VBXE FPGA core "FX"
wersja 1.0 (beta 2)

Podręcznik programisty

Spis treści

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 bufory 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 buforów 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.		

bajt.bit XDLC	nazwa bitu	znaczenie	dodatkowe dane
2.3	XDLC_OVATT	Ustalenie szerokości ekranu i priorytetu OVERLAY względem trybów ANTICa	2 bajty 1. Szerokość overlay 2. Priorytet główny
uwagi:	Szerokość OVERLAY: 0 = NARROW (256 pikseli, zgodny z ANTIC narrow) 1 = NORMAL (320 pikseli, zgodny z ANTIC normal) 2 = WIDE (338 pikseli, zgodny z ANTIC wide) Domyślnie (u góry ekranu) ustawiana jest szerokość NORMAL (320 pikseli). 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 b7 - 1 = OVERLAY ponad PF3 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 zestawów priorytetów P0...P3 narzuconych przez mapę atrybutów.		
2.4	-	rezerwa (=0)	-
2.5	-	rezerwa (=0)	-
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.		

TRYBY OVERLAY

Tryb graficzny

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). Pozostałe wartości wybierają z palety kolor o numerze C = wartość + 256, czyli z zakresu 257 - 511. 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 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 = 0x100 .. 0x17F						

b0 ... b6 = numer koloru znaku (0...127) czyli kolory 256 ... 383 z palety.
Tło znaku jest przeźroczyste.

gdy b7 = 1 wówczas:

b7	b6	b5	b4	b3	b2	b1	b0
1	kolor tła znaku = 0x110 ... 0x117			kolor ustawionego piksela = 0x100 ... 0x10F			

b0 ... b3 = numer koloru znaku (0...15) czyli kolory 256 ... 271 z palety.
b4 ... b6 = numer koloru tła (0 ... 7) mapowane jako kolory 272 ... 279 w palecie.

Pełna linia trybu tekstowego zajmuje więc 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).

Priorytety OVERLAY <-> tryby ANTIC/GTIA

Technika regulacji priorytetów pomiędzy OVERLAY a trybami ANTIC/GTIA jest następująca:

Każdy z rejestrów kolorów GTIA (poza BKGND) ma w głównym rejestrze priorytetów VBXE bit mówiący czy (jeżeli ANTIC/GTIA wyświetla dany kolor) kolor ten ma być zastąpiony przez kolor OVERLAY czy też ma zasłonić kolor OVERLAY. OVERLAY ma zawsze priorytet jedynie nad kolorem BKGND.

Patrz też opis XDL (znaczenie bitów w rejestrze priorytetów).

Rejestr główny priorytetów ustawiany jest przez XDL. Dodatkowo, jeżeli włączona jest mapa atrybutów, wówczas rejestr główny przestaje mieć znaczenie a zaczyna obowiązywać jeden z czterech predefiniowanych rejestrów priorytetów wybierany dla danego pola mapy atrybutów. Rejestry predefiniowane ustawia się za pomocą mechanizmu MSEL/PRIORMAP (patrz opis mapy atrybutów oraz mechanizmu MSEL/PRIORMAP).

W trybach GTIA 9,11 (16 jasności / 16 kolorów) OVERLAY jest umieszczony zawsze przed obrazem generowanym przez ANTIC (ale może być za PMG).

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_OVATT (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_OVATT + 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    ;CHBASE 0x20 * 0x800
.byte 0       ;(xdlc_ovatt) - narrow overlay
.byte 255     ;(xdlc_ovatt) - overlay najwyższy priorytet
```

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 (jest to jedyne powiązanie mapy atrybutów z wyświetlaniem OVERLAY), oraz lokalną zmianę rozdzielczości obrazu generowanego przez duet ANTIC/GTIA z HIRES na 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 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 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. Zawatrość tych bajtów określa w kolejności:
  - lokalny kolor PF0
  - lokalny kolor PF1
  - lokalny kolor PF2
  - lokalny priorytet ANTIC<>OVERLAY (1 z 4 predefiniowanych) i wymuszenie lokalnej zmiany rozdzielczości

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

4 wybieralne rejestry priorytetów o funkcji zgodnej z rejestrem głównym definiowane są za pomocą mechanizmu MSEL/PRIORMAP.

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
Lokalny substytut rejestru COLPF0 GTIA							

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

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

b7	b6	b5	b4	b3	b2	b1	b0
RES	zarezerwowane - wpisywać zawsze 0					PSEL1	PSEL0

\
b7 (RES) - lokalne przełączanie HIRES <-> CCR (1 = włączone)

b0, b1 (PSEL0 / PSEL1) - wybór jednego z 4 predefiniowanych priorytetów OVERLAY <-> ANTIC/GTIA.

PSEL1	**PSEL0**	**wybierany rejestr**
0	0	Rejestr priorytetów P0 (wpisywany do MB0)
0	1	Rejestr priorytetów P1 (wpisywany do MB1)
1	0	Rejestr priorytetów P2 (wpisywany do MB2)
1	1	Rejestr priorytetów P3 (wpisywany do MB3)

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

MSEL/RGB

Mechanizm zmiany składowych RGB kolorów w palecie.

VBXE pozwala na wyświetlenie naraz do 512 kolorów (z 21-bitowej palety 2097152 barw) z których 256 są to kolory "starych" trybów ANTIC/GTIA a drugie 256 to kolory OVERLAY.

Paletę RGB dla każdego z 512 kolorów uaktualnia się w VBXE wpisując składowe (8-bitowe, najmłodszy bit zawsze = 0) kolorów R, G, B do rejestrów odpowiednio MB1, MB2 i MB3. Następnie młodszą część numeru koloru (0 - 255) wpisujemy do rejestru MB0 a do rejestru MSEL wpisuje się wartość 0xc0 + najstarszy bit numeru koloru na pozycji b0.

Czyli:

MSEL = 0xc0 dla koloru z zakresu 0 - 255 (paleta GTIA), lub
MSEL = 0xc1 dla koloru z zakresu 256 - 511 (paleta OVERLAY).

Wszystkie składowe wybranego koloru uaktualniane są w momencie zapisu do MSEL.

Paleta ANTIC/GTIA : to kolory o numerach od 0 do 255. Numer koloru odpowiada wartości którą wpisujemy do rejestrów PFx, PMx i BKGND układu GTIA

Paleta OVERLAY : to kolory o numerach od 256 - 511. Tryb graficzny OVERLAY używa kolorów 257 - 511. Tryb tekstowy OVERLAY używa 2 wariantów opisanych wcześniej przy okazji opisu trybu tekstowego.

Oba tryby OVERLAY (graficzny / tekstowy) indeksują kolor bez uwzględniania najstarszego bitu jego numeru - tzn domyślnie dodawana jest wartość 256 do numeru koloru.

MSEL/PRIORMAP

Mechanizm ten pozwala ustawić 4 rejestry priorytetów ANTIC/GTIA <-> OVERLAY używane i wybierane przez mapę atrybutów zamiast głównego rejestru priorytetów ustawianego przez XDL.

Mapa atrybutów pozwala (poprzez bity b0 i b1 w czwartym bajcie danych mapy) wybrać dla aktualnego pola jeden z czterech predefiniowanych priorytetów.

- do rejestru MB0 wpisujemy definicję priorytetów P0
- do rejestru MB1 wpisujemy definicję priorytetów P1
- do rejestru MB2 wpisujemy definicję priorytetów P2
- do rejestru MB3 wpisujemy definicję priorytetów P3
- do rejestru MSEL wpisujemy wartość 0x80. W tym momencie wewnętrzne rejestry priorytetów Px zostaną uaktualnione.

Znaczenie bitów w każdym rejestrze priorytetu jest analogiczne jak w rejestrze głównym priorytetów, używanym gdy mapa atrybutów jest (lokalnie lub na całym ekranie) wyłączona. Patrz opis XDL. Na obszarze aktywnej mapy kolorów rejestr główny priorytetów nie obowiązuje.

MEMAC

MEMAC jest częścią rdzenia VBXE odpowiedzialną za udostępnianie systemowi (CPU i ANTIC) 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.

Dostęp do pamięci VBXE odbywa się poprzez okna w dwóch obszarach:

0x2000 - 0x3FFF - "MEMAC A" (okno 8 KB)
0x4000 - 0x7FFF - "MEMAC B" (okno 16 KB)

Każdy z tych obszarów może być:

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

Dodatkowo wybór banku dla CPU i ANTICa jest niezależny, tzn. CPU może "widzieć" inny obszar RAM VBXE niż ANTIC

Okno MEMAC A ma wielkość 8KB -> RAM VBXE podzielony jest na 64 banki.
Okno MEMAC B ma wielkość 16KB -> RAM VBXE podzielony jest na 32 banki.

Specjalną funkcją MEMAC jest emulacja standardowego rozszerzenia RAM 320KB RAMBO. Rozszerzenie jest wyłączane, gdy włączony jest dostęp do pamięci VBXE przez okno MEMAC B.

Rejestry sterujące

MA_CPU

bit 7 - 1 = CPU widzi RAM VBXE w obszarze MEMAC A
0 = CPU widzi tam normalny RAM komputera
bit 6 - zarezerwowany
bity 0 do 5 wybór 1 z 64 banków 8 KB RAM VBXE

MA_ANTIC

bit 7 - 1 = ANTIC widzi RAM VBXE w obszarze MEMAC A
0 = ANTIC widzi tam normalny RAM komputera
bit 6 - zarezerwowany
bity 0 do 5 wybór 1 z 64 banków 8 KB RAM VBXE

MB_CPU

bit 7 - 1 = CPU widzi RAM VBXE w obszarze MEMAC B
0 = CPU widzi tam normalny RAM komputera
bit 6 - zarezerwowany
bit 5 - zarezerwowany
bity 0 do 4 - wybór 1 z 32 banków 16 KB RAM VBXE

MB_ANTIC
bit 7 - 1 = ANTIC widzi RAM VBXE w obszarze MEMAC B
0 = ANTIC widzi tam normalny RAM komputera
bit 6 - zarezerwowany
bit 5 - zarezerwowany
bity 0 do 4 - wybór 1 z 32 banków 16 KB RAM VBXE

MAPOWANIE ADRESÓW

Adres RAM w VBXE jest 19-to bitowy (512KB). Złożenie numeru banku oraz adresu w ramach okna MEMAC A/B pozwala obliczyć adres docelowy operacji na pamięci. Jest to ważne, ponieważ Rdzeń wymaga czasami podawania bezwzględnych adresów 19-to bitowych więc musimy wiedzieć gdzie dokładnie są nasze dane.

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

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

Przykład: Zapisujemy do pamięci VBXE pod adres 0x12800, używamy okna MEMAC A:

NR_BANKU_A[5:0] = 0x12800 / 0x2000 = 9
adres (ATARI) = 0x2000 + (0x12800 & 0x1fff) = 0x2000 + 0x800 = 0x2800

Przykład: Zapisujemy do pamięci VBXE pod adres 0x12800, używamy okna MEMAC B:

NR_BANKU_B[4:0] = 0x12800 / 0x4000 = 4
adres (ATARI) = 0x4000 + (0x12800 & 0x3fff) = 0x4000 + 0x2800 = 0x6800

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 17 bajtów:

Nr bajtu	nazwa	opis
1	source_adr0	bity 0 ... 7 adresu danych źródłowych blittera
2	source_adr1	bity 8 ... 15 adresu danych źródłowych blittera
3	source_adr2	bity 16 ... 18 adresu danych źródłowych blittera
4	source_step0	bity 0 ... 7 odległości pomiędzy liniami danych źródłowych
5	source_step1	bity 8 ... 11 odległości pomiędzy liniami danych źródłowych
6	dest_adr0	bity 0 ... 7 adresu obszaru docelowego blittera
7	dest_adr1	bity 8 ... 15 adresu obszaru docelowego blittera
8	dest_adr2	bity 16 ... 18 adresu obszaru docelowego blittera
9	dest_step0	bity 0 ... 7 odległości pomiędzy liniami obszaru docelowego
10	dest_step1	bity 8 ... 11 odległości pomiędzy liniami obszaru docelowego
11	blt_width0	bity 0 ... 7 szerokości kopiowanego obiektu
12	blt_width1	bit 8 szerokości kopiowanego obiektu
13	blt_height	bity 0 ... 7 wysokości kopiowanego obiektu
14	blt_and_mask	maska AND źródła danych
15	blt_or_mask	maska OR źródła danych
16	blt_collision_mask	maska AND wykrywania kolizji
17	blt_control	dodatkowe informacje (patrz opis)

source_adr

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

source_step

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

source step = 0...4095

dest_adr

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

dest_step

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

dest step = 0...4095

blt_width

Szerokość kopiowanego obiektu.

blt_width = 0...511 co odpowiada szerokości 1...512 punktów.

blt_height

Wysokość kopiowanego obiektu.

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_or_mask

Ustawianie bitów danych źródłowych:

source'' = source' OR blt_or_mask

Tej operacji poddawany jest każdy bajt danych źródłowych.

blt_collision_mask

Maska ukrywania kolizji.
Wykrywanie kolizji następuje w trybach 1,2,3,4,5 pracy blittera (patrz opis blt_control).
Warunek wykrycia kolizji jest następujący:

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

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

blt_control

Bajt kontrolujący tryb pracy i zachowanie blittera.

b7	b6	b5	b4	b3	b2	b1	b0
DY	DX	SY	SX	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_or_mask;
	dest' = source";
	WriteDest(dest');

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 jest ona różna od zera 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_or_mask;
	if (source" != 0)
	{
	dest = ReadDest();
	if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	dest' = source";
	WriteDest(dest');
	}

MODE	opis

2	Wartość zapisywana dest' jest sumą arytmetyczną danych source'' oraz wartości dotychczasowej dest.
	
	source = ReadSource();
	source' = source & blt_and_mask;
	source'' = source' | blt_or_mask;
	if (source'' != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) 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_or_mask;
	if (source'' != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	  dest' = dest | source'';
	  WriteDest(dest');
	}

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_or_mask;
	dest = ReadDest();
	if (source'' != 0 && (dest & blt_collision_mask))
	{
	  BLT_COLLISION_CODE = dest;
	}
	dest' = dest & source'';
	WriteDest(dest');

MODE	opis

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_or_mask;
	if (source'' != 0)
	{
	  dest = ReadDest();
	  if (dest & blt_collision_mask) BLT_COLLISION_CODE = dest;
	  dest' = dest ^ source'';
	  WriteDest(dest');
	}

6	nieużywane / zarezerwowane
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)

Bity SX, SY, DX, DY.

Dla jasności poniższego opisu należy dodać, że przed startem pracy blittera wykonywana jest operacja:

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

oraz, że odczyt i zapis danych następują z i do adresów oznaczonych ' (prim).

SX - sposób modyfikacji adresu danych źródłowych w ramach poziomej linii danych

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

SY - sposób modyfikacji adresu danych źródłowych po zakończeniu poziomej linii danych

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

Po czym następuje operacja:

source_adr' = source_adr;

DX - sposób modyfikacji adresu danych źródłowych w ramach poziomej linii danych

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

DY - sposób modyfikacji adresu danych źródłowych po zakończeniu poziomej linii danych

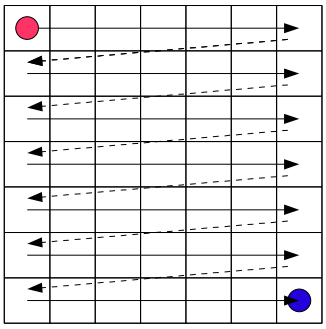
DY = 0 : dest_adr = dest_adr + dest_step
DY = 1 : dest_adr = dest_adr - dest_step

Po czym następuje operacja:

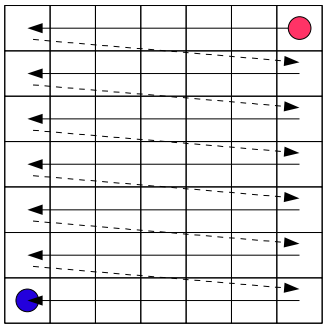
dest_adr' = dest_adr;

Poniższy rysunek ukazuje sposób modyfikacji adresów source / destination w czasie pracy blittera.

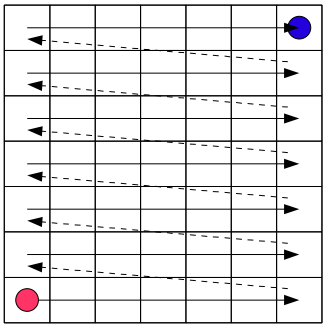
S(D)X=0
S(D)Y=0



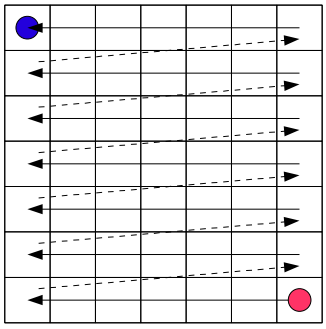
S(D)X=1
S(D)Y=0



S(D)X=0
S(D)Y=1



S(D)X=1
S(D)Y=1



- START (= source_adr lub dest_adr w BCB)
- STOP - ostatni bajt danych

blt_width = 6 (7 punktów)
blt_height = 6 (7 linii)
1 kratka = 1 bajt
Adresy kolejnych wierszy oddalone od siebie
o wartość source_step lub dest_step

Manipulując bitami SX, SY, DX i DY 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.

Blitter i stałe dane źródłowe

Jeżeli wynik operacji:

(~blt_and_mask | blt_or_mask)

jest równy 0xFF, wówczas dane źródłowe są STAŁE - nie zależą od odczytanych z obszaru źródłowego i wartość danych źródłowych wynosi dokładnie blt_or_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

Adres	Zapis	Odczyt
Dx40	VIDEO_CONTROL	CORE_VERSION (= 0x10)
Dx41	XDL_ADR0 bity 0 ... 7	255
Dx42	XDL_ADR1 bity 8 ... 15	255
Dx43	XDL_ADR2 bity 16 ... 18	255
Dx44	MSEL	255
Dx45	MB0	255
Dx46	MB1	255
Dx47	MB2	255
Dx48	MB3	255
Dx49	-	255
Dx4A	-	255
Dx4B	-	255
Dx4C	MA_CPU	255
Dx4D	MA_ANTIC	255
Dx4E	MB_CPU	255
Dx4F	MB_ANTIC	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	-	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.

VIDEO_CONTROL

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	xcolor	xdl_enabled
-	-	-	-	-	-	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 hires (graficzny / tekstowy) uniezależnij kolor zapalonego piksela od koloru tła.

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

xdl_enable:

1 = włącz przetwarzanie XDL począwszy od najbliższej synchronizacji pionowej.

0 = wyłącz przetwarzanie XDL począwszy od najbliższej synchronizacji pionowej.

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

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.

MSEL

b7	b6	b5	b4	b3	b2	b1	b0
SEL1	SEL0	Funkcja zależna od SEL1 i SEL0					
w-0	w-0	w-x	w-x	w-x	w-x	w-x	w-x

SEL1	SEL0	działanie
0	0	nic nie rób
0	1	zarezerwowane / zabronione
1	0	wykonaj MSEL / PRIORMAP
1	1	wykonaj MSEL / RGB

Funkcje pozostałych bitów: patrz opis mechanizmów MSEL/RGB oraz MSEL/PRIORMAP.

MB0, MB1, MB2, MB3

Patrz opis mechanizmów MSEL/RGB oraz MSEL/PRIORMAP.

MA_CPU

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	nr banku 0 ... 63					
w-0	-	w-x	w-x	w-x	w-x	w-x	w-x

MEMAC. Nr banku pamięci VBXE w oknie 0x2000 - 0x3FFF widziany przez CPU gdy ENA = 1. Gdy ENA = 0 wówczas CPU widzi tutaj normalną pamięć ATARI.

MA_ANTIC

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	nr banku 0 ... 63					
w-0	-	w-x	w-x	w-x	w-x	w-x	w-x

MEMAC. Nr banku pamięci VBXE w oknie 0x2000 - 0x3FFF widziany przez ANTIC gdy ENA = 1. Gdy ENA = 0 wówczas ANTIC widzi tutaj normalną pamięć ATARI.

MB_CPU

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	-	nr banku 0 ... 31				
w-0	-	-	w-x	w-x	w-x	w-x	w-x

MEMAC. Nr banku pamięci VBXE w oknie 0x4000 - 0x7FFF widziany przez CPU gdy ENA = 1. Gdy ENA = 0 wówczas CPU widzi tutaj normalną pamięć ATARI.

MB_ANTIC

b7	b6	b5	b4	b3	b2	b1	b0
ENA	-	-	nr banku 0 ... 31				
w-0	-	-	w-x	w-x	w-x	w-x	w-x

MEMAC. Nr banku pamięci VBXE w oknie 0x4000 - 0x7FFF widziany przez ANTIC gdy ENA = 1. Gdy ENA = 0 wówczas ANTIC widzi tutaj normalną pamięć ATARI.

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.

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.

CORE_VERSION

Wersja rdzenia, BCD. 0x10 = v1.0.

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.

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_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.

HISTORIA WERSJI

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.