment. (if CSEL = 255, CSEL overflows to 0) PSEL is unaffected.

MEMAC_CONTROL (MEMC)

b7	b6	b5	b4	b3	b2	b1	b0
Base address of MEMAC				MCE	MAE	window size	
rw-x	rw-x	rw-x	rw-x	rw-0	rw-0	rw-x	rw-x

Base address of MEMAC window:

Ie. when 4 is written here, base address will be set to 0x4000, writing 7 here will result in setting base address to 0x7000 etc.

MCE - MEMAC CPU ENABLE - when set, MEMAC A window will be accessible by 6502. This bit is reset when RESET button is pressed.

MAE - MEMAC ANTIC ENABLE - when set, MEMAC A window will be accessible by ANTIC. This bit is also reset on RESET.

WARNING: window will stay disabled if the MGE bit (MEMAC GLOBAL ENABLE) in MEMAC_BANK_SEL register is reset.

Window size:

b1	b0	window size
0	0	4K
0	1	8K
1	0	16K
1	1	32K

MEMAC_BANK_SEL (MEMS)

b7	b6	b5	b4	b3	b2	b1	b0
MGE	bank number 0 ... 127 for 4K window						
MGE	bank number 0 ... 63 for 8K window						-
MGE	bank number 0 ... 31 for 16K window					-	-
MGE	bank number 0 ... 15 for 32K window				-	-	-
rw-0	rw-x	rw-x	rw-x	rw-x	rw-x	rw-x	rw-x

Bits b0 - b6: Selects VRAM bank number that will be mapped into Atari address space.

Bit b7 - MGE : MEMAC GLOBAL ENABLE - when set enables MAMAC A window as previously defined in MEMAC_CONTROL register. This global enable bit allows to use MEMAC_CONTROL register as configuration only, so the programmer can set it once at the beggining of the code, and use only MEMAC_BANK_SEL register during the run.

WARNING: Application should restore values of MEMAC window configuration and current bank state at exit.

MEMAC_B_CONTROL (MEMB)

b7	b6	b5	b4	b3	b2	b1	b0
MBCE	**MBAE**	**-**	VRAM bank number (0 ... 31)				
w-0	w-0	-	w-x	w-x	w-x	w-x	w-x

MBCE - MEMAC-B CPU ENABLE - when set 6502 can access VRAM.

MBAE - MEMAC-B ANTIC ENABLE - when set ANTIC can access VRAM.

Bank number - 32 banks, 16kB each gives access to whole 512k of VRAM.

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.

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_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.

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.

CORE_VERSION

Core type:

0x10 = FX v1.XY core
0x11 = GTIA emu v1.XY core

MINOR_REVISION

b7	b6	b5	b4	b3	b2	b1	b0
RAMBO	**X** digit (X=0...7)			**Y** digit (Y=0...9)			
r	r	r	r	r	r	r	r

RAMBO = 1 - core with RAMBO 256K emulation ("r" core).
RAMBO = 0 - core without RAMBO emulation ("a" core).

FX cores with only Y digit diffrent are compatible with each other (no function changes, just bugfixes).

FX cores with X digit diffrent are not compatible, and if program detects core version that is diffrent with one expected should terminate.

FX core versions:

0x26 = FX v1.26a
0xa6 = FX v1.26r

GTIA emu versions:

0x06 = GTIA emu v1.06a
0x86 = GTIA emu v1.06r

COLMASK

The AND mask that allows to detect collisions between the Overlay and the Playfield / P/MG of the ANTIC/GTIA. See the Detetection of raster collisions section.

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.
See the Detetection of raster collisions section.

P0, P1, P2, P3

OVERLAY - PLAYFIELD/PMG Priority select registers used when Attribute Map is enabled (on the area covered by the AM). See chapter "Priorities of displayed data of OVERLAY vs PLAYFIELD/PMG".

HISTORY

Version 1.26

- bugfix in blitter: ZOOM'ing while copying in X axis didn't work properly.
- Fixed emulation of BASIC GR.11 (GTIA MODE) – now luminance of %0000 pixel value is correct.
- Change in XCOLOR bit (VIDEO_CONTROL register of FX core) operation – now when SET, ANTIC HIRES pixels not only have bacground indenpedent colors, but also have priority of PF1 color as in any other ANTIC GFX mode (GTIA priority registers can be used as normal).
- Change in handling of priority of OVERLAY-GTIA. Now, if any of GTIA generated colors overlaps, and for any of them PRIORITY was set to override with OVERLAY, OVERLAY pixel will be displayed, compared to OVERLAY logically OR'ed with GTIA as it was in 1.24 core and older. Situation like this can have place if GTIA PRIOR register will have value 0, and some PF/PMG colors will overlap.
- Change in priority registers – main one and these loaded with XDL/color map. Now 6^{th} bit chooses priority of colors PF2 and PF3, and 7^{th} bit changes priority for COLBAK (ie to hide OVERLAY behind COLBAK).
- In GTIA-only cores PAL-BLENDING was added.
- In GTIA-only cores is now possible to change palette to user defined one – no more issues with personal preferences or dislikes to default palette (LAOO.ACT). Please reffer to FX section on how to change palette (Palette 0).

Wersja 1.25

Unofficial version, never released to GP.

Version 1.24

- minor internal changes
- bugfix in XDL width change from narrow to normal or wide and from normal to wide: error was introducing additional (erratic) fetch from XDLC from VRAM memory what resulted in vertical shift of following overlay lines
- added VGA output handling

Version 1.23

- changes of internal timings and EXTSEL handling - not important from programmer's point of view

Version 1.22

- Changes improoving stability of the hardware

Version 1.21

- bugfix in GTIA emulation - PRIOR register behaviour ($D01B)
- since now, cores with "r" suffix will have bit 7 set in MINOR_REVISION register

Version 1.20

- new MEMAC-A and MEMAC-B memory windows.

- **WARNING: since now CASINH is required to be connected to pin 9 (AUX4) of J3 connector on VBXE v1.x. For VBXE v2.0 CASINH signal should be connected to J3 connector pin 4. CASINH is available on pin #16 of MMU, or pin #4 of FREDDIE chip.**
- Changes from 1.09 core were translated by Candle'O'Sin (Sebastian Bartkowicz)
- Proofreading and grammar check by Dan Winslow

Version 1.10

- Attribute Map: fixed RES bit functionality
- Attribute Map: added 1 bit overlay to the Attribute Map cell
- Blitter: fixed zoom behaviour, extended BCB to 21 bytes.

Version 1.09

- removal of OV_COLOR_SHIFT
- XDL can now change palettes for OVERLAY and PLAYFIELD/PMG (the same bits are used that were previously used for OV_COLOR_SHIFT)
- new blitter collision detection model - see "blt_collision_mask"
- new raster collision detection model
- OVERLAY/attribute map field collision detection (see raster collision detection and attribute map – CATT bit)
- MSEL/RGB mechanism removed. - new color modification registers available (CSEL,PSEL,CR,CG,CB)
- MSEL/PRIOMAP mechanism removed - P0, P1, P2 and P3 registers have now dedicated IO addresses.
- XDLC_OVATT bit renamed to XDLC_ATT
- Changes from 1.07 core were translated by Mikey (Michał Szwaczko)

Wersja 1.08

- security update (highly recommended) in the MEMAC A/B area handler. Now, VBXE will honour the fact that the external expansion can force this area available for memory access (via EXTSEL signal, like the CAS INHIBIT signal from the computer) VBXE will not map its own memory in this area. One needs to remember to write $FF to $D301 before any MEMAC B usage. (or any other value that will shutdown a possible external or internal RAM expansion) Otherwise, the RAM expansion could get higher priority than the MEMAC accessible VRAM, via EXTSEL assertion.

- Writing of any value into any of the $D080 - $D0FF addresses (GTIA registers copy) will result in a software VBXE reset (identical with RESET keypress) resulting in termination of XDL processing (OVERLAY and attribute map become unavailable) and some bits of the control registers will be reset to the default values. CAUTION: no RESET is able to restore the default VBXE palette if it has been already modified by a program. Restoring of the palette is possible only by re-loading the core. This function allows for restoring of the original screen after cold/warm start of the system, by OS ROM jump, not by RESET keypress.
- New register MINOR_REVISION (read only) for easy identification of the loaded core.

Version 1.0 beta 7

- First English version of the programmer's manual. Translated by Drac030 (Konrad M. Kokoszkiewicz).