VideoBoard XE
1.1 / 1.2 / 2.0 / 2.1

rdzeń

FX v1.26

GTIA v1.06

Podręcznik programisty

31.08.2013

Spis treści

WSTĘP

Niniejszy dokument jest podręcznikiem programisty dla rdzeni FPGA ACEX EP1K50TC144-3 napisanych dla karty VBXE w wersji 1.x / 2.x. (gdzie 1.x = VBXE rev. 1.1 i VBXE rev. 1.2; 2.x = VBXE rev. 2.0 i VBXE rev. 2.1).

Dostępne pliki rdzeni:

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

Rdzenie z literą „r" na końcu nazwy pliku (np. „v1fx126r.xbf”) zawierają emulację rozszerzenia RAMBO 256K dla komputerów pozbawionych standardowego rozszerzenia RAM. W komputerach z wbudowanym rozszerzeniem pamięci należy stosować rdzenie standardowe, z literą „a" na końcu nazwy pliku (np. „v1fx126a.xbf”). Rdzenie dla konsoli 5200 (litera „g” na końcu nazwy pliku) nie zawierają emulacji rozszerzenia RAMBO.

Rdzenie „GTIA" zawierają tylko emulację układu GTIA, bez dodatkowych funkcji dostępnych w rdzeniach "FX" – dostępna jest jedynie możliwość modyfikacji kolorów w palecie 0. Opis tego rdzenia znajduje się w rozdziale „Rdzeń GTIA".

Rozpoznanie typu uruchomionego rdzenia opisano w rozdziale „Wykrywanie wersji rdzenia".

Różnice pomiędzy wersjami rdzeni dla komputerów XL/XE a konsoli 5200 opisano w rozdziale „Rdzenie dla konsoli 5200".

Rdzenie VGA

Rdzenie VGA nie są obecnie dostępne i nie są rozwijane.

Rdzeń „GTIA"

Rdzeń „GTIA" pozbawiony jest wszelkich rozszerzeń rdzenia FX, zawiera samą emulację układu GTIA a z rejestrów dodatkowych (patrz rozdział „REJESTRY SPRZĘTOWE RDZENIA") używane są tylko CORE_VERSION i MINOR_REVISION służące do odczytu typu i wersji rdzenia oraz CSEL, CR, CG i CB służące do modyfikacji palety.

Rdzeń „GTIA” w odróżnieniu od rdzenia FX emuluje działanie linii opóźniającej PAL, powodującej obniżenie pionowej rozdzielczości kolorów („PAL blending”), co jest wykorzystywane w niektórych programach do zwiększenia ilości kolorów na ekranie. Rdzeń FX wysyła czysty sygnał RGB pozbawiony tego efektu.

CORE_VERSION ma dla rdzenia GTIA wartość 0x11.

MINOR_REVISION dla rdzenia GTIA v1.06a ma wartość 0x06 a dla rdzenia GTIA v1.06r ma wartość 0x86.

Rdzenie dla konsoli 5200

Dostępne dla konsoli 5200 rdzenie "FX" i "GTIA" różnią się od tych dostępnych dla komputerów XL/XE następująco:

- GTIA jest mapowane na obszar $C000-$CFFF,
- (tylko rdzeń FX) : MEMAC-B nie istnieje,
- (tylko rdzeń FX) : okno MEMAC-A jest ustawione na stałe pod adres $D800 a jego długość wynosi zawsze 4KB ($D800-$E7FF) i nie można tych parametrów zmienić pomimo tego, że odpowiednie bity istnieją w rejestrze MEMAC_CONTROL (bity te są ignorowane).

Rdzenie dla konsoli 5200 nie obsługują rozszerzenia pamięci RAMBO.

Wykrywanie wersji rdzenia

Niniejszy opis obowiązuje od rdzenia FX v1.21 oraz GTIA v1.01 i będzie również obowiązywał w następnych wersjach.

Typ rdzenia należy wykryć przez odczyt rejestru CORE_VERSION:

Gdy CORE_VERSION = 0x10 - mamy do czynienia z rdzeniem FX 1.xx.
Gdy CORE VERSION = 0x11 - mamy do czynienia z rdzeniem GTIA 1.xx.
Inna wartość - rdzeń nieznany lub brak VBXE.

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

Następnie odczytujemy MINOR_REVISION. Bit 7 tego rejestru oznacza, czy rdzeń posiada emulację rozszerzenia RAMBO 256K (b7 = 1) lub jej nie posiada (b7 = 0). Na ogół ta informacja nie jest nam potrzebna, więc zerujemy ten bit maską #$7F. bity 6-0 zawierają zakodowaną BCD wersję rdzenia (np. $26 = rdzeń 1.26, $06 = rdzeń 1.06). Umawiamy się, że rdzenie kompatybilne ze sobą będą różniły się jedynie bitami 0-3 w MINOR_REVISION. Różnica na bitach 4-6 oznacza, że rdzenie nie są kompatybilne ze sobą.

```
core_fx:
        lda     MINOR_REVISION
        and     #$7F
        sta     exact_core_revision
        and     #$70
        cmp     #$20  ; #$20...#$2F - compatible cores (bugfixes, small improvements only)
        beq     fx_compatible_revision

fx_incompatible_revision:
        jmp     error

core_gtia_emu:
        lda     MINOR_REVISION
        and     #$7F
        sta     exact_core_revision
        jmp     end
```

Rdzeń FX

XDL

XDL (eXtended Display List) jest to lista rozkazów sterujących wyświetlaniem OVERLAY i mapy atrybutów przez VBXE. XDL może zostać załadowana do pamięci VBXE poprzez okno MEMAC (patrz opis MEMAC). XDL może zostać umieszczony w dowolnym miejscu 512KB VRAM VBXE - do wskazania jego początku służą rejestry XDL_ADR0, XDL_ADR1 i XDL_ADR2. Włączanie przetwarzania XDL następuje po ustawieniu bitu XDL_ENABLED w rejestrze VIDEO_CONTROL. Nie ma ograniczenia co do długości XDL. Organizacja ekranu przez XDL jest pionowa (analogicznie jak w przypadku ANTIC DL zaczyna się od góry ekranu).

Struktura XDL :

XDLC (2 bajty)
dodatkowe dane (od 0 do 20 bajtów)
XDLC (2 bajty)
dodatkowe dane (od 0 do 20 bajtów)
...
...
XDLC ze znacznikiem XDLC_END (2 bajty)
dodatkowe dane (od 0 ... 20 bajtów)

XDLC jest to słowo sterujące XDL, ma ono zawsze długość 2 bajtów. Każdy bit tego słowa ma odmienne znaczenie (patrz tabela 1). Część bitów służy do włączania funkcji wyświetlania, część jest informacją, czy kontroler XDL powinien pobrać dodatkowe dane (i jakie). Bity nie są ze sobą powiązane - w dowolnym momencie może być użyta dowolna ich kombinacja. Można np. załadować adres OVERLAY wcale go nie włączając. Przetwarzanie słowa XDLC rozpoczyna się przed początkiem wyświetlania linii.

bajt.bit XDLC	nazwa bitu	znaczenie	dodatkowe dane
1.0	XDLC_TMON	włącz tryb tekstowy OVERLAY	-
1.1	XDLC_GMON	włącz tryb graficzny OVERLAY	-
1.2	XDLC_OVOFF	wyłącz OVERLAY	-
uwagi:	Ustawienie więcej niż jednego z bitów 0.0, 0.1 i 0.2 spowoduje zawsze wyłączenie OVERLAY. Domyślnie (od góry ekranu) OVERLAY jest wyłączony. Nie ustawienie żadnego z tych bitów spowoduje, że zachowany zostanie dotychczasowy stan działania - włączony lub wyłączony OVERLAY. Może to być przydatne np. gdy chcemy tylko zmienić zestaw znaków lub wartości scrollingów.		
1.3	XDLC_MAPON	włącz mapę atrybutów	-
1.4	XDLC_MAPOFF	wyłącz mapę atrybutów	-
uwagi:	Ustawienie więcej niż jednego z bitów 0.3 i 0.4 spowoduje zawsze wyłączenie mapy atrybutów. Domyślnie (od góry ekranu) mapa jest wyłączona. Nie ustawienie żadnego z tych bitów spowoduje, że zachowany zostanie dotychczasowy stan działania - włączona lub wyłączona mapa atrybutów. Może to być przydatne np. gdy chcemy tylko zmienić zestaw znaków lub wartości scrollingów.		
1.5	XDLC_RPTL	następne x linii bez zmian	1 bajt - ilość linii (x)
uwagi:	Po wyświetleniu aktualnej linii będzie jeszcze x kolejnych linii, w których XDL nie będzie przetwarzana a stan wyświetlania (włączone / wyłączone etc.) będzie kontynuowany. Np. chcąc wyświetlić pełen wiersz trybu tekstowego można włączyć tryb tekstowy przez XDLC_TMON i jednocześnie ustawić XDLC_RPTL podając 7 jako ilość dodatkowych linii w rezultacie zostanie wyświetlonych 8 linii czyli pełen wiersz trybu tekstowego.		
1.6	XDLC_OVADR	ustaw adres i krok OVERLAY	5 bajtów (3 bajty adres i 2 bajty krok) kolejność od młodszych do starszych.
uwagi:	Adres OVERLAY jest 19 - bitowy (512KB). Krok oznacza o ile ma się zwiększyć automatycznie adres dla kolejnej linii trybu graficznego lub tekstowego i może być liczbą z zakresu 0 ... 4095. Kolejność danych: 1. AOV[7:0] 2. AOV[15:8] 3. AOV[18:16] 4. OVSTEP[7:0] 5. OVSTEP[11:8] W trybie graficznym wartość OVSTEP dodawana jest do AOV po każdej wyświetlonej linii. W trybie tekstowym automatyczne zwiększenie adresu wystąpi po wyświetleniu ostatniej (dolnej) linii znaków.		

bajt.bit XDLC	nazwa bitu	znaczenie	dodatkowe dane
1.7	XDLC_OVSCRL	ustaw wartości scrollingów dla trybu tekstowego	2 bajty: 1. hscroll (1 bajt) 2. vscroll (1 bajt)
uwagi:	hscroll może być liczbą z przedziału 0 ... 7, gdzie 0 oznacza linię nie przesuniętą a 7 linię przesuniętą o 7 pikseli w lewo. vscroll może być liczbą z przedziału 0 ... 7, gdzie 0 oznacza linię nie przesuniętą a 7 linię przesuniętą o 7 pikseli do góry. Domyślnie (u góry ekranu) hscroll = vscroll = 0. Scrolling dla trybu tekstowego może być zmieniany w każdej linii ekranu. Ustawienie bitu XDLC_OVSCRL nie oznacza włączenia funkcji scrollingu (która jest zawsze włączona) a jedynie USTAWIENIE WARTOŚCI REJESTRÓW SCROLLINGU. Wartości te będą obowiązywały w kolejnych liniach aż do następnej zmiany przez XDL. Jednostką scrollingu poziomego jest piksel o szerokości 1/2 piksela HIRES ANTIC.		
2.0 (drugi bajt XDLC)	XDLC_CHBASE	ustaw zestaw znaków	1 bajt = adres generatora znaków
uwagi:	Generator zawiera 256 znaków 8x8 pikseli i mieści się w pamięci VBXE. Każdy zestaw rozpoczyna się od adresu podzielnego przez 0x800 co daje 256 możliwych adresów w pamięci 512KB. Jak wszystkie inne dane w pamięci VBXE generator jest tam umieszczany korzystając z okna MEMAC.		
2.1	XDLC_MAPADR	ustaw adres i krok mapy atrybutów	5 bajtów (3 bajty adres i 2 bajty krok) kolejność od młodszych do starszych.
uwagi:	Mapa atrybutów może rozpocząć się w dowolnym miejscu VRAM VBXE. Kolejność danych: 1. AMAP[7:0] 2. AMAP[15:8] 3. AMAP[18:16] 4. MAPSTEP[7:0] 5. MAPSTEP[11:8] Wartość MAPSTEP jest dodawana automatycznie do wartości AMAP po zakończeniu wyświetlania pola mapy atrybutów w pionie (to jest po wyświetleniu ostatniej - dolnej - linii tego pola) o ile nie nastąpi jawna zmiana przez XDL.		

bajt.bit XDLC	nazwa bitu	znaczenie	dodatkowe dane
2.2	XDLC_MAPPAR	ustaw wartości scrollingów oraz szerokość i wysokość pola mapy atrybutów	4 bajty: 1. hscroll (1 bajt) 2. vscroll (1 bajt) 3. width (1 bajt) 4. height (1 bajt)
uwagi:	hscroll może być liczbą z przedziału <0 ... 31>, gdzie 0 oznacza linię mapy nie przesuniętą a 31 linię przesuniętą o 31 pikseli w lewo. vscroll może być liczbą z przedziału <0 ... 31>, gdzie 0 oznacza linię mapy nie przesuniętą a 31 linię przesuniętą o 31 pikseli do góry. width - szerokość pola mapy w punktach <7 ... 31> == 8 do 32 punktów (odpowiadających rozdzielczości HIRES ANTICa) height - wysokość pola mapy w liniach <0 ... 31> == 1 do 32 linii hscroll i vscroll mapy nie powinien przekroczyć wartości odpowiednio width i height. Domyślnie (u góry ekranu) przyjmowane są wartości: hscroll = vscroll = 0; height = width = 7; (szerokość pola 8x8) Wielkość pola i scrolling mapy atrybutów może być zmieniany w każdej linii ekranu. Jednostką szerokości i wartości scrollingu hscroll mapy atrybutów jest 1 piksel w rozdzielczości HIRES ANTIC. Ustawienie bitu XDLC_MAPPAR nie oznacza włączenia funkcji scrollingu mapy atrybutów (która jest zawsze włączona) a jedynie USTAWIENIE WARTOŚCI REJESTRÓW SCROLLINGU. Wartości te będą obowiązywały w kolejnych liniach aż do następnej zmiany przez XDL.		

<table>
<tr><th>bajt.bit
XDLC</th><th>nazwa bitu</th><th>znaczenie</th><th>dodatkowe dane</th></tr>
<tr><td>2.3</td><td>XDLC_ATT</td><td>Ustalenie szerokości ekranu (zarówno OVERLAY jak i MAPY ATRYBUTÓW) oraz priorytetu OVERLAY względem trybów ANTICa. Zmiana obowiązujących palet kolorów.</td><td>2 bajty:

1. Szerokość OVERLAY i mapy + zmiana palet,
2. Priorytet główny</td></tr>
<tr><td>uwagi:</td><td colspan="3">
BAJT 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: szerokość OVERLAY i MAPY ATRYBUTÓW:

0 = NARROW (256 pikseli, zgodny z ANTIC narrow)
1 = NORMAL (320 pikseli, zgodny z ANTIC normal)
2 = WIDE (336 pikseli, zgodny z ANTIC wide - szerszy o 8 pikseli z każdej strony od trybu NORMAL)

Domyślnie (u góry ekranu) ustawiana jest szerokość NORMAL (320 pikseli).

XDL OV PALETTE: paleta dla OVERLAY obowiązująca od tej linii ekranu. Domyślnie od góry ekranu OVERLAY używa palety nr 1.

XDL PF PALETTE: paleta dla PLAYFIELD oraz PMG obowiązująca od tej linii ekranu. Domyślnie od góry ekranu jest to paleta nr 0.

UWAGA: jeżeli włączona jest mapa atrybutów, wówczas palety dla OVERLAY i PLAYFIELD/PMG wybierane są przez dane mapy.

BAJT 2: Priorytet główny.

b0 - 1 = OVERLAY ponad PM0, 0 = OVERLAY zasłonięty przez PM0
b1 - 1 = OVERLAY ponad PM1
b2 - 1 = OVERLAY ponad PM2
b3 - 1 = OVERLAY ponad PM3
b4 - 1 = OVERLAY ponad PF0
b5 - 1 = OVERLAY ponad PF1
b6 - 1 = OVERLAY ponad PF2 i PF3
b7 - 1 = OVERLAY ponad COLBAK

Domyślnie (u góry ekranu) priorytet główny ustawiony jest na wartość 255. Priorytet główny jest nieważny, gdy aktywna jest mapa atrybutów - w tym przypadku decyduje jeden z 4 rejestrów priorytetów P0...P3 wybrany przez mapę atrybutów (patrz bajt 4 definicji pola mapy).
</td></tr>
<tr><td>2.4</td><td>XDLC_HR</td><td>Włączenie trybu High Resolution OVERLAY graficznego.</td><td>-</td></tr>
</table>

bajt.bit XDLC	nazwa bitu	znaczenie	dodatkowe dane
uwagi:	Bit ma znaczenie tylko gdy XDLC_GMON == 1. Tryb HR ma rozdzielczość 640 pikseli w poziomie dla standardowej szerokości ekranu i można wyświetlić 16 kolorów z przedziału 0x00 - 0x0F w aktualnie wybranej dla OVERLAY palecie. Każdy piksel reprezentowany jest przez nibble danych (4 bity) w pamięci VBXE. Każdy bajt danych ma 2 nibble: starszy nibble odpowiada za piksel pierwszy z lewej strony.		
2.5	XDLC_LR	Włączenie trybu Low Resolution OVERLAY graficznego.	-
	Bit ma znaczenie tylko gdy XDLC_GMON == 1. Tryb LR ma rozdzielczość 160 pikseli w poziomie dla standardowej szerokości ekranu. Ilość kolorów identyczna jak w standardowym trybie graficznym (256).		
2.6	-	rezerwa (=0)	-
2.7	XDLC_END	koniec XDL (ostatni rekord XDL), czekaj na VSYNC.	-
uwagi:	XDLC_END mówi, że przetwarzane pole XDLC jest ostatnim w liście i po wyświetleniu ekranu wg. jego zawartości należy czekać na synchronizację pionową a następnie rozpocząć przetwarzanie listy XDL od początku.		

Kolejność pobierania danych przez XDL

Dodatkowe dane (adresy, rejestry scrollingu itd.) pobierane są lub nie, w zależności od ustawień bitów w słowie XDLC (patrz tabela - opis XDL). Kolejność ułożenia tych danych jest zawsze taka sama: dane z młodszych bitów pobierane są przed danymi ze starszych bitów XDLC.

- XDLC_RPTL (1 bajt)
- XDLC_OVADR (5 bajtów)
- XDLC_OVSCRL (2 bajty)
- XDLC_CHBASE (1 bajt)
- XDLC_MAPADR (5 bajtów)
- XDLC_MAPPAR (4 bajty)
- XDLC_ATT (2 bajty)

Maksymalnie po słowie XDLC może znajdować się 20 bajtów danych.

Przykład: fragment XDL tworzący 16 linii (2 wiersze) trybu tekstowego:

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

.word XDLC

.byte 15      ;ile linii bez zmian (XDLC_RPTL)
.long adr     ; 3 bajty adresu wyświetlanego (XDLC_OVADR)
.word 160     ; skok automatyczny (XDLC_OVADR)
.byte 0x20    ;(XDLC_CHBASE) CHBASE 0x20 * 0x800
.byte 0       ;(XDLC_ATT) - narrow OVERLAY, palety 0 i 0.
.byte 255     ;(XDLC_ATT) - OVERLAY ma najwyższy priorytet
```

TRYBY OVERLAY

Kombinacje bitów w słowie XDLC służące do włączania poszczególnych trybów OVERLAY VBXE.

XDLC_TMON	XDLC_GMON	XDLC_HR	XDLC_LR	włączany tryb
0	1	0	0	Graficzny SR (320 / 256c)
0	1	1	0	Graficzny HR (640 / 16c)
0	1	0	1	Graficzny LR (160 / 256c)
0	1	1	1	Kombinacja zabroniona
1	0	X	X	Tekstowy 80 kolumn
1	1	X	X	Kombinacja zabroniona, działanie jak XDLC_OVOFF

Tryby graficzne

Tryb SR (Standard Resolution)

Tryb graficzny o rozdzielczości poziomej 256/320/336 punktów (szerokość wybierana przez XDL, o rozdzielczości pionowej decyduje struktura XDL) w 255 kolorach. Za każdy piksel w tym trybie odpowiada jeden bajt w pamięci VBXE. Bajt o wartości zero (0) powoduje, że piksel nie jest wyświetlany (jest przeźroczysty - chyba, że ustawiony jest bit no_trans w rejestrze VIDEO_CONTROL). Pozostałe wartości wybierają z aktywnej palety OVERLAY kolor o numerze C = wartość bajtu. Po wyświetleniu każdej linii trybu graficznego VBXE automatycznie zwiększa adres pobieranych danych dla następnej linii o wartość z przedziału 0 ... 4095 bajtów zapisaną w XDL.

Tryb LR (Low Resolution)

Tryb graficzny o rozdzielczości poziomej 128/160/168 punktów. Pozostałe informacje zgodne z trybem SR.

Tryb HR (High Resolution)

Tryb graficzny o rozdzielczości poziomej 512/640/672 punktów (szerokość wybierana przez XDL, o rozdzielczości pionowej decyduje struktura XDL) w 16 kolorach.
Jeden bajt danych graficznych zawiera w tym trybie informacje o 2 pikselach, po 4 bity na piksel:

b7	b6	b5	b4	b3	b2	b1	b0
Piksel z lewej strony				Piksel z prawej strony			

Przeźroczystość w tym trybie to wartość "0" nibble'a.

Każdy piksel wybiera kolor 0 ... 15 z aktywnej (lokalnie lub globalnie) palety OVERLAY.

Tryb tekstowy

Tryb o rozdzielczości poziomej 64/80/84 znaki (szerokość wybierana przez XDL, o rozdzielczości pionowej decyduje struktura XDL) w 128 lub 16+8 kolorach. Struktura danych w przypadku trybu tekstowego wygląda następująco:

znak (1 bajt), atrybut (1 bajt), znak, atrybut, znak, atrybut, ... itd.

Znak jest liczbą z zakresu 0-255 i jednoznacznie określa, który font z zestawu znaków zawierającego 256 fontów zostanie wyświetlony.

Atrybut ma następującą strukturę:

b7 - decyduje czy znak ma przeźroczyste, czy też barwne tło

gdy b7 = 0 wówczas:

b7	b6	b5	b4	b3	b2	b1	b0
0	kolor ustawionego piksela znaku = 0x00 .. 0x7F						

b0 ... b6 = numer koloru znaku (0...127) czyli kolory 0...127 z aktywnej (lokalnie lub globalnie) palety OVERLAY.

Tło znaku jest przeźroczyste jeżeli bit no_trans w rejestrze VIDEO_CONTROL jest skasowany (0) - w przeciwnym wypadku tło znaku ma kolor nr 128.

gdy b7 = 1 wówczas:

b7	b6	b5	b4	b3	b2	b1	b0
1	kolor ustawionego piksela znaku = 0x00 .. 0x7F kolor skasowanego piksela znaku (tła) = 0x80 .. 0xFF (kolor piksela ustawionego + 0x80)						

b0 ... b6 = numer koloru znaku (0...127 dla znaku i 128...255 dla jego tła) czyli kolory 0...255 z aktywnej (lokalnie lub globalnie) palety OVERLAY.

Tło znaku nie jest przeźroczyste.

Innymi słowy:

kolor znaków (ustawionych pikseli): zawsze 0 ... 127 (atrybut & 127)
kolor tła znaków (skasowanych pikseli):

a) gdy VC bit 2 (no_trans) == 0
gdy (atrybut < 128) -> tło przeźroczyste
w przeciwnym przypadku kolor tła = (atrybut & 127) + 128 (czyli 128 ... 255)

b) gdy VC bit 2 (no_trans) == 1
gdy (atrybut < 128) -> kolor tła = 128
w przeciwnym przypadku kolor tła = (atrybut & 127) + 128 (czyli 128 ... 255)

Pełna linia trybu tekstowego zajmuje w pamięci ilość bajtów odpowiadającą 2 x szerokość linii w znakach. Do tego dochodzi możliwe rozszerzenie widocznej linii o jeden znak z powodu scrollingu hscroll.

Scrolling trybu tekstowego

Patrz opis XDL (tabela).

Kolory przeźroczyste OVERLAY

I. Gdy bit no_trans w rejestrze VIDEO_CONTROL jest ustawiony, wówczas żaden z kolorów OVERLAY nie jest przeźroczysty, zarówno w trybach graficznych jak i trybie tekstowym OVERLAY (niezależnie od stanu bitu trans15 w rejestrze VIDEO_CONTROL).

II. Gdy bit no_trans w rejestrze VIDEO_CONTROL jest skasowany oraz bit trans15 w rejestrze VIDEO_CONTROL jest skasowany (sytuacja standardowa) wówczas :

- OVERLAY graficzny SR / LR : kolor o danej 8-bitowej "0" jest przeźroczysty
- OVERLAY graficzny HR : kolor o wartości nibbla "0" jest przeźroczysty (są 2 nibble (piksele) na bajt danych graficznych)
- OVERLAY tekstowy : kolor tła znaku jest przeźroczysty pod warunkiem, że bit7 atrybutu znaku ma wartość 0.

III. Gdy bit no_trans w rejestrze VIDEO_CONTROL jest skasowany oraz bit trans15 w rejestrze VIDEO_CONTROL jest ustawiony wówczas przeźroczyste są kolory jak w p. II oraz dodatkowo :

- OVERLAY graficzny SR / LR : każdy kolor o danej 0xHF (gdzie H = 0...F, czyli 0x0F, 0x1F, 0x2F itd. do 0xFF) jest przeźroczysty
- OVERLAY graficzny HR : kolor o wartości nibbla "0xF" jest przeźroczysty
- OVERLAY tekstowy : każdy kolor o danej 0xHF (gdzie H = 0...F, czyli 0x0F, 0x1F, 0x2F itd. do 0xFF) jest przeźroczysty

Bit trans15 pozwala na tworzenie niewidzialnych obiektów na OVERLAY, które mogą służyć np. jako płaszczyzny do detekcji kolizji z innymi obiektami OVERLAY.

Priorytety wyświetlania OVERLAY vs PLAYFIELD/PMG

Istnieją następujące rejestry sterujące priorytetem wyświetlania danych PLAYFIELD/PMG i OVERLAY :

- główny rejestr priorytetów (ustawiany przez XDL) - ma domyślnie wartość 255 od góry ekranu, co oznacza, że OVERLAY ma priorytet ponad wszystkimi kolorami PF/PMG i COLBAK.

- rejestry P0, P1, P2 i P3 - używane ZAMIAST rejestru głównego ustawianego przez XDL gdy włączona jest mapa atrybutów. Ich znaczenie jest takie samo jak rejestru głównego, zawartość 4 bajtu danych mapy atrybutów decyduje który z rejestrów P0, P1, P2 czy P3 jest aktywny dla danego pola mapy. Rejestry P0-3 ustawiane są przez użytkownika poprzez zapis odpowiednich wartości w obszarze IO rdzenia FX - patrz opis rejestrów sterujących.

Znaczenie bitów w rejestrze głównym priorytetów (w rejestrach P0, P1, P2 i P3 jest identyczne):

b0 - 1 = OVERLAY ponad PM0, 0 = OVERLAY zasłonięty przez PM0
b1 - 1 = OVERLAY ponad PM1, 0 = OVERLAY zasłonięty przez PM1
b2 - 1 = OVERLAY ponad PM2, 0 = OVERLAY zasłonięty przez PM2
b3 - 1 = OVERLAY ponad PM3, 0 = OVERLAY zasłonięty przez PM3
b4 - 1 = OVERLAY ponad PF0, 0 = OVERLAY zasłonięty przez PF0
b5 - 1 = OVERLAY ponad PF1, 0 = OVERLAY zasłonięty przez PF1
b6 - 1 = OVERLAY ponad PF2 i PF3, 0 = OVERLAY zasłonięty przez PF2 i PF3
b7 - 1 = OVERLAY ponad COLBAK, 0 = OVERLAY zasłonięty przez COLBAK

- W trybach GTIA 9,11 (16 jasności / 16 kolorów) grafika generowana przez ANTIC/GTIA jest traktowana jak kolor COLBAK niezależnie od wartości piksela.

- W przypadku nałożenia się dowolnej kombinacji kolorów PFx/PMx na siebie, o tym czy OVERLAY jest wyświetlany czy też nie decyduje kolor GTIA, który umieszczony zostaje „na wierzchu” przez logikę priorytetów GTIA. W przypadku jednak, gdy logika priorytetów GTIA umieszcza kolory na sobie wykonując na nich operację OR (tak dzieje się przy niektórych kombinacjach kolorów i wartości rejestru GTIA PRIOR = 0), wówczas ustawienie priorytetu na OVERLAY w dowolnym z nakładających się kolorów spowoduje wyświetlenie w tym miejscu piksela OVERLAY (bez żadnych modyfikacji typu OR).

Wykrywanie kolizji OVERLAY - PLAYFIELD/PMG (detekcja rastrowa)

Możliwe jest wykrycie kolizji pomiędzy wyświetlanymi danymi OVERLAY (zarówno w trybie graficznym jak i tekstowym) a dowolnym z kolorów COLPM0,1,2,3,COLPF0,1,2,3 oraz polem mapy atrybutów z ustawionym bitem CATT.

Wykrywanie kolizji odbywa się automatycznie w trakcie wyświetlania obrazu przez VBXE.

Konfiguracja "czułości" detektora kolizji rastrowych odbywa się poprzez zapis do rejestru COLMASK. Znaczenie bitów w COLMASK wyjaśnia poniższa tabela:

bit COLMASK	jeżeli ustawiony, wówczas wykrywamy kolizje obiektów OVERLAY z PLAYFIELD/PMG jeżeli kolor OVERLAY (numer palety jest nieistotny) zawiera się w przedziale:
0	0-31
1	32-63
2	64-95
3	96-127
4	128-159
5	160-191
6	192-223
7	224-255

Dozwolona jest dowolna kombinacja bitów COLMASK.

Odczyt aktualnego kodu kolizji odbywa się z rejestru COLDETECT :

bit COLDETECT	jeżeli ustawiony, wówczas wykryto kolizje obiektów OVERLAY z:
0	kolorem PM0
1	kolorem PM1
2	kolorem PM2
3	kolorem PM3
4	kolorem PF0
5	kolorem PF1
6	kolorem PF2 lub PF3
7	polem mapy atrybutów z ustawionym bitem CATT

Każda z kombinacji bitów COLDETECT jest możliwa - sprawdzanie konkretnej kolizji w rejestrze COLDETECT przez software powinno odbywać się z udziałem odpowiedniej maski AND.

Kasowanie wykrytej kolizji (wyzerowanie rejestru COLDETECT) odbywa się przez zapis dowolnej wartości do rejestru COLCLR.

UWAGA: wykrywane są kolizje jedynie z kolorami "nieprzeźroczystymi" OVERLAY.
UWAGA: priorytety wyświetlania nie mają wpływu na detekcję kolizji.
UWAGA: nie należy mylić powyższego mechanizmu z wykrywaniem kolizji OVERLAY-OVERLAY oferowanym przez blitter (patrz opis blt_collision_mask).

MAPA ATRYBUTÓW

Mapa atrybutów umożliwia lokalną (w obrębie pola mapy, czyli prostokąta o wielkości od 8x1 do 32x32 pikseli HIRES) zmianę lokalnego zestawu kolorów PF0, PF1 i PF2, zmianę lokalnego priorytetu OVERLAY<->ANTIC/GTIA na jeden z 4 predefiniowanych, zmianę lokalnej palety kolorów dla trybów ANTIC i OVERLAY (niezależnie) oraz lokalną zmianę rozdzielczości obrazu generowanego przez duet ANTIC/GTIA z ANTIC HIRES na ANTIC CCR lub odwrotnie. (CCR = Color Clock Resolution - rozdzielczość, w której 1 piksel ma 1 cykl koloru czyli 160 pikseli w poziomie).

Mapa atrybutów umożliwia więc znaczną poprawę jakości (szczególnie ilości kolorów) grafiki ATARI nawet bez użycia OVERLAY.

Możliwości mapy atrybutów:

- wielkość pola mapy (XxY): od 8x1 do 32x32 punktów ANTIC HIRES.
- adres danych mapy w pamięci VBXE: dowolny, z dokładnością do 1 bajtu.
- automatyczne zwiększanie adresu po wyświetleniu pełnej lini mapy: programowane w zakresie 0 do 4095 bajtów.
- scrolling XY z rozdzielczością 1 piksela ANTIC HIRES. sterowany przez XDL, możliwa zmiana rejestrów w każdej linii
- Każde pole mapy definiowane jest przez zestaw 4 bajtów znajdujących się kolejno po sobie w pamięci VBXE. Zawartość tych bajtów określa:
    - bajt 1: lokalny kolor PF0 w trybach CCR / wzorek dla tła w trybach ANTIC HIRES
    - bajt 2: lokalny kolor PF1
    - bajt 3: lokalny kolor PF2
    - bajt 4:
        - lokalny priorytet ANTIC<>OVERLAY (1 z 4 predefiniowanych)
        - wymuszenie lokalnej zmiany rozdzielczości (bit RES)
        - lokalną paletę kolorów dla trybów ANTIC i OVERLAY (niezależnie). Do wyboru jedna z 4 palet. ANTIC i OVERLAY mogą korzystać z dwóch różnych albo z tej samej palety w obrębie pola mapy
        - wsparcie rastrowej detekcji kolizji mapy atrybutów z obiektami OVERLAY (bit CATT)

Mapa atrybutów jest całkowicie sterowana przez XDL, tzn. jej adres w pamięci, rejestry scrollingu, wielkości pola są zawarte w programie XDL.

Dane mapy atrybutów

Każde pole mapy definiowane jest w pamięci VBXE przez 4 kolejno położone po sobie bajty:

b7	b6	b5	b4	b3	b2	b1	b0
PF0. Lokalny substytut rejestru COLPF0 GTIA (lub overlay data w trybach ANTIC HIRES)							

b7	b6	b5	b4	b3	b2	b1	b0
PF1. Lokalny substytut rejestru COLPF1 GTIA							

b7	b6	b5	b4	b3	b2	b1	b0
PF2. Lokalny substytut rejestru COLPF2 GTIA							

b7	b6	b5	b4	b3	b2	b1	b0
MAP PF PALETTE		**MAP OV PALETTE**		**CATT**	**RES**	**PS**	

b6, b7 - (MAP PF PALETTE) - wybór lokalnej palety dla obiektów normalnego PLAYFIELD oraz PMG. Wartość 0 - 3 wybiera paletę o takim numerze.

b4, b5 - (MAP OV PALETTE) - wybór lokalnej palety dla danych OVERLAY. Wartość 0 - 3 wybiera paletę o takim numerze.

UWAGA: Paleta wybrana przez mapę atrybutów (MAP PF/OV PALETTE) ma priorytet nad wybraną przez XDL (XDL PF/OV PALETTE), tzn. jeżeli mapa atrybutów jest włączona to wówczas na obszarze przez nią pokrytym obowiązują palety wybierane przez mapę atrybutów zamiast palety domyślnej lub wybranej przez XDL.

b3 - (CATT) - bit pomocniczy wykrywania kolizji. Jeżeli jest on ustawiony, wówczas w trakcie wyświetlania obrazu istnieje możliwość wykrycia kolizji obiektów OVERLAY z danym polem mapy atrybutów niezależnie od wyświetlanych przez ANTIC/GTIA danych. Patrz : opis detekcji kolizji rastrowych.

b2 - (RES) - lokalne przełączanie HIRES <-> CCR (1 = włączone). Bit ten "odwraca" wybrany przez standardową DL ANTIC-a rozdzielczość robiąc (lokalnie w obrębie pola mapy) z rozdzielczości CCR tryb ANTIC HiRes i odwrotnie. (patrz też opis dokładnego działania bitu RES w dalszej części dokumentu).

b0, b1 - (PS) - wybór jednego z 4 rejestrów priorytetów OVERLAY <-> ANTIC/GTIA: P0, P1, P2 lub P3.

PS		**wybierany rejestr**
0	0	Rejestr priorytetów P0
0	1	Rejestr priorytetów P1
1	0	Rejestr priorytetów P2
1	1	Rejestr priorytetów P3

UWAGA: Na obszarze aktywnej mapy atrybutów globalne rejestry GTIA : COLPF0, COLPF1 i COLPF2 oraz rejestr globalny priorytetów nie są używane.

UWAGA: Szerokość linii Mapy Atrybutów może być ustawiana jako zgodna z trybem wide/normal/narrow ANTIC-a za pomocą XDL (patrz XDLC_ATT).

Mapa Atrybutów w trybie ANTIC CCR

W trybie ANTIC CCR (natywnym lub wymuszonym przez bit RES - patrz dalej) włączona Mapa Atrybutów unieważnia stan globalnych rejestrów kolorów GTIA: COLPF0, COLPF1 oraz COLPF2 podmieniając je na odpowiednio PF0, PF1 oraz PF2 zawarte w danych mapy (bajty 1, 2 oraz 3). COLPF3 nie ma swojego odpowiednika w danych mapy i pozostaje zawsze rejestrem globalnym, dotyczącym całego ekranu.

Mapa atrybutów w trybie ANTIC HIRES

W trybie ANTIC HIRES (natywnym lub wymuszonym przez bit RES - patrz dalej) pierwszy bajt definicji pola mapy atrybutów odpowiadający normalnie (w trybach ANTIC CCR) za lokalny kolor PF0, zyskuje inne znaczenie (ponieważ w tym trybie COLPF0 nie jest używany) - kolejne (począwszy od najstarszego) bity wartości tu wpisanej decydują jaki kolor tła wybierany jest w kolejnych (od strony lewej do prawej) pikselach obrazu :

bit = 0 -> tło ma kolor PF2 (z 3 bajtu mapy atrybutów)
bit = 1 -> tło ma kolor COLPF3 (z globalnego rejestru GTIA)

Jest to jakby dodatkowy jednobitowy "overlay". Dzięki niemu ANTIC HIRES ma 3 kolory w obrębie pola mapy zamiast dwóch.

Dla pól mapy o szerokości 8 pikseli ANTIC HIRES każdy bit tego overlay wybiera kolor tła dla pojedynczego piksela. Przy polu mapy o szerokości od 9 do 16 pikseli każdy bit overlay wybiera kolor tła dla 2 sąsiednich pikseli - a przy polu mapy o szerokości powyżej 16 pikseli każdy bit overlay wybiera kolor tła dla 4 sąsiadujących pikseli.

Bajt 2 mapy wybiera kolor dla zapalonego piksela. Należy pamiętać, że jeżeli bit XCOLOR w rejestrze VIDEO_CONTROL jest skasowany to wówczas starsze 4 bity koloru piksela nie są uwzględniane - za to podstawiane są tutaj starsze 4 bity z koloru tła piksela.
Jeżeli XCOLOR jest ustawiony, wówczas kolor piksela jest w pełni niezależny od koloru tła oraz wszystkie rejestry kolorów mają pełne 8 bitów zamiast 7 (co przekłada się dla standardowej palety w 16 zamiast 8 jasności).

Opis działania bitu RES

Bit RES w czwartym bajcie mapy atrybutów pozwala na lokalne odwrócenie sposobu interpretacji danych graficznych przesyłanych przez ANTIC.

Normalnie ANTIC decyduje (na początku każdej linii obrazu) jak GTIA (VBXE) będzie interpretował i wyświetlał dane :

- jako ANTIC HIRES (tryby 2, 3 oraz F z DL ANTICA), lub
- jako ANTIC CCR (pozostałe tryby graficzne i tekstowe)

Bit RES, kiedy jest ustawiony, pozwala na lokalne wymuszenie zmiany (odwrócenia) interpretacji przez VBXE - można dzięki temu wyświetlić część poziomej linii ekranu w zmienionej rozdzielczości. Obowiązują następujące zasady konwersji danych:

a) konwersja CCR -> sztuczny tryb ANTIC HIRES

Każdy piksel w rozdzielczości CCR ma szerokość 2 pikseli w trybie ANTIC HIRES i jest zmieniany na dwa sąsiednie piksele w trybie ANTIC HIRES wg tabeli :

kolor piksela w CCR	**HIRES, piksel z lewej strony**	**HIRES, piksel z prawej strony**
BACKGROUND	Tło HIRES	Tło HIRES
COLPF0	Tło HIRES	kolor PF1 (2 bajt mapy)
COLPF1	kolor PF1 (2 bajt mapy)	Tło HIRES
COLPF2	kolor PF1 (2 bajt mapy)	kolor PF1 (2 bajt mapy)
COLPF3	kolor PF1 (2 bajt mapy)	kolor PF1 (2 bajt mapy)

Gdzie "Tło HIRES" - kolor generowany wg zasad opisanych wcześniej w rozdziale "**Mapa atrybutów w trybie ANTIC HIRES".** Zależnie od wartości 1 bajtu mapy może być to kolor PF2 (z 3 bajtu mapy) lub globalny kolor COLPF3.

b) konwersja ANTIC HIRES -> sztuczny tryb CCR

Pary sąsiadujących bitów zamieniane są na piksel w trybie CCR wg tabeli:

HIRES, piksel z lewej strony	**HIRES, piksel z prawej strony**	**kolor wynikowy**
0	0	PF0 (1-szy bajt mapy)
0	1	PF1 (2-gi bajt mapy)
1	0	PF2 (3-ci bajt mapy)
1	1	globalny COLPF3

MODYFIKACJA PALET RGB

VBXE pozwala na wyświetlenie naraz do 1024 kolorów (z 21-bitowej palety sprzętowej 2097152 barw, najmłodszy bit rejestrów składowych koloru jest pomijany). Istnieją 4 zestawy (palety) po 256 predefiniowanych przez użytkownika kolorów każda.

Paletę RGB dla każdego z tych 1024 (4 x 256) kolorów uaktualnia się w VBXE następująco:

- wpisujemy numer palety (0-3) do rejestru PSEL,
- wpisujemy numer koloru w palecie (0-255) do rejestru CSEL,
- wpisujemy wartość składowej R koloru do rejestru CR
- wpisujemy wartość składowej G koloru do rejestru CG
- wpisujemy wartość składowej B koloru do rejestru CB

UWAGA: Zapis do CB (ale nie do CR ani CG!) spowoduje automatyczne zwiększenie CSEL o 1 - można od razu wpisywać wartości następnego koloru w palecie bez "jawnej" modyfikacji CSEL.

UWAGA: Jeżeli CSEL "zwiększy się" automatycznie z 255 na 0 to pomimo tego PSEL pozostanie BEZ ZMIAN - aby edytować następną paletę należy wykonać zapis odpowiedniej wartości do PSEL, inaczej zaczniemy ponowną modyfikację tej samej palety od koloru nr 0.

Taki sposób modyfikacji palety pozwala na uzyskiwanie efektów bazujących na rotacji palety z minimalnym obciążeniem procesora. Wraz ze zmianą sposobu przełączania aktualnie używanej palety pozwala na prostszą i wizualnie bardziej poprawną realizacje efektów typu FADE (płynne wygaszenie, czy też "zapalenie" obrazka). Realizując tego typu efekty tylko na części palety programista jest w stanie ciekawiej wyświetlać np napisy końcowe w creditsach, mając przy tym statyczną grafikę obok.

UWAGA: Zapisy do każdego z rejestrów CR, CB i CG powodują natychmiastowy efekt na ekranie w postaci zmiany wyświetlanego koloru (o ile dana paleta i kolor jest aktualnie wyświetlany) - składowe nie są buforowane.

UWAGA: Rdzenie GTIA pozwalają na modyfikację palety 0 w identyczny sposób. W tych rdzeniach nie istnieje rejestr PSEL.

Domyślnie po włączeniu zasilania paleta nr 0 zawiera kolory dla PLAYFIELD/PMG będące zmodyfikowaną paletą "laoo.act". Palety 1-3 zawierają wszystkie składowe równe 0 (wszystkie kolory czarne).

Proszę pamiętać, że przy modyfikacji palety nr 0 w obowiązku programisty jest przywrócenie jej wartości domyślnych przed wyjściem z programu.

MEMAC

MEMAC jest częścią rdzenia VBXE odpowiedzialną za udostępnianie systemowi (CPU i ANTIC-owi) pamięci 512KB zainstalowanej w VBXE.

Zapis i odczyt do i z pamięci udostępnionej przez MEMAC kosztuje VBXE 1 cykl zegara PCLK (14.18MHz) na jeden bajt. Po stronie CPU i ANTICa zapis i odczyt nie różni się od dostępu do jakiegokolwiek obszaru RAM / ROM lub IO. Technicznie VBXE odłącza normalny RAM komputera i podstawia własny RAM korzystając z sygnału EXTSEL. Dodatkowo ze względu na wprowadzenie "ruchomego" okna MEMAC-A począwszy od wersji FX v1.20 rdzeń sprawdza stan linii CASINH MMU (należy ją przylutować do VBXE - patrz opis w dokumencie "opis montazu.pdf" lub w "historii wersji" w niniejszym dokumencie) aby wykluczyć możliwość konfliktu z pamięcią OS, BASIC lub CARTRIDGE ROM.

MEMAC posiada dwa niezależne "okna":

MEMAC-A - okno o definiowalnym adresie bazowym oraz definiowalnej długości.
MEMAC-B - okno o stałej długości 16K umieszczone pod stałym adresem $4000.

UWAGA: Jeżeli Okno MEMAC-A jest tak skonfigurowane, że jego obszar pokrywa się w całości lub w części z oknem MEMAC-B, wówczas priorytet ma zawsze MEMAC-A. Nie ma takiego konfliktu, jeżeli jedno (obojętnie które) okno służy do dostępu przez 6502 a drugie przez ANTIC - w tym przypadku okna mogą się bez problemu w całości lub częściowo "pokrywać".

UWAGA: W przypadku, gdy używamy okna leżącego w obszarze pomiędzy $4000 a $7FFF (MEMAC-B lub odpowiednio skonfigurowane MEMAC-A), wówczas aby zapisać lub odczytać dane z/do pamięci VRAM należy najpierw bezwzględnie wyłączyć rozszerzenie pamięci RAM poprzez zapis wartości 0xFF lub 0xFE do rejestru PORTB ($D301). Spowodowane jest to tym, że część rozszerzeń RAM (zależnie od konstrukcji) ma wyższy priorytet dostępu do szyny komputera niż VBXE. W efekcie niemożliwy jest scenariusz, że którekolwiek okno MEMAC częściowo lub w całości przykryje bank AKTYWNEJ pamięci extended.

UWAGA: Dotyczy tylko rdzenia w wersji r (np. "FX v1.26r") :

Specjalną funkcją MEMAC jest emulacja standardowego rozszerzenia RAM 320KB RAMBO (64KB standardowej pamięci + 256KB rozszerzonej, bez osobnego dostępu ANTIC-a i CPU). Rozszerzenie RAM mapowane jest do górnej połowy pamięci VRAM VBXE (adresy 0x40000 - 0x7FFFF w VBXE) - należy pamiętać, że ta pamięć jest współdzielona przez emulowany ext. RAM i normalne funkcje VBXE.

Rdzeń FX w wersji a (np. "FX v1.26a") pozbawiony jest funkcji emulacji rozszerzenia RAM, tym samym umożliwia normalną pracę tradycyjnym rozszerzeniom zamontowanym w komputerze.

MEMAC-A

Okno MEMAC-A może być umieszczone w dowolnym miejscu przestrzeni adresowej 6502 z dokładnością do $1000 (4096) bajtów, czyli może się rozpoczynać od adresu:

$0000, lub
$1000, lub
$2000, lub
...
itd.
...
$E000, lub
$F000

Wielkość okna jest również konfigurowalna - do wyboru jest:

$1000 (4KB), lub
$2000 (8KB), lub
$4000 (16KB), lub
$8000 (32KB)

UWAGA: Okno rozpoczynające się np. od $F000 a mające mieć teoretycznie wielkość $2000, $4000 lub $8000 bajtów zostanie "ucięte" na adresie $FFFF (nie nastąpi "zawinięcie" okna na adres $0000).

UWAGA: Jeżeli w obszarze, w którym ma być okno MEMAC-A pojawi się pamięć ROM (OS lub BASIC lub CARTRIDGE lub rejestry sprzętowe $D000-$D7FF) wówczas pamięć ROM (lub rejestry sprzętowe) ma (mają) zawsze pierwszeństwo - pamięć VBXE będzie w tym miejscu niedostępna zarówno do odczytu jak i zapisu. Konsekwencją tego jest konieczność wyłączania OS jeżeli chcemy okno MEMAC-A umieścić np. od adresu $E000. W obszarze rejestrów sprzętowych pamięć VBXE nie może być dostępna, ponieważ w ATARI nie można odłączyć rejestrów sprzętowych.

Okno MEMAC-A może być:

- wyłączone (wówczas jest tam standardowy RAM komputera)
- włączone dla CPU (CPU widzi RAM VBXE, ANTIC widzi RAM komputera)
- włączone dla ANTICa (ANTIC widzi RAM VBXE, CPU widzi RAM komputera)
- włączone dla CPU i ANTICa

ANTIC i CPU widzą zawsze ten sam obszar pamięci VBXE (mają wspólny rejestr wyboru banku - MEMS).

MEMAC-A sterowany jest przez 2 rejestry (patrz też opis rejestrów na końcu dokumentu):

MEMAC_CONTROL (lub MEMC) - ten rejestr służy do wyboru bazowego położenia okna, jego wielkości oraz włączania dostępu do VRAM niezależnie dla 6502 i ANTICa. Rejestr jest typu R/W (zapis / odczyt). Po RST bity konfiguracyjne MCE i MAE są wyzerowane (okno niewidoczne dla CPU i dla ANTICa). Pozostałe bity po RST bez zmian.

MEMAC_BANK_SEL (lub MEMS) decyduje o tym, jaki obszar pamięci 512K VRAM "widziany" jest poprzez okno MEMAC-A. Rejestr R/W, RST nie zmienia ostatnio wpisanej wartości. Dodatkowo w tym rejestrze znajduje się bit globalnego zezwolenia na włączenie okna MEMAc (bit MGE). Bit ten jest kasowany automatycznie po RST.

Opis bitów rejestru MEMAC_BANK_SEL (lub MEMS):

Dla okna o długości $1000:
b6...b0 wybór banku 4K (0...127)
b7 zarezerwowane (wpisywać zawsze 0)

Dla okna o długości $2000:
b6...b1 wybór banku 8K (0...63)
b7 zarezerwowane, b0 ignorowane

Dla okna o długości $4000:
b6...b2 wybór banku 16K (0...31)
b7 zarezerwowane, b0 i b1 ignorowane

Dla okna o długości $8000:
b6...b3 wybór banku 32K (0...15)
b7 zarezerwowane, b0, b1 i b2 ignorowane

MEMAC-B

Okno MEMAC-B jest umieszczone począwszy od stałego adresu $4000. Jego długość wynosi zawsze 16K.

Okno MEMAC-B może być:

- wyłączone (wówczas jest tam standardowy RAM komputera)
- włączone dla CPU (CPU widzi RAM VBXE, ANTIC widzi RAM komputera)
- włączone dla ANTICa (ANTIC widzi RAM VBXE, CPU widzi RAM komputera)
- włączone dla CPU i ANTICa

Konfiguracją okna MEMAC-B oraz wyborem jednego z 32 banków steruje jeden rejestr MEMAC_B_CONTROL (w skrócie MEMB). W odróżnieniu od rejestrów MEMAC-A, rejestr MEMB jest uproszczony i służy tylko do zapisu - odczytywana wartość to zawsze 255.

BLITTER

Blitter wbudowany w rdzeń VBXE pozwala na kopiowanie i wypełnianie obszarów pamięci VBXE o dowolnej wielkości.

Blitter sterowany jest przez tzw. BlitterList - specjalną sekwencję danych umieszczaną w pamięci VBXE przez procesor (programistę) ATARI. Ogólna struktura BlitterList wygląda następująco:

BCB
BCB
BCB
...
BCB ze skasowanym znacznikiem NEXT

BCB - Blitter Command Block - BlitterList składa się z jednego lub więcej BCB.
BCB jest zestawem danych informacyjnych dla blittera, Każdy BCB opisuje jedną operację blittera. BCB ma długość 21 bajtów (patrz opis struktury BCB poniżej).

Blitter kopiuje dane na zasadzie "linia po linii". Programista ustala początek (adres) pierwszej linii obszaru docelowego i źródłowego, długość linii obiektu źródłowego, ilość linii obiektu źródłowego, odstęp (w bajtach) pomiędzy kolejnymi liniami obiektów źródłowego i docelowego, odstęp pomiędzy danymi wewnątrz pojedynczej linii źródłowej i docelowej, oraz dane dodatkowe:

- zoom X i Y
- maskę AND i XOR dla danych źródłowych
- dodatkowy opcjonalny "pattern feature"
- maskę wykrywania kolizji.

Manipulując znakami source/dest step x/y (patrz niżej) można odwracać obiekt w pionie i poziomie, zmieniać kierunek kopiowania zapobiegając nadpisywaniu etc. Należy jednak pamiętać o właściwym dla wybranego kierunku ustaleniu wartości początkowych adresów source_adr i dest_adr.

STRUKTURA BCB

lp.	nazwa	ilość bajtów	opis
1	source_adr	3	Adres początku danych źródłowych blittera w VRAM (19 bitów ważnych, najmłodszy bajt pierwszy)
2	source_step_y	2	Wartość ze znakiem z zakresu < - 4096 ... 4095 >. Określa odległość w bajtach pomiędzy początkami kolejnych linii danych źródłowych w VRAM.
3	source_step_x	1	Wartość ze znakiem z zakresu < - 128 ... 127 >. Określa "skok" w bajtach pomiędzy kolejnymi adresami danych źródłowych w ramach kopiowania linii danych. Dla danych "ciągłych" należy wpisywać tu wartość 1 (lub -1).
4	dest_adr	3	Adres początku obszaru docelowego dla operacji blittera w VRAM (19 bitów ważnych, najmłodszy bajt pierwszy)
5	dest_step_y	2	Wartość ze znakiem z zakresu < - 4096 ... 4095 >. Określa odległość w bajtach pomiędzy początkami kolejnych linii obszaru docelowego w VRAM.
6	dest_step_x	1	Wartość ze znakiem z zakresu < - 128 ... 127 >. Określa "skok" w bajtach pomiędzy kolejnymi adresami docelowymi w ramach kopiowania linii danych. Dla ciągłego obszaru docelowego wpisywać tu wartość 1 (lub -1).
7	blt_width	2	Szerokość kopiowanego obiektu źródłowego. (ilość bajtów danych w linii) - 1. Wpisujemy to wartość od 0 do 511.
8	blt_height	1	Wysokość kopiowanego obiektu źródłowego. (ilość linii) - 1. Wpisujemy to wartość od 0 do 255.
9	blt_and_mask	1	maska AND źródła danych
10	blt_xor_mask	1	maska XOR źródła danych
11	blt_collision_mask	1	maska AND wykrywania kolizji
12	blt_zoom	1	zoom X i Y przenoszonego obiektu
13	pattern_feature	1	sterowanie wypełnianiem wzorkiem
14	blt_control	1	dodatkowe informacje (patrz opis)

source_adr

Dane źródłowe dla operacji blittera mogą znajdować się w dowolnym miejscu pamięci VBXE.

source_step_y

Mówi o ile bajtów ma zostać zwiększony / zmniejszony source_adr po skopiowaniu poziomej linii danych źródłowych o szerokości blt_width.

source_step_y = -4096 ... 4095

Stosując wartości ujemne można np. odwracać obiekt w pionie.

source_step_x

Kolejne dane źródłowe w linii mogą znajdować się bezpośrednio po sobie (gdy source_step_x = 1 lub source_step_x = -1) lub być przeplecione innymi danymi, których blitter pobierając dane źródłowe ma nie uwzględniać. Skok adresu może przyjmować wartości:

source_step_x = -128 ... 127

Stosując wartości ujemne można np. odwracać obiekt w poziomie.

dest_adr

Dane docelowe dla operacji blittera mogą znajdować się w dowolnym miejscu pamięci VBXE.

dest_step_y

Mówi o ile bajtów ma zostać zwiększony / zmniejszony dest_adr po skopiowaniu poziomej linii danych źródłowych o szerokości blt_width.

dest_step_y = -4096...4095

dest_step_x

Kolejne adresy docelowe w linii mogą znajdować się bezpośrednio po sobie (gdy dest_step_x = 1 lub dest_step_x = -1) - mogą jednak być oddalone o większy równomierny skok.

dest_step_x = -128 ... 127

blt_width

Szerokość kopiowanego obiektu źródłowego (W BAJTACH) pomniejszona o 1.

blt_width = 0...511 co odpowiada szerokości 1...512 bajtów - co w trybach OVERLAY SR i LR oznacza szerokość 1...512 w punktach. W trybie HR OVERLAY oznacza to szerokość 2...1024 punktów.

blt_height

Wysokość kopiowanego obiektu źródłowego (w liniach) pomniejszona o 1.

blt_height = 0...255 co odpowiada wysokości 1...256 linii

blt_and_mask

Kasowanie bitów danych źródłowych:

source' = source AND (&) blt_and_mask

Tej operacji poddawany jest każdy bajt danych źródłowych. "source" to odczytany przez blitter bajt danych źródłowych.

blt_xor_mask

Odwracanie bitów danych źródłowych:

source'' = source' XOR (^) blt_xor_mask

Tej operacji poddawany jest każdy bajt danych źródłowych, po wcześniejszym skasowaniu wybranych bitów przez blt_and_mask.

blt_collision_mask

Maska wykrywania kolizji. Wykrywanie kolizji następuje w trybach 1,2,3,4,5 i 6 pracy blittera (patrz opis blt_control).

W trybach 1, 2, 3, 4 i 5 pracy blittera :

Każdy bit maski kolizji odpowiada jednemu segmentowi palety. Każdy segment palety ma 32 kolory (0-31, 32-63, 64-95 itd). Używając odpowiednio palety, możemy zapewnić sobie wykrywanie kolizji z poszczególnymi grupami obiektów jakie mamy na ekranie.

Wykrycie kolizji następuje jeżeli spełniony jest warunek:
(source'' != 0 && collision_sr()), gdzie:

source'' - jest to dana źródłowa (0-255) po operacjach AND i XOR.

```
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" są to odczytane dane docelowe (przed zapisem zmienionych przez blitter).

Przykładowo, przeznaczając kolory 0-31 (bit 0) na tło, a kolory 32-63 oraz 96-127 (bit 1 oraz 3) na elementy statyczne planszy pozwala nam w prosty sposób sprawdzić czy wystąpiła kolizja danego obiektu z elementem statycznym planszy, czy też nie ma żadnej kolizji tego typu, tym samym zwalniając procesor od konieczności sprawdzenia z czym nastąpiła kolizja.

Tym sposobem możemy również w prosty sposób zrezygnować z usług ANTIC'a i pozwolić na generowanie całego wyglądu ekranu przez VBXE zwalniając tym samym magistrale dla procesora (brak cykli DMA, zostają jedynie cykle REFRESH).

W trybie 6 pracy blittera (tryb HR pracy VBXE) :

Paleta liczy zaledwie 16 kolorów, więc na każdy bit maski przypadają dwa kolejne kolory. Oba nibble bajtu są rozpatrywane oddzielnie.

Wykrycie kolizji następuje jeżeli spełniony jest warunek:
(source" != 0 && collision_hr()), gdzie:

source" - jest to dana źródłowa o wartości 0-15 (nibble starszy lub młodszy - rozpatrywane oddzielnie) po operacjach AND i XOR.

```
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" są to odczytane dane docelowe (przed zapisem zmienionych przez blitter). Nibble starszy lub młodszy (rozpatrywane oddzielnie).

UWAGA: dozwolona jest dowolna kombinacja bitów blt_collision_mask.

blt_zoom

Bity BLT_ZOOMX i BLT_ZOOMY :

Możliwe jest rozciągnięcie obiektów docelowych w poziomie i w pionie poprzez pomnożenie wielkości obiektu źródłowego (blt_width / blt_height) o całkowity mnożnik od 1 do 8.

b7	b6	b5	b4	b3	b2	b1	b0
-	BLT_ZOOMY			-	BLT_ZOOMX		

Dane źródłowe pozostają bez zmian (blt_width / blt_height odnosi się zawsze do rozmiarów danych źródłowych) - powiększa się tylko obszar docelowy. Powiększenie to jest równe:

ZOOMX(Y) = BLT_ZOOMX(Y) + 1

pattern_feature

b7	b6	b5	b4	b3	b2	b1	b0
IN_USE	-	PATTERN_WIDTH					

Pattern Feature umożliwia "zapętlanie" danych źródłowych w obrębie "poziomej" linii danych. Jeżeli bit IN_USE jest skasowany, wówczas funkcja Pattern Feature nie jest używana - dane źródłowe nigdy nie są zapętlane.

Jeżeli IN_USE = 1, wówczas:

po skopiowaniu PATTERN_WIDTH+1 (1 ... 64) bajtów nastąpi przywrócenie adresu danych źródłowych takiego, jaki obowiązywał na początku linii (zamiast kolejnej modyfikacji adresu źródłowego o source_step_x) - po czym nastąpi ponowne skopiowanie PATTERN_WIDTH+1 bajtów, przywrócenie adresu danych źródłowych z początku linii itd. aż do skopiowania w sumie blt_width+1 bajtów (kopiowanie patternu zostanie przerwane, jeżeli (blt_width+1)%(PATTERN_WIDTH+1) != 0 czyli blt_width ma wyższy priorytet).

Funkcja Pattern Feature nie ma wpływu na zmianę adresu danych źródłowych "w pionie" - zawsze obowiązuje przyrost (lub zmniejszenie adresu) o source_step_y bajtów względem początku poprzedniej kopiowanej linii.

blt_control

Bajt kontrolujący tryb pracy i zachowanie blittera.

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	NEXT	MODE		

MODE	opis

0	Tzw. "COPYMODE". Każdy bajt danych źródłowych source" jest kopiowany do obszaru danych docelowych - bez uwzględniania przeźroczystości (wartości 0) i sprawdzania kolizji.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	dest' = source";
	WriteDest(dest');

MODE	opis

1	Podstawowy tryb pracy blittera. Dane source" są kopiowane do obszaru docelowego POD WARUNKIEM, że (source" != 0). Jeżeli blt_collision_mask ma wartość różną od zera, wówczas przed skopiowaniem odczytana zostanie wartość danych dest i jeżeli spełniony jest warunek collision_sr() wówczas nastąpi "wykrycie" kolizji i zapisanie kodu dest do rejestru BLT_COLLISION_CODE. Wykrywanie kolizji powoduje spowolnienie pracy blittera. Jeżeli jest niepotrzebne - wówczas lepiej ustawić blt_collision_mask na 0 - wykrywanie będzie wyłączone a blitter będzie pracował szybciej.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (collision_sr()) BLT_COLLISION_CODE = dest;
	  dest' = source";
	  WriteDest(dest');
	}

2	Wartość zapisywana dest' jest sumą arytmetyczną danych source" oraz wartości dotychczasowej dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (collision_sr()) BLT_COLLISION_CODE = dest;
	  dest' = dest + source";
	  WriteDest(dest');
	}

3	Wartość zapisywana dest' jest wynikiem operacji logicznej OR na każdym bicie danych source" oraz wartości dotychczasowej dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (collision_sr()) BLT_COLLISION_CODE = dest;
	  dest' = dest | source";
	  WriteDest(dest');
	}

MODE	opis

4	Wartość zapisywana dest' jest wynikiem operacji logiczne AND na każdym bicie danych source" oraz wartości dotychczasowej dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	dest = ReadDest();
	if (source" != 0 && collision_sr())
	{
	  BLT_COLLISION_CODE = dest;
	}
	dest' = dest & source";
	WriteDest(dest');

5	Wartość zapisywana dest' jest wynikiem operacji logicznej XOR na każdym bicie danych source" oraz wartości dotychczasowej dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	  if (collision_sr()) BLT_COLLISION_CODE = dest;
	  dest' = dest ^ source";
	  WriteDest(dest');
	}

6	Wsparcie dla trybu HR OVERLAY graficznego. Zasadniczo tryb podobny do trybu 1, z tą różnicą, że rozpatrywanie przeźroczystości i detekcja kolizji przeprowadzane są na poziomie nibble'a (4 bitów) a nie całego bajtu.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source" = source' ^ blt_xor_mask;
	if (source" != 0)
	{
	  dest = ReadDest();
	
	  if (source"[3:0] != 0)
	  {
	   if (collision_hr()) 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 (collision_hr()) BLT_COLLISION_CODE[7:4] = dest[7:4];
	   dest'[7:4] = source"[7:4];
	  }
	 else dest'[7:4] = dest[7:4];
	
	  WriteDest(dest');
	}

MODE	opis
7	nieużywane / zarezerwowane

NEXT - jeżeli ten bit jest skasowany (= 0), wówczas ten BCB jest ostatnim w BlitterList i po wykonaniu zaprogramowanego zadania blitter zakończy pracę kasując flagę BUSY w rejestrze BLITTER_BUSY oraz wywołując przerwanie IRQ jeżeli ustawiono wcześniej bit zezwolenia na przerwanie w rejestrze IRQ_CONTROL. Jeżeli bit NEXT jest ustawiony (= 1) wówczas po zakończeniu operacji opisywanej przez aktualny BCB blitter zachowa się następująco:

- skasuje flagę BUSY w rejestrze BLITTER_BUSY
- jednocześnie ustawi flagę BCB_LOAD w rejestrze BLITTER_BUSY
- odczyta z pamięci VBXE dane kolejnego BCB
- ustawi flagę BUSY w rejestrze BLITTER_BUSY
- rozpocznie wykonywanie nowej operacji
- po zakończeniu skasuje flagę BUSY
- sprawdzi stan bitu NEXT
- itd. (koniec pracy lub następny BCB)

Blitter i stałe dane źródłowe

Jeżeli warunek:

(blt_and_mask == 0)

jest spełniony, wówczas dane źródłowe są STAŁE - nie zależą od odczytanych z obszaru źródłowego i wartość danych źródłowych wynosi dokładnie blt_xor_mask. Blitter optymalizuje swoją pracę, pomijając fazę zbędnego odczytu danych źródłowych z pamięci - operacja zostaje wykonana szybciej. Wypełnianie pamięci stałą wartością jest więc szybkie.

REJESTRY SPRZĘTOWE RDZENIA FX

Adres	Zapis	Odczyt
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 bity 0 ... 7	BLT_COLLISION_CODE
Dx51	BL_ADR1 bity 8 ... 15	255
Dx52	BL_ADR2 bity 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 lub 7 zależnie od podłączenia VBXE w komputerze.

REJESTRY SPRZĘTOWE RDZENIA GTIA

Adres	Zapis	Odczyt
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 lub 7 zależnie od podłączenia VBXE w komputerze.

OPIS REJESTRÓW

VIDEO_CONTROL

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

legenda:

- pierwsza linia: numer bitu b0 - b7
- druga linia: nazwa / funkcja bitu ('-' = bit nieużywany)
- trzecia linia:
    'w' - bit do zapisu
    'r' - bit do odczytu
    'rw' - bit do zapisu i odczytu
    "-0" stan "0" po RESET
    "-1" stan "1" po RESET
    "-x" stan nieokreślony po RESET

xcolor:

1 = wyświetlaj kolory PM0, PM1, PM2, PM3, PF0, PF1, PF2, PF3, BKGND z uwzględnieniem bitu 0 (16 zamiast 8 jasności) oraz w trybie ANTIC hires (graficzny / tekstowy) uniezależnij kolor zapalonego piksela od koloru tła (kolor zapalonego piksela jest pobierany wtedy z PF1).

0 = pełna zgodność z GTIA. 8 jasności w rejestrach (128 kolorów) i kolor piksela w ANTIC hires zależny od koloru tła.

UWAGA: wartość bitu xcolor dotyczy zarówno rejestrów globalnych GTIA jaki i kolorów zmodyfikowanych przez mapę atrybutów.

xdl_enable:

1 = włącz przetwarzanie XDL począwszy od najbliższej synchronizacji pionowej.

0 = wyłącz przetwarzanie XDL począwszy od najbliższej synchronizacji pionowej.

no_trans:

0 = w trybach graficznych i tekstowym OVERLAY kolor o kodzie danej 8-bitowej (lub nibbla dla trybu graficznego HR) 0 traktuje jako przeźroczystość i wyświetla znajdujący się "pod spodem" obraz PLAYFIELD lub PMG.

1 = OVERLAY graficzny i tekstowy wyświetla wszystkie dane bez żadnych kolorów przeźroczystych.

Bit ten pozwala na bezproblemowe wyświetlanie obrazków w dokładnie 256 kolorach.

UWAGA: bit no_trans nie ma wpływu na pracę blittera, który w większości trybów pracy

kolor o kodzie danej 0 traktuje jako kolor przeźroczysty.
trans15:

Bit ten ma znaczenie tylko gdy no_trans = 0. Pozwala na uzyskanie dodatkowych kolorów przeźroczystych: patrz rozdział "Kolory przeźroczyste OVERLAY".

XDL_ADR0 (eXtended Display List ADdRess)

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

bity 0 ... 7 adresu XDL w pamięci VBXE.

Adres XDL należy ustalić przed włączeniem przetwarzania XDL (xdl_enable w rejestrze VIDEO_CONTROL).

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

bity 8 ... 15 adresu XDL w pamięci VBXE.

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

bity 16 ... 18 adresu XDL w pamięci VBXE.

CSEL (Color SELect)

b7	b6	b5	b4	b3	b2	b1	b0
nr koloru (startowego) do modyfikacji składowych RGB							
w-x	w-x	w-x	w-x	w-x	w-x	w-x	w-x

Przed rozpoczęciem modyfikacji składowych RGB kolorów należy numer koloru, który chcemy modyfikować (lub od którego chcemy rozpocząć modyfikację) wpisać do CSEL. UWAGA: CSEL ulega automatycznemu zwiększeniu o 1 po zapisie do rejestru CB.

PSEL (Palette SELect)

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	nr palety do modyfikacji	
-	-	-	-	-	-	w-x	w-x

Przed rozpoczęciem modyfikacji składowych RGB kolorów należy numer palety wpisać do PSEL. PSEL w odróżnieniu od CSEL nigdy nie zmienia się automatycznie.

CR (Component Red)

b7	b6	b5	b4	b3	b2	b1	b0
Składowa 7 bitowa R (czerwony)							-
w-x	w-x	w-x	w-x	w-x	w-x	w-x	-

Zmiana składowej R wybranego koloru następuje NATYCHMIAST po zapisie do CR.

CG (Component Green)

b7	b6	b5	b4	b3	b2	b1	b0
Składowa 7 bitowa G (zielony)							-
w-x	w-x	w-x	w-x	w-x	w-x	w-x	-

Zmiana składowej G wybranego koloru następuje NATYCHMIAST po zapisie do CG.

CB (Component Blue)

b7	b6	b5	b4	b3	b2	b1	b0
Składowa 7 bitowa B (niebieski)							-
w-x	w-x	w-x	w-x	w-x	w-x	w-x	-

Zmiana składowej B wybranego koloru następuje NATYCHMIAST po zapisie do CB. Zapis do CB powoduje ponadto automatyczne zwiększenie CSEL o 1. (Gdy CSEL = 255 wówczas CSEL przyjmie wartość 0). Wartość rejestru PSEL pozostaje bez zmian.

REJESTRY MEMAC-A

MEMAC_CONTROL (MEMC)

b7	b6	b5	b4	b3	b2	b1	b0
Adres bazowy okna MEMAC				MCE	MAE	Wielkość okna	
rw-x	rw-x	rw-x	rw-x	rw-0	rw-0	rw-x	rw-x

Adres bazowy okna MEMAC

Np. wpisanie 4 jako adresu bazowego ($4x do rejestru MC) spowoduje, że okno będzie się rozpoczynało od adresu $4000. $7x spowoduje, że okno będzie się rozpoczynało od adresu $7000 itp.

MCE - MEMAC CPU ENABLE - wpisanie 1 konfiguruje okno MEMAC do dostępu przez 6502. Bit ten po RST przyjmuje wartość 0.

MAE - MEMAC ANTIC ENABLE - wpisanie 1 konfiguruje okno MEMAC do dostępu przez ANTIC. Bit ten po RST przyjmuje wartość 0.

UWAGA: okno zostanie włączone jeżeli bit MGE (MEMAC GLOBAL ENABLE) w rejestrze MEMAC_BANK_SEL ma wartość 1. Czyli warunek włączenia okna jest taki:

Dostęp dla 6502: MGE == 1 i MCE == 1
Dostęp dla ANTICa: MGE == 1 i MAE == 1

Wielkość okna:

b1	b0	wielkość okna
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	nr banku 0 ... 127 dla okna 4K						
MGE	nr banku 0 ... 63 dla okna 8K						-
MGE	nr banku 0 ... 31 dla okna 16K					-	-
MGE	nr banku 0 ... 15 dla okna 32K				-	-	-
rw-0	rw-x	rw-x	rw-x	rw-x	rw-x	rw-x	rw-x

Bity b0 - b6: Wybór banku pamięci VRAM "widzianego" przez okno MEMAC. Działanie zależne od wybranej wielkości okna.

Bit b7 - MGE : MEMAC GLOBAL ENABLE - wartość 1 włącza okno MEMAC wg konfiguracji wybranej w MEMAC_CONTROL. Ten globalny bit "włączający" okno pozwala na traktowanie rejestru MEMAC_CONTROL jako wyłącznie konfiguracyjnego, czyli można ustawić MEMAC CONTROL na początku programu na żądaną konfigurację okna a później do wyboru banku oraz włączania / wyłączania okna posługiwać się już tylko rejestrem MEMAC_BANK_SEL.

UWAGA: Program, który używa MEMAC, po zakończeniu swojego działania powinien do obydwu rejestrów MEMAC (MEMC i MEMS) wpisać 0.

REJESTR MEMAC-B

MEMAC_B_CONTROL (MEMB)

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

MBCE - MEMAC-B CPU ENABLE - wartość 1 włącza bank MEMAC-B do dostępu dla 6502.

MBAE - MEMAC-B ANTIC ENABLE - wartość 1 włącza bank MEMAC-B do dostępu dla ANTICa.

Nr banku - 32 banki po 16K dają dostęp do całej pamięci 512K VRAM VBXE.

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

bity 0 ... 7 adresu BlitterList w pamięci VBXE.

Adres BlitterList (tm) w pamięci VBXE ma 18 bitów. BlitterList może się mieścić w dowolnym miejscu z dokładnością do jednego bajtu. Nie ma limitu długości BlitterList. Po wpisaniu adresu do BL_ADR można uruchomić blitter. Po zakończeniu pracy blittera wartość BL_ADR pozostaje BEZ ZMIAN.

BL_ADR należy ustalić przed uruchomieniem blittera.

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

bity 8 ... 15 adresu BlitterList w pamięci VBXE.

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

bity 16 ... 18 adresu BlitterList w pamięci VBXE.

BLITTER_START

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

Wpisanie wartości 1 na pozycji bitu b0 spowoduje wystartowanie blittera - rozpoczyna się odczyt BlitterList a następnie Blitter przechodzi do wykonywania właściwego zadania wg danych otrzymanych w BlitterList.
W trakcie pracy blittera - jeżeli zachodzi taka potrzeba - można wpisać wartość 0 na pozycji bitu b0 - wówczas blitter zostaje natychmiast zatrzymany. Mechanizm ten pozwala przerwać pracę zapętlonego w nieskończoność blittera (jest to możliwe, gdy np. uruchomimy blitter przy wypełnionej wartościami 0xFF całej pamięci VBXE - wówczas w BlitterList ciągle będzie pojawiał się znacznik "NEXT"). W normalnej pracy blittera funkcja STOP nie powinna być używana.

BLT_COLLISION_CODE

Kod wykrytej kolizji w czasie pracy blittera. Kolizja została wykryta jeżeli BLT_COLLISION_CODE != 0. Kod odpowiada niezerowej wartości koloru piksela nadpisanej przez blitter. BLT_COLLISION_CODE jest zerowany automatycznie w momencie uruchomienia blittera.

BLITTER_BUSY

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

Rejestr ten ma wartość różną od 0 w czasie, gdy blitter pracuje - tj. przetwarza dane BlitterList / BCB (BCB_LOAD = 1) lub wykonuje właściwą operację (BUSY = 1). Po przejściu w stan IDLE rejestr ma wartość 0 i można przygotować blitter do nowego zadania.

IRQ_CONTROL

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

Włączenie przerwania IRQ generowanego po wykonaniu wszystkich zadań z BlitterList. (w momencie przejścia Blittera ze stanu BUSY do stanu IDLE).
IRQE = 0 - przerwanie wyłączone.
IRQE = 1 - zezwolenie na generowanie przerwań.

Dodatkowo zapis dowolnej wartości bitu IRQE spowoduje skasowanie żądania przerwania o ile już wystąpiło.

IRQ_STATUS

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

IRQF = 0 - brak zgłoszenia przerwania przez VBXE.
IRQF = 1 - zgłoszenie przerwania końca pracy blittera aktywne (linia IRQ w stanie 0). Należy skasować poprzez zapis do rejestru IRQ_CONTROL.

CORE_VERSION

Typ rdzenia:

0x10 = rdzeń FX v1.XY
0x11 = rdzeń GTIA v1.XY

MINOR_REVISION

b7	b6	b5	b4	b3	b2	b1	b0
RAMBO	Cyfra **X** numeru wersji			Cyfra **Y** numeru wersji			
r	r	r	r	r	r	r	r

RAMBO = 1 - rdzeń z emulacją rozszerzenia RAM RAMBO 256K (rdzeń "r").
RAMBO = 0 - rdzeń standardowy ("a").

Numerowanie wersji rdzenia: v1.XY, gdzie X i Y są zakodowane powyżej (X=0...7, Y=0...9).

Rdzenie FX różniące się jedynie cyfrą Y są kompatybilne programowo ze sobą (rdzeń z wyższą wartością Y zawiera prawdopodobnie poprawki błędów).

Rdzenie FX różniące się cyfrą X są niekompatybilne i program, który wykryje rdzeń FX o nieznanej mu wartości X powinien odmówić pracy.

Wersja rdzenia FX:

0x26 = rdzeń FX v1.26a/g
0xa6 = rdzeń FX v1.26r

Wersja rdzenia GTIA:

0x06 = GTIA v1.06a/g
0x86 = GTIA v1.06r

COLMASK

Maska wyboru zakresów danych OVERLAY służąca do wykrywania kolizji z grafiką PLAYFIELD/PMG.

Patrz opis wykrywania kolizji rastrowych.

COLCLR

Wpisanie dowolnej wartości spowoduje wyzerowanie rejestru COLDETECT.

COLDETECT

"Zatrzaśnięty" rejestr wykrytych kolizji rastrowych. Kolizje są wykrywane w trakcie wyświetlania obrazu. Bity ustawione oznaczają, że wykryto kolizję.

Patrz opis wykrywania kolizji rastrowych.

P0, P1, P2, P3

Rejestry wyboru priorytetów OVERLAY - PLAYFIELD/PMG używane gdy włączona jest mapa atrybutów (na obszarze przykrytym przez tę mapę). Patrz rozdział "Priorytety wyświetlania OVERLAY vs PLAYFIELD/PMG".

HISTORIA WERSJI

Wersja 1.26

- bugfix w blitterze: nie działała prawidłowo funkcja ZOOM w osi X kopiowanego obiektu.
- poprawa emulacji trybu GR11 GTIA (poprawna luminancja dla pikseli o wartości 0000).
- zmiana działania bitu XCOLOR w rejestrze VIDEO_CONTROL rdzenia FX – od teraz (gdy XCOLOR = 1) piksele w trybach ANTIC HIRES nie tylko uzyskują kolor w pełni niezależny od tła ale też w procesie generacji obrazu uczestniczą jako kolor PF1 na normalnych prawach tak jak w trybach nie-hires, czyli podlegają np. priorytetom GTIA.
- zmiana w obsłudze priorytetów OVERLAY-GTIA. Od teraz jeżeli nakładają się dowolne kolory generowane przez GTIA i dla choćby jednego z nich ustawiony jest priorytet „wyświetl OVERLAY" to wyświetlany jest OVERLAY bez żadnej modyfikacji typu „OR kolor GTIA" jak to miało miejsce w rdzeniach 1.24 i starszych. Taka sytuacja może wystąpić np. przy ustawieniu wartości rejestru PRIOR GTIA na 0 i nałożeniu się kolorów PF/PMG.
- zmiana w rejestrach priorytetów OVERLAY-GTIA – rejestrze głównym ładowanym przez XDL i w rejestrach p0-p3 wybieranych mapą koloru. Od teraz bit 6 odpowiada jednocześnie za wybór priorytetu dla PF2 i PF3 a bit 7 pozwala na wybór priorytetu dla COLBAK (ukrycie OVERLAY za COLBAK), co dotąd nie było możliwe.
- w rdzeniach „GTIA" dodano PAL BLENDING (symulacja działania linii opóźniającej koloru w dekoderach PAL).
- w rdzeniach „GTIA" możliwa jest od teraz modyfikacja palety (tylko paleta 0) tak samo jak w rdzeniu FX – adresy rejestrów do tego służących są takie same jak w rdzeniu FX (nie jest używany tylko rejestr PSEL).

Wersja 1.25

- wersja nieoficjalna

Wersja 1.24

- drobne zmiany wewnętrzne
- poprawka błędu występującego przy zmianie (przez XDL) szerokości overlay z narrow na normal / wide oraz z normal na wide : błąd skutkował dodatkowym (błędnym) pobraniem XDLC z pamięci VRAM co objawiało się niespodziewanym przesunieciem "w górę" następnych linii na ekranie
- dodano obsługę monitorów VGA

Wersja 1.23

- zmiany wewnętrznych timingów, zmiany w obsłudze EXTSEL, z punktu widzenia programisty nieistotne

Wersja 1.22

- Zmiany poprawiające stabilność sprzętu

Wersja 1.21

- poprawka w kodzie emulacji GTIA dotycząca rejestru PRIOR ($D01B)
- od teraz rdzeń "R" będzie miał ustawiony bit 7 w rejestrze MINOR_REVISION a rdzeń "A" będzie miał ten bit skasowany

Wersja 1.20

- nowy MEMAC-A i MEMAC-B.
- UWAGA: od teraz wymagane jest podłączenie sygnału CASINH do punktu 9 (AUX4) złącza J3 na PCB VBXE v1.x. Dla VBXE v2.0 sygnał CASINH należy podłączyć do punktu 4 złącza J3. CASINH dostępne jest na nóżce 16 MMU lub nóżce 4 FREDDIE.

Wersja 1.10

- mapa atrybutów : naprawa działania bitu RES
- mapa atrybutów od teraz nie tylko może zmienić kolor pikseli i tła w trybie ANTIC HIRES ale również dodać własne elementy graficzne do tła, zwiększając ilość kolorów.
- Blitter. BCB ma teraz 21 bajtów - zwiększenie funkcjonalności.

Wersja 1.09

- likwidacja OV_COLOR_SHIFT
- XDL może od teraz zmieniać paletę dla OVERLAY i PLAYFIELD/PMG (używane są te same bity, które były dotychczas przeznaczone dla OV_COLOR_SHIFT)
- nowy model detekcji kolizji przez blitter - patrz opis "blt_collision_mask"
- nowy model detekcji kolizji rastrowych
- możliwość wykrycia kolizji pomiędzy OVERLAY a polem mapy atrybutów (patrz detekcja kolizji rastrowych oraz opis mapy atrybutów - bit CATT)
- likwidacja mechanizmu MSEL/RGB - nowe rejestry służące do szybszej modyfikacji kolorów RGB (CSEL, PSEL, CR, CG, CB)
- likwidacja mechanizmu MSEL/PRIORMAP - rejestry P0, P1, P2 i P3 mają od teraz swoje dedykowane adresy w obszarze IO rdzenia FX
- bit XDLC_OVATT przemianowany na XDLC_ATT

Wersja 1.08

- poprawka bezpieczeństwa (update wysoce rekomendowany) w kodzie obsługi obszaru MEMAC A/B - od teraz VBXE uszanuje fakt, że zewnętrzne rozszerzenie wymusi w tym obszarze dostęp do pamięci przez sygnał EXTSEL (jako pochodnej sygnału CAS INHIBIT z komputera). W takim przypadku VBXE zrezygnuje z podłączania własnej pamięci. Należy pamiętać, żeby przez wykorzystaniem obszaru MEMAC B wpisać do $D301 wartość $ff (lub inną, taką która wyłączy ewentualne zewnętrzne (lub wewnętrzne, "wbudowane" w rdzeń FX) rozszerzenie RAM), ponieważ mogłoby ono uzyskać na drodze wymuszenia EXTSEL priorytet nad pamięcią VRAM dostępną przez MEMAC.
- zapis dowolnej wartości pod dowolny adres z zakresu $D080 - $D0FF (kopie rejestrów GTIA) spowoduje programowy reset VBXE identyczny z tym po

naciśnięciu klawisza RESET w komputerze, czyli wyłączone zostanie przetwarzanie XDL (a więc znika OVERLAY i mapa atrybutów) oraz niektóre bity rejestrów sterujących zostaną zainicjowane (ustawione w stan wyjściowy). UWAGA: żaden RESET nie jest w stanie przywrócić defaultowej palety VBXE o ile została ona zmodyfikowana przez dowolny program. Przywrócenie palety możliwe jest jedynie poprzez ponowne "wgranie" rdzenia. Powyższa funkcja pozwala na przywrócenie standardowego ekranu przy ciepłym / zimnym starcie komputera zainicjowanym przez skok do procedur systemu a nie przez naciśnięcie klawisza RESET.
- wprowadzenie rejestru MINOR_REVISION (tylko odczyt) dla ułatwienia identyfikacji wersji załadowanego rdzenia.

Wersja 1.0 beta 7

- BLITTER: poprawka w kodzie blittera - flaga NEXT w blt_control powinna działać zgodnie z opisem - poprzednio powodowała zapętlenie blittera.
- BLITTER: zamiana maski BLT_OR_MASK na BLT_XOR_MASK. Dzięki temu zachowując poprzednie możliwości dostajemy też nowe.
- BLITTER: nowy bit INTLVE w BLT_ZOOM - umożliwia niezależną pracę blittera na atrybutach lub kodach znaków w trybie tekstowym OVERLAY.
- BLITTER: nowy bajt w BCB (od teraz BCB ma 19 bajtów) : pattern_feature - umożliwia kopiowanie w ramach jednej "poziomej" linii danych powtzrzalnych "wzorków" o długości od 1 do 64 bajtów.

Wersja 1.0 beta 6

- zmiana w funkcjonowaniu atrybutów znaków i wyborze kolorów dla znaku i jego tła w trybie tekstowym OVERLAY.

Wersja 1.0 beta 5

- zmiana mapowania RAM emulowanego rozszerzenia 320KB RAMBO z dolnej połowy VRAM VBXE do górnej połowy VRAM VBXE - obecnie pamięć współdzielona z emulowanym rozszerzeniem to adresy 0x40000 - 0x7FFFF w obszarze adresowym VBXE.

Wersja 1.0 beta 4

- dodałem wykrywanie kolizji pomiędzy obiektami OVERLAY a grafiką PLAYFIELD (ANTIC) i PMG (GTIA). Nowe rejestry: COLMASK, COLCLR, COLDETECT.
- nowe tryby graficzne OVERLAY - tryb HR (rozdzielczość pozioma 640 pikseli w 16 kolorach) oraz tryb LR (rozdzielczość pozioma 160 pikseli w 256 kolorach).
- możliwość włączenia dodatkowych kolorów przeźroczystych w OVERLAY - mogą być pomocne do wykrywania kolizji. Służy do tego nowy bit "trans15" w rejestrze VIDEO_CONTROL.
- nowy tryb pracy blittera (tryb 6) jako wsparcie dla trybu HR OVERLAY graficznego.
- usunąłem sztywny podział kolorów na paletę OVERLAY i paletę ANTIC/GTIA. Od teraz są 4 palety, które można przydzielać niezależnie dla OVERLAY i Antic/GTIA o ile aktywna jest mapa atrybutów. Jeżeli mapa atrybutów jest nieaktywna wówczas

OVERLAY zawsze korzysta z palety 1 a kolory ANTIC/GTIA z palety 0 (jest to zgodne z dotychczasowym zachowaniem).

- zmiany w czwartym bajcie mapy atrybutów.
- kolor OVERLAY można zmodyfikować dodatkowo przez rejestr OV_COLOR_SHIFT ustawiany przez XDL.
- możliwość powiększania obiektu docelowego kopiowanego przez blitter w pionie i w poziomie (niezależnie) - wielkość obiektu mnożona jest przez całkowity mnożnik od x1 do x8.
- Wydłużenie bloku BCB w BlitterList z 17 do 18 bajtów (ze względu ma nowe atrybuty blt_zoomx i blt_zoomy).

Wersja 1.0 beta 3

- nowy bit b2 (*no_trans*) w rejestrze VIDEO_CONTROL - pozwala na wyłączenie przeźroczystości w OVERLAY graficznym i traktowanie koloru o kodzie danej 0 jak każdego innego koloru włącznie z możliwością zdefiniowania palety RGB.

Wersja 1.0 beta 2

- zmiana kolejności bitów w słowie XDLC.
- całkowicie zmieniony nowy model mapy atrybutów (pole mapy opisywane czterema bajtami zamiast jednego).
- usunięcie teraz już zbędnego mechanizmu MSEL/COLORMAP.

Wersja 1.0 beta 1

Pierwsza publiczna wersja.