XGI Volari V3XE(T)

2D Programming Guide

Document No.: XG42-301-001

Document Ver.: 1.0

Index

Introduction

This programming guide is a guide for the XGI Volari serious chips (XG40, XG42) programming. The goal of this guide is to support programmers a document to finish their works.

PART. I. Graphic Initialization

Chap 1. Set Mode

To set mode with VGA BIOS support should follow the steps listed below:

SET THE OUTPUT SCRATCH REGISTER TO IDENTIFY WHICH DEVICE YOU WANT TO OUTPUT.

Set the frame rate index into scratch register to identify the target frame rate.

Call INT 10h function 0 to set mode.

Setup CRT1 screen display pitch if you set mode to output on CRT1.

Setup CRT2 screen display pitch if you set mode to output on CRT2.

Setup CRT1 display start address if you set mode to output on CRT1.

Setup CRT2 display start address if you set mode to output on CRT2.

Enable frame buffer linear access.

1-1 Scratch Register (The part is modifying)

The scratch registers of XGI Volari Family display chip are reserved on CR30-CR38. To programming these registers, you should use the following operations:

Operation	Step

Read Register	mov dx, CRTCIOBasePort;(*3D4 or relocated*)
	mov al, SCRATCH_REGISTER_ID
	out dx, al
	inc dx
	in al, dx ; al = the scratch register content

Write Register	mov dx, CRTCIOBasePort
	mov al, SCRATCH_REGISTER_ID
	out dx, al
	inc dx
	mov al, Value
	out dx, al

The scratch register definition is listed below:

Register	Bit	Description
CR30	D[0]	Disable/Enable (0/1) Simultaneous View
	D[1]	CRT1/CRT2 (0/1) Mode Select
	D[2]	Set VB Output to Composite
	D[3]	Set VB Output to S-Video
	D[4]	Set VB Output to SCART
	D[5]	Set VB Output to LCD
	D[6]	Set VB Output to CRT2
	D[7]	Set VB Output to Hi-Vision TV
CR31	D[0]	NTSC/PAL (0/1) Mode Select
	D[1]	VB at Lock-Mode (only used by BIOS)
	D[2]	Enable/Disable (0/1) Simultaneously Display at Lock-Mode
	D[3]	External Set Mode Function (only used by BIOS)
	D[4]	Enable/Disable (0/1) Load Bridge-CRT2 RAMDAC Data

	D[5]	Enable/Disable (0/1) VB Output (when CR30[1:0]=0)
	D[6]	DOS/Driver(0/1) Mode Select
	D[7]	'Hot Key' Display Control Select : VBIOS/Driver(0/1)
CR32	D[0]	VB Connected With Composite
	D[1]	VB Connected With S-Video
	D[2]	VB Connected With SCART
	D[3]	VB Connected With LCD
	D[4]	VB Connected With CRT2
	D[5]	CRT1 Monitor is Connected
	D[6]	VB Connected With Hi-Vision TV
	D[7]	VB Connected With DVI Combo Connector
CR33	D[3:0]	CRT1 Refresh Rate
	D[7:4]	CRT2 Refresh Rate
CR34	D[7:0]	Bridge-CRT2 Mode ID (only used by BIOS)
CR35	D[1:0]	SR16[1:0] Number of refreshes to do with in 16ms
	D[2]	OSD Event Identify Bit (OSD Event Queue = 1)
		0 : No OSD Event Occur
		1 : New OSD Event Occur
	D[3]	Trumpion/Chrontel TV Full/Window DOS Identify Bit
		0 : Full Screen DOS
		1 : Window Screen DOS
	D[7:6]	SR16[7:6] RQ capacity
CR36	D[3-0]	LCD Panel
		0001 :800x600 Panel
		0010 :1024x768 Panel
		0011 :1280x1024 Panel
	D[7-4]	LCD Type
CR37	D[0]	Set 24-bit/18-bit(0/1) RGB to LVDS/TMDS transmitter
	D[3:1]	External Chip Select
		001 : SiS301
		010 : LVDS
		011 : LVDS + Trumpion Scaling Chip
	D[4]	Expanding/Non-expanding(0/1) Display (LVDS only)
	D[4]	UnderScan/OverScan(0/1) Display (Chrontel TV only)
	D[5]	LCD Polarity Select (301 only)
		0 : VESA DMT Standard
		1 : EDID 2.x Define
	D[6]	LCD Horizontal Polarity Select (301/LVDS)
		0 : High Active
		1 : Low Active
	D[7]	LCD Vertical Polarity Select (301/LVDS)
		0 : High Active
		1 : Low Active

Note: VB1 means the VB chip connects to XGI Volari Family CRT1 output. The VB2 means the VB chip connects to XGI Volari Family CRT2 output. VB1A means the channel A of VB1, VB1B means the channel B of VB1, VB2A means the channel A of VB2 (it always connects to CRT1), and VB2B means the channel B of VB2.

The video bridge connect types are listed below:

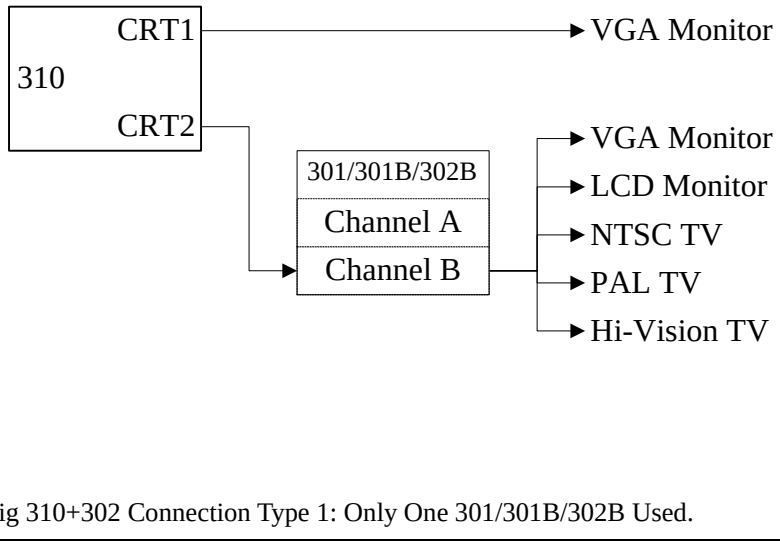


Fig 310+302 Connection Type 1: Only One 301/301B/302B Used.

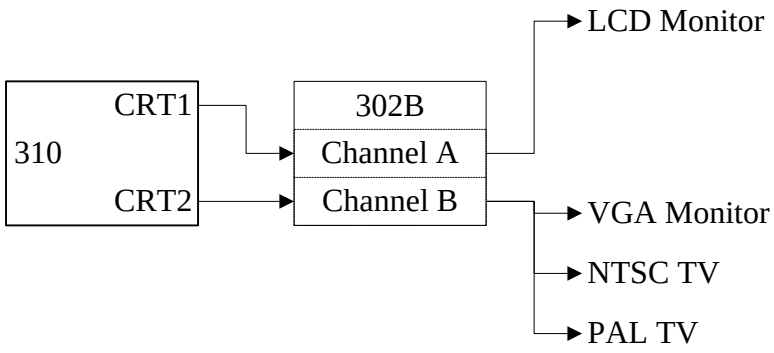


Fig 310+301/302 Connection Type 2:

Only One 302B Used, Support LCD+TV Output.

i.) To set up output device with scratch registers

The detail programming method will generate after VBIOS team defined the 301C/302LV scratch register specification.

ii.) To setup the refresh rate

To setup refresh rate is easy on XGI Volari Family scratch register setting. To set the CRT1 frame rate index for the target resolution on CR33 D[3:0], and set the CRT2 frame rate index for the target resolution on CR33 D[7:4], do not care about the connect output device.

The refresh rate indexes without modified by OEM are list in the table below:

resolution\ID	1	2	3	4	5	6	7	8
640x480	60	72	75	85	100	120	160	200
800x600	56	60	72	75	85	100	120	160
1024x768	87(i)	60	70	75	85	100	120	
1152x864	60	75	85					
1280x960	60	75	85					
1280x1024	87(i)	60	75	85				
1600x1200	60	65	70	75	85			
1920x1440	60	65	70	75	85			
2048x1536	60	65	70	75	85			
800x480	60							
1024x576	60							
1280x720	60							

1-2 BIOS Set Mode

The VBIOS support int 10h standard interface to support video service. The sub-function ah=0 is the function to set mode. The usage is as below;

```
mov  ah, 0
mov  al, ModeID (look up the VBIOS mode table)
int  10h
```

If you set the output to CRT1, you can use INT 10h sub-function ah = 0Fh to get the current mode ID.

```
mov  ah, 0Fh
int  10h
; al = ModeID (look up the VBIOS mode table) of CRT1
```

If you set the output to CRT2, you can read back the value of CR36 to check the current mode ID.

1-3 Set CRT1 Color Depth

Whenever we set the mode via BIOS (or call the routine to set mode), we should set the color depth as the table descripted:

Color Depth	GR5 D[6]	SR6 D[4:2]	Notes
8 bits per pixel	1	000	
15 bits per pixel (x555 format)	0	001	not suggested
16 bits per pixel (565 format)	0	010	
24 bits per pixel (888 format)	0	011	not suggested
32 bits per pixel (x888 format)	0	100	

The sample code is listed below:

```
.if wBpp == 8
        mov  dx, 3ceh
        mov  al, 05h
        out  dx, al
        inc  dx
        in   al, dx
        or   al, 1 shl 6
        out  dx, al
        mov  dx, 3c4h
        mov  al, 06h
        out  dx, al
        inc  dx
        out  dx, al
        inc  dx
        in   al, dx
        and  al, 11100011b
        out  dx, al
.elseif wBpp == 16
        mov  dx, 3ceh
        mov  al, 05h
        out  dx, al
        inc  dx
        in   al, dx
        and  al, 0ffh - (1 shl 6)
        out  dx, al
        mov  dx, 3c4h
        mov  al, 06h
        out  dx, al
        inc  dx
        out  dx, al
        inc  dx
        in   al, dx
        and  al, 11101011b
        out  dx, al
.elseif wBpp == 32
        mov  dx, 3ceh
        mov  al, 05h
        out  dx, al
        inc  dx
        in   al, dx
        and  al, 0ffh - (1 shl 6)
        out  dx, al
        mov  dx, 3c4h
        mov  al, 06h
        out  dx, al
        inc  dx
        out  dx, al
        inc  dx
        in   al, dx
        and  al, 11110011b
        out  dx, al
.endif
```

1-4 Set CRT1 Display Line Width

The display line width is SR10h. To set the register for adjust the hardware line width. The description of SR 10h is listed below:

SR10 Display line width register

Index: 0x10

Default value: 00h

D[7:0]Display line width [7:0] in unit of 64 bytes.

To set the value as

SR10h = ((ScreenWidth*wBpp)/64)+1

```
mov   dx, 3c4h
mov   al, 10h
out    dx, al
movzx       eax, wScreenWidth
movzx       ecx, wBpp
imul  eax, ecx
mov   dx, 3c5h
shr    eax, 6
inc    al
out    dx, al
```

1-5 Set CRT1 Screen Pitch

The term "screen pitch" is the byte count of the two vertical neighbor pixel on screen. For special implementation, the screen pitch may not be equal to the size (screen width)×(pixel size).

To set screen pitch in the register of XGI Volari Family is in the unit "character", the character equal to 8 pixel for all enhance mode.

There are two registers related to the CRT1 screen pitch:

Register	Description
CR13	SCREEN OFFSET (Screen pitch) D[7:0] Screen pitch D[7:0]
SR 0E	Extended CRT pitch register D[7:5] Horizontal retrace skew adjustment [2:0] (in unit DCLK) D[4] Reserved

	D[3:0]	Screen pitch D[11:8]

Therefore, on XGI Volari Family chip maximum screen pitch cannot be over 16384 bytes.

For example if we want to set the screen pitch as 2560 bytes (if it is 1280×1024×16Bpp mode), we use the step listed below:

```
mov  cx, wScreenPitch
shr  cx, 3  ; in the unit 8 bytes
mov  dx, wCRTCIOBase
mov  al, 13h
out  dx, al
inc  dx
mov  al, cl
out  dx, al
mov  dx, wExtendedIOBase
mov  al, 0eh
out  dx, al
inc  dx
in   al, dx
and  al, 0F0h
and  ch, 0Fh
or   al, ch
out  dx, al
```

Notice: If you want to set the screen pitch under interlaced mode, you should set it to be double of the width×pixel size. Because under interlaced mode, the next CRT scan line will be at the next next position. To double the pitch, you can see third scan line on the second CRT scan time, and the position is correct.

1-6 Set CRT2 Screen Pitch

To setup the CRT2 screen pitch, the pitch register, you should programming the CRT2/LCD registers of XGI Volari Family. The registers IO base is on XXXX+4, XXXX means the relocated IO base on PCI configuration space offset 18h. There are two associated registers to set the CRT2 screen pitch:

Register	Description	
Register 7 Access Memory Line offset	D[7:0]	CRT2 Screen pitch [7:0]
Register 9 Overflow Register	D[7:4] D[3:0]	CRT2 horizontal total [11:8] CRT2 Screen pitch [7:0]

For the previous sample on CRT2 case, the setting is listed below:

```
mov  cx, wScreenPitch
shr  cx, 3  ; in the unit 8 bytes
mov  dx, wGMCRT2LCDIOBase
mov  al, 7h
out  dx, al
inc  dx
mov  al, cl
```

```
out   dx, al
mov   dx, wGMCRT2LCDIOBase
mov   al, 09h
out   dx, al
inc   dx
in    al, dx
and   al, 0F0h
and   ch, 0Fh
or    al, ch
out   dx, al
```

1-7 Set CRT1 Screen Starting Address

The starting address is the first byte displayed on screen of total frame buffer. There are two types of display, normal (SR0F D[2]=0) or fast flip (SR0F D[2]=1).

i.) CRT1 Screen Starting Address on Normal Mode

On normal mode, there are four associated registers for the starting address setting:

Register	Description
CR 0D Screen Start Address Low	D[7:0] Screen Start Address Bit [7:0] (in unit DWORD)
CR 0C Screen Start Address High	D[7:0] Screen Start Address Bit [15:8] (in unit DWORD)
SR 0D Screen Start Address overflow	D[7:0] Screen Start Address Bit [23:16] (in unit DWORD)
SR 37 Extended CRT starting address	D[0] Screen Start Address Bit [24] (in unit DWORD)

For example, to show the frame buffer starting at 0x433800 on CRT1, you should follow the steps below:

```
mov   ecx, 433800h
shr   ecx, 2; in unit dword
mov   dx, wCRTCIOBase
mov   al, 0dh
out   dx, al
inc   dx
mov   al, cl
out   dx, al ; program CR 0D
dec   dx
mov   al, 0ch
out   dx, al
inc   dx
mov   al, ch
out   dx, al ; program CR 0C
shr   eax, 16
mov   dx, wExtendedIOBase
mov   al, 0dh
out   dx, al
inc   dx
mov   al, cl
out   dx, al ; program SR 0D
dec   dx
mov   al, 37h
```

```
out   dx, al
inc   dx
in    al, dx
and   al, 0FEh
or    al, ch
out   dx, al ; program SR 37 D[0]
```

ii.) CRT1 Screen Starting Address on Flip Mode

On flip mode, the starting address is controlled by flip addressing registers. The associated starting address registers and control registers are below:

Register	Description
SRF CRT misc. control register	Default value: 00h D7 Enable CRT1 low resolution modes 0: disable 1: enable D6 PCI address decoding enhanced mode enable 0: disable 1: enable (recommend for 1024x768 or higher 16 color modes) D5 Reserved D4 Reserved D3 Disable line compare 0: enable 1: disable D2 Fast change mode enable (SR2F-D5) 0: disable 1: enable D1 CRT1 software programming Nbuf flip enable 0: disable (use SR37, SRD, CRC, CRD) 1: enable (use MMIO8540-MMIO855F) D0 Enable CRT gen-lock with external video controller 0: disable 1: enable

SR3E CRT1 software flipping register I	Default value: 00h D[7:4] CRT1 fast left page flip starting address memory location selection [3:0] Write: 0000: select MMIO8540 as left starting address 0001: select MMIO8544 as left starting address 0010: select MMIO8548 as left starting address 0011: select MMIO854C as left starting address 0100: select MMIO8550 as left starting address 0101: select MMIO8554 as left starting address 0110: select MMIO8558 as left starting address 0111: select MMIO855C as left starting address 1xxx: reserved The starting address selection was written into this register as next expected left starting address selection. D[3:0] CRT1 fast right page flip starting address memory location selection [3:0] Write: 0000: select MMIO8540 as right starting address 0001: select MMIO8544 as right starting address 0010: select MMIO8548 as right starting address 0011: select MMIO854C as right starting address 0100: select MMIO8550 as right starting address 0101: select MMIO8554 as right starting address 0110: select MMIO8558 as right starting address 0111: select MMIO855C as right starting address 1xxx: reserved The starting address selection was written into this register as next expected right starting address selection. Read: 0000: current display MMIO8540 as right starting address 0001: current display MMIO8544 as right starting address 0010: current display MMIO8548 as right starting address 0011: current display MMIO854C as right starting address 0100: current display MMIO8550 as right starting address 0101: current display MMIO8554 as right starting address 0110: current display MMIO8558 as right starting address 0111: current display MMIO855C as right starting address 1000: current display MMIO8540 as left starting address 1001: current display MMIO8544 as left starting address 1010: current display MMIO8548 as left starting address 1011: current display MMIO854C as left starting address 1100: current display MMIO8550 as left starting address 1101: current display MMIO8554 as left starting address 1110: current display MMIO8558 as left starting address 1111: current display MMIO855C as left starting address The read data will be triggered by write a password to SR5 register as current starting address selection.

CR55 CRT2 queue flipping register	Default value: 00h D7 CRT2 queue flipping enable 0: disable 1: enable D6 CRT2 queue stereo flipping enable 0: disable 1: enable D5 CRT2 queue flipping waiting for HR 0: waiting for VR 1: waiting for HR D4 Selection of MM854X, MM855X as starting address for CRT1/CRT2 0: for CRT1 1: for CRT2 D3 CRT1 queue flipping enable 0: disable 1: enable D2 CRT1 queue stereo flipping enable 0: disable 1: enable D1 CRT1 queue flipping waiting for HR 0: waiting for VR 1: waiting for HR D0 CRT1/CRT2 double frames stereo enable 0: single frame (left or right) 1: double frames (left and right)
MMIO8540 Fast left page flip starting address 1 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast left page flip starting address 1 bit [24:0] If CR55[4]=0, MMIO854x and MMIO855x defined for CRT1 If CR55[4]=1, MMIO854x and MMIO855x defined for CRT2
MMIO8544 Fast right page flip starting address 1 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast right page flip starting address 1 bit [24:0]
MMIO8548 Fast left page flip starting address 2 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast left page flip starting address 2 bit [24:0]
MMIO854C Fast right page flip starting address 2 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast right page flip starting address 2 bit [24:0]
MMIO8550 Fast left page flip starting address 3 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast left page flip starting address 3 bit [24:0]
MMIO8554 Fast right page flip starting address 3 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast right page flip starting address 3 bit [24:0]
MMIO8558 Fast left page flip starting address 4 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast left page flip starting address 4 bit [24:0]

MMIO855C Fast right page flip starting address 4 register	Default value: xxxxxxxxh D[31:25] reserved D[24:0] Fast right page flip starting address 4 bit [24:0]

There are two types of page flip, software flip and queue flip. The queue file type is an automatic page flip control by hardware, and it will be described on later section.

Type software flipping can separate into two different programming styles: normal style and stereo style.

The programming method in normal style:

Always use the left page flip starting address registers.

Set CR55 D[4] zero to identify the target is CRT1. There are two groups of starting address registers for CRT1 and CRT2. They use the same MMIO ports, thus you must choose which CRT you want to update before you update the starting address registers.

Write the starting address into the selected starting address register.

Set SR3E D[7:4] to identify which page flip you select.

Set SR0F D[2] 1 to enable the software page flip.

Note: if the queuing flip (CR55 D[3]) enabled, the software page flip will be disabled for the priority issue.

For example: To show the frame started with 0x433800 on CRT1 screen with page flip register 8554.

```
mov  dx, wCRTCIOBase
mov  al, 55h
out  dx, al
inc  dx
in   al, dx
and  al, (0FFh - 10h)
out  dx, al ; set CR55 D[4] zero, choose CRT1.
mov  eax, 433800h
shr  eax, 2; in unit DWORD
mov  es, wMMIOBase
mov  es:[8554h], eax
mov  dx, wExtendedIOBase
mov  al, 3Eh
out  dx, al
inc  dx
in   al, dx
and  al, 0Fh
or   al, (0101b shl 4)   ; set the 8554
out  dx, al
dec  dx
mov  al, 0fh
out  dx, al
inc  dx
in   al, dx
or   al, 2   ; enable page flip
out  dx, al
```

The term "stereo" means to display two image/video simultaneously, let user feel the effect of stereo. The stereo effect in XGI Volari Family can be implemented with page flip two different frame interlaced, and separate them into each eye input with special device (such as stereo glasses).

The programming method in stereo style:

Set CR55 D[4] zero to identify the target is CRT1.

Set CR55 D[0] 1 to enable double frame (stereo) mode.

Write the left frame starting address into the selected left frame starting address register.

Write the right frame starting address into the selected right frame starting address register.

Set SR3E D[7:4] to identify which left page flip you select.

Set SR3E D[3:0] to identify which right page flip you select.

Set SR0F D[2] 1 to enable the software page flip.

For example, the left frame is at 0x433800, right frame is at 0x533760, then we program the stereo mode with left frame register 8540 and right register 855C with the following steps:

```
mov  dx, wCRTCIOBase
mov  al, 55h
out  dx, al
inc  dx
in   al, dx
and  al, (0ffh-10h-1)
or   al, 1
out  dx, al ; set register group to CRT1, enable stereo mode
mov  es, wMMIOBase
mov  ecx, 433800h
mov  ebx, 533760h
shr  ecx, 2; in DWORD mode
shr  ebx, 2
mov  es:[8540h], ecx
mov  es:[855ch], ebx
mov  dx, wExtendedIOBase
mov  al, 3Eh
out  dx, al
inc  dx
mov  al, (0000b shl 4 + 0111b)      ; left on 8540, right on 855C
out  dx, al
dec  dx
mov  al, 0Fh
out  dx, al
inc  dx
in   al, dx
or   al, 2  ; enable fast flip
out  dx, al
```

1-8 Set CRT2 Screen Starting Address

As the CRT1 starting address, CRT2 screen starting address mode also has two types, normal mode and fast page flip mode. The switch of CRT2 normal/flip mode is in Part I register 0x24 D[6] as the SR0F D[1] action under CRT1 case.

i.) CRT2 Screen Starting Address on Normal Mode

In normal mode, the associated register is listed below:

Register	Description
Part I Register 02 Memory start address[24], PCI Bus Clock and FIFO Threshold Low	Register Type:Read/Write Read/Write Port:xx05, Index 02h Default : 00h D[7] CRT2 memory read start address bit[24] D[6] VB programming clock(VBHCLK) 0 : PCI clock divide by 2. 1 : PCI clock divide by 4. D[5:0] BRTHRLT[5:0], CRT2 FIFO threshold low.
Part I Register 04 Access Memory Starting Address High	Register Type:Read/Write Read/Write Port:xx05, Index 04h Default :xxh D[7:0]CRT2 memory read start address bits[23:16]
Part I Register 05 Access Memory Starting Address Median	Register Type : Read/Write Read/Write Port : xx05, Index 05h Default : xxh D[7:0] CRT2 memory read start address bits[15:8]
Part I Register 06 Access Memory Starting Address Low	Register Type : Read/Write Read/Write Port: xx05, Index 06h Default : xxh D[7:0]CRT2 memory read start address bits[7:0]

For example, you can use the following steps to set CRT2 starting address on 0x433800:

```
mov  dx, wRelocatedIOBase
add  dx, 4  ; xx04/xx05, Part I register
mov  ecx, 433800h
shr  ecx, 2
mov  al, 6
out  dx, al
inc  dx
mov  al, cl
out  dx, al ; set address [7:0]
dec  dx
mov  al, 5
out  dx, al
inc  dx
mov  al, ch
out  dx, al ; set address [15:8]
shr  ecx, 16
mov  al, 4
out  dx, al
inc  dx
mov  al, cl
out  dx, al ; set address [23:16]
```

```
dec   dx
mov   al, 2
out   dx, al
inc   dx
in    al, dx
shl   ch, 7
or    al, ch
out   dx, al ; set address [24]
```

ii.) CRT2 Screen Starting Address on Flip Mode

The CRT2 flip mode have also two types, queue flip and software page flip, as the CRT1 case. The software page flip can also separate into two types, single frame mode and double frame. The associated registers are listed on the section of CRT1 page flip description.

To setup single frame, you must follow the steps:

Set CR55 D[4] as 1 to identify CRT2 setting.

Set CR55 D[0] as zero to identify single frame mode.

Write the starting address into one of those page flip address registers.

Set SR3E D[7:4] to identify which register should be flip up.

Set Part I register 0x24 D[6] to enable CRT2 page flip.

For example, if we want to flip the frame at 0x433800 to CRT2 with register 854C, following the steps:

```
mov   dx, wCRTCIOBase
mov   al, 55h
out   dx, al
inc   dx
in    al, dx
and   al, (0FFh-10h-1)
or    al, 10h       ; CRT 2 flip address, single frame (normal mode)
out   dx, al
mov   ecx, 433800h
shr   ecx, 2; in DWORD unit
mov   es, wMMIOBase
mov   es:[854C], ecx
mov   dx, wExtendedIOBase
mov   al, 3Eh
out   dx, al
inc   dx
in    al, dx
and   al, 0fh
or    al, (011b shl 4)    ; set CRT2 flip register 854C
out   dx, al
mov   dx, wRelocatedIOBase
add   dx, 4 ; Part I Register Base
mov   al, 24
out   dx, al
inc   dx
in    al, dx
or    al, 1 shl 6   ; enable CRT2 software flip
```

```
out    dx, al
```

To setup double frame (stereo) mode, you must follow the steps:

Set CR55 D[4] as 1 to identify CRT2 setting.

Set CR55 D[0] as 1 to identify double frame mode.

Write the left frame starting address into one of those left page flip address registers.

Write the right frame starting address into another of those right page flip address registers.

Set SR3E D[7:4] to identify which register should be flip as left frame.

Set SR3E D[3:0] to identify which register should be flip as right frame.

Set Part I register 0x24 D[6] to enable CRT2 page flip.

For example, if we want to flip the left frame at 0x433800 to CRT2 with register 854C and the right frame at 0x538880 with register 8550, follow the steps:

```
mov  dx, wCRTCIOBase
mov  al, 55h
out  dx, al
inc  dx
in   al, dx
and  al, (0FFh-10h-1)
or   al, 11h       ; CRT 2 flip address, double frame (stereo) mode
out  dx, al
mov  ecx, 433800h
shr  ecx, 2; in DWORD unit
mov  ebx, 538880h
shr  ebx, 2
mov  es, wMMIOBase
mov  es:[854C], ecx
mov  es:[8550], ebx
mov  dx, wExtendedIOBase
mov  al, 3Eh
out  dx, al
inc  dx
mov  al, (011b shl 4 + 110b)  ; set CRT2 left page at 854C and right page at 8550
out  dx, al
mov  dx, wRelocatedIOBase
add  dx, 4  ; Part I Register Base
mov  al, 24
out  dx, al
inc  dx
in   al, dx
or   al, 1 shl 6   ; enable CRT2 software flip
out  dx, al
```

Note: In general, page flip will not setup under set mode phase. The discussion in this section just to describe how to display different frame buffer content.

1-9 Enable Linear Addressing

To enable linear addressing mode, programming can access all of the frame buffer of adapter without switch bank. The register about linear addressing mode is listed below:

Register	Description

SR 20 PCI address decoder setting register	Default value: 20h D7 PCI linear address mode enable 0: disable 1: enable D6 reserved D5 Relocated VGA I/O port decoding enable 0: disable 1: enable D4 Standard VGA I/O port decoding disable 0: enable 1: disable D3 Enable video playback MMIO address decoding 0: disable 1: enable D2 Disable standard memory address (A000-Bfff) access 0: enable 1: disable D1 reserved D0 Memory mapped I/O enable 0: disable 1: enable

Chap 2. Palette And Gamma Correction

The term "palette" is a table to translate color index to true RGB color value. In our VGA chip, there are two color mode: indexed color mode and direct color mode. Under indexed color mode, the color format in frame buffer is an index of palette and display hardware will translate the color index into the associated color which stored in the "palette registers", the output the physical color to display device. Under direct color mode, the color format in frame buffer is the real color RGB element, and display hardware will output the color element to display device directly.

We implements the 256 colors mode as indexed color mode, and there are 768 bytes of palette registers to translate the color index into real color. The "High-Color" mode (64K colors mode with color depth 16bits) and "True-Color" mode (16.7M colors mode with color depth 32bits) are implemented as direct color mode.

The term "Gamma Correction" is an operation to translate the "logical color" to the physical visual color. According to differences of output devices, the same color value in frame buffer will be represented into different visual color. In order to manage the color between different output devices, "gamma correction" is given to translate the "logical color value" (the color in frame buffer or in palette) into visual color. Assume the density function of the output device is F(x), x is the logical color value, and $F(x) \neq x$, we can translate the color value with the function $G(x) = F^{-1}(x)$, then the output result will become $F(F^{-1}(x))=x$. The given function G(x) is the function for gamma correction.

The palette registers is useless under high color mode and true color mode on XGI Volari Family, thus we support the gamma correction function with the palette registers under high-color and true color. You can stored your red/green/blue gamma correction function into the red/green/blue field of the palette registers.

Under 256 colors mode, the palette register must reserve for the color translation. To translate the colors before they are stored into palette registers can implement the gamma correction function under 256 colors mode.

2-1 CRT1 Palette Programming For 256 Colors Mode

There are two interfaces for programming palette registers: port I/O interface and MMIO interface. However, you must modify the contains of the palette registers under the display is not active (in horizontal blanking region or vertical blanking region) to avoid the noise on display.

The initial steps to programming the XGI Volari Family CRT1 palette registers are listed below:

Turn off of gamma correction bit (SR07 D[4] = 0). The programming steps in two interface is described in the following two sections.

i.) Programming CRT1 Palette with Port I/O

The port I/O interface is the traditional interface to programming palette. There are three I/O ports about palette programming:

Name	Port	Description
Palette Read Index	3C7h/XX47h	The port to select the index of the palette register to read
Palette Write Index	3C8h/XX48h	The port to select the index of the palette register to

		write
Palette Data	3C9h/XX49h	The port to read/write the data of the selected palette register

The method to read back the color value in a given palette index is listed below:

Out the index to port 3C7h.

Read the red color value from port 3C9h.

Read the green color value from port 3C9h

Read the blue color value from port 3C9h

The method to read back the color value in a given palette index is listed below:

Out the index to port 3C8h.

Write the red color value from port 3C9h.

Write the green color value from port 3C9h

Write the blue color value from port 3C9h

The palette data should read three times from the same port to get a set of RGB colors, or write three times to the same port to put a set of RGB color.

With the interface, the RGB color have 6 bits red part, 6 bits green part, and 6 bits blue part. For example, to write the 24-bit RGB color 0x123456 to the palette index 0x08 is listed below:

```
mov  ecx, 123456h      ; set ecx as the RGB color 0x123456
mov  al, 08h
mov  dx, 3C8h
out  dx, al ; select the palette index 8 to modify
mov  dx, 3C9h
ror  ecx, 16
mov  al, cl  ; give the red color part
rol  ecx, 16
shr  al, 2   ; translate 8 bits red part into 6 bits
out  dx, al
ror  ecx, 8
mov  al, cl  ; give the green color part
rol  ecx, 8
shr  al, 2   ; translate 8 bits green part into 6 bits
out  dx, al
mov  al, cl  ; give the blue color part
shr  al, 2   ; translate 8 bits blue part into 6 bits
out  dx, al
```

The following code is an example to get the color value in index 0x4F into an 24-bit RGB color:

```
mov  al, 4Ch
mov  dx, 3C7h
out  dx, al ; select the palette index 8 to fetch
mov  dx, 3C9h
in   al, dx ; get the red color part
shl  al, 2   ; translate 6-bit red color value into 8-bit
mov  cl, al
shl  ecx, 8
```

```
in      al, dx ; get the green color part
shl     al, 2   ; translate 6-bit green color value into 8-bit
mov   cl, al
shl     ecx, 8
in      al, dx ; get the blue color part
shl     al, 2   ; translate 6-bit blue color value into 8-bit
mov   cl, al
; ecx = the 24bits RGB color
```

ii.) Programming CRT1 Palette with MMIO

The port I/O is the traditional path to programming palette. The speed of port I/O is slower than memory mapped I/O (MMIO). We implement the MMIO palette programming interface on XGI Volari Family. The register definition is defined below:

Port	Bit	Description
8570	D[31:24]	Palette R/W index
	D[23:16]	Blue [7:0]
	D[15:8]	Green [7:0]
	D[7:0]	Red [7:0]

The programming method with MMIO interface is very easy. You can just write the MMIO port 0x8570 to set the palette with the following method:

Set the index value to D[31:24].

Set the blue "8-bit" value into D[23:16].

Set the green "8-bits" value into D[15:8].

Set the red "8-bits" value into D[7:0].

For example, to write the value 0x123456 into the palette register index 0x05 with MMIO interface is shown below code piece:

```
mov   ecx, 123456h       ; set ecx as the RGB color 0x123456
mov   al, 05h
shl     eax, 8
mov   cl, al
shl     eax, 8
shr     ecx, 8
mov   al, cl
shl     eax, 8
shr     ecx, 8
mov   al, cl  ; the index in D[31:24]
                ; the blue in D[23:16]
                ; the green in D[15:8]
                ; the red in D[7:0]
mov   es, wMMIOBaseSelector
mov   dword ptr es:[8570h], eax ; output an 24 bit color into palette register.
```

To read back a palette value from palette registers should write index byte into the MMIO port 0x8573, the read back the double word value from port 0x8570. For example to read the value in index

0x4C is shown below:

```
mov  es, wMMIOBaseSelector
mov  al, 4ch
mov  byte ptr es:[8573h], al
mov  eax, dword ptr es:[8570h]
xor  ecx, ecx
mov  cl, ah
mov  ch, al
shl  ecx, 8
shr  eax, 8
mov  cl, ah ; ecx contains the 24 bits color.
```

2-2 CRT2 Palette Programming with SiS 301/302/303

With SiS 301/302/303 video bridge (VB) chip, XGI Volari Family can support the secondary output device. The palette registers of the secondary output is not on XGI Volari Family chip but in SiS 301/302/303 video bridge.

You have the only way to program the palette registers of VB: the port I/O. For SiS 301/302/303 chips, the I/O port is in the I/O mapping space of the connected VGA PCI I/O space. The I/O ports are listed below:

Port	Description
XXXX + 16h (XXXX is the relocated I/O base)	Palette Register Index (Palette Read/Palette Write)
XXXX + 16h (XXXX is the relocated I/O base)	Palette Register Data (Triple read/write to update 3 color elements)

To programming the palette registers of VB, you must enable the I/O write enable before you want to program them. The register is listed below:

Register	Field	Description
CRT2 Enable Write Register	Register Type	Read/Write
	Port	XXXX+04 = 2Fh XXXX+05
	D[0]	Enable Write Register. 1: enable write registers

For example, if you want to write RGB Color 0x123456 into CRT2 palette 0x05, you should program with the following method:

```
mov  dx, wRelocatedIOBase
add  dx, 4
mov  al, 2Fh
out  dx, al
inc  dx
in   al, dx
or   al, 1   ; enable VB register programming
mov  dx, wRelocatedIOBase
add  dx, 16h
mov  al, 05h       ; select palette 05h
out  dx, al
inc` dx
mov  al, 12h
out  dx, al ; programming red field
mov  al, 34h
out  dx, al ; programming green field
mov  al, 56h
out  dx, al ; programming blue field
```

The following sample is to read the palette index 06h value to an RGB calue in ecx:

```
mov  dx, wRelocatedIOBase
add  dx, 4
mov  al, 2Fh
out  dx, al
inc  dx
in   al, dx
or   al, 1   ; enable VB register programming
mov  dx, wRelocatedIOBase
add  dx, 16h
mov  al, 05h       ; select palette 05h
out  dx, al
inc` dx
in   al, dx ; read red part
mov  cl, al
shl  ecx, 8
in   al, dx ; read green part
mov  cl, al
shl  ecx, 8
in   al, dx ; read blue part
mov  cl, al
```

It is different to the case under CRT1 256 color palette programming, the SiS VB palette is 8 bits for each color field, thus you can not translate 8 bit color into 6 bit color format under CRT2 programming.

2-3 CRT1 Gamma Correction

If the display color mode is not palette index mode, such as high color mode (color depth is 16 bits) or true color mode (color depth is 32 bits), the palette register can be used on gamma correction. The method to write/read palette is similar to the way under 256 color mode, but before you want to write the gamma translate table into palette register, you must enable SR07 D[2] to enable the gamma function under true color mode or high color mode. If you program the palette with I/O path, you must write 8 bits for red/green/blue color into palette instead of the 6 bits value under 256 color.

The following segment is the sample to write gamma translate table into palette register with port I/O path.

```
mov  bx, 0
.repeat
        mov  dx, 3C8h
        mov  al, bl
        out   dx, al
        mov  dx, 3C9h
        mov  al, bCRT1TranslateR[bx]
        out   dx, al
        mov  al, bCRT1TranslateG[bx]
        out   dx, al
        mov  al, bCRT1TranslateB[bx]
        out   dx, al
        inc   bx
.until bx == 256
```

2-4 CRT2 Gamma Correction

As the case of CRT1 gamma correction, the CRT2 gamma correction is also implemented. If you need to enable the CRT2 gamma correction function, you must enable the CRT2 gamma function.

If the VB chip is SiS 301, you must enable the register 0Dh in part four (port is XXXX+14h and XXXX+15h), the following list the register of 301:

Enable 256 Color Graphic Mode/Gamma Function and Mode Selection Register Type : Read/Write Read/Write Port : xxxx+15, Index 0dh Default : 04h D[7] RLCDPLLSPLOW, LCD PLL Speed Low D[6] RVBCLKDRVLOW, VBCLK Output Driving 0 : VBCLK Output Driving High 1 : VBCLK Output Driving Low D[5] RCRT1TVSIMUL, Enable Simultaneously Display CRT1/TV at Standard Mode D[4] RT0Enh256C, Enable Enhance 256 Color Graphic Mode D[3] REnGamma, Enable Gamma Function D[2:0] RVBCRTMOD[2:0], Mode Selection 001 : Video Bridge Slave and CRT1 to RAMDAC 011 : Video Bridge Slave and CRT2 to RAMDAC 100 : Video Bridge Master and LCD Standard Mode 110 : Video Bridge Master and LCD Enhancement Mode 101 : Video Bridge Master and TV Standard Mode 111 : Video Bridge Master and TV Enhancement Mode

If the VB chip is SiS 302 or SiS 303, the enable bit modified into D[4], see the description of the register below:

Enable 256 Color Graphic Mode/Gamma Function and Mode Selection Register Type:Read/Write Read/Write Port:xx15, Index 0dh Default:04h

D[7] RVGAPED, CRT2 Padstall Function
D[6] RCRT1TVSIMUL, Enable Simultaneously Display CRT1/TV at Standard Mode
D[5] RT0Enh256C, Enable Enhance 256 Color Graphic Mode
D[4] REnGamma, Enable Gamma Function
D[3:0] RVBCRTMOD[3:0], Mode Selection
RVBCRTMOD[3] : Enable direct LCD (Low active)
RVBCRTMOD[2:0]:
001 : Video Bridge Slave and CRT1 to RAMDAC
011 : Video Bridge Slave and CRT2 to RAMDAC
100 : Video Bridge Master and LCD Standard Mode
110 : Video Bridge Master and LCD Enhancement Mode
101 : Video Bridge Master and TV Standard Mode
111 : Video Bridge Master and TV Enhancement Mode

If you enable the Gamma correction bit on high-color mode or true-color mode, the value in palette registers will become gamma translate table.

Chap 3. Hardware Cursor Programming

To implement hardware cursor, on screen frame buffer will not be bothered by the cursor drawing. XGI Volari Family supports those hardware cursor listed below:

Monochrome Cursor

15 Bits Color Cursor

16 Bits Color Cursor

32 Bits Color Cursor

32 Bits Color Cursor with alpha blending shadow

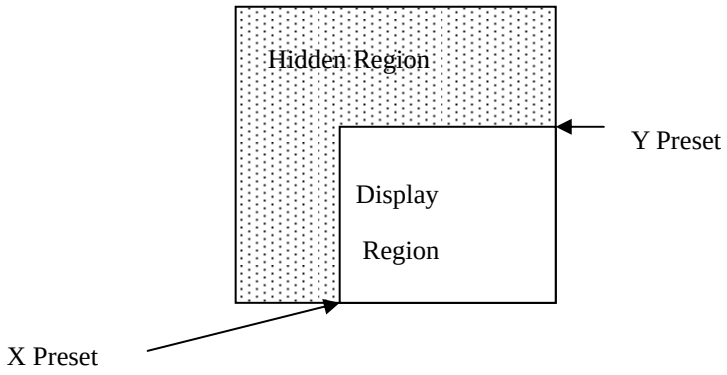
We will describe how to program hardware cursor on XGI Volari Family.

3-1 Term definition

The hardware cursor of XGI Volari Family is a 64×64 pixels square. There are some terms must be defined before we discuss how to program XGI Volari Family hardware cursor.

Cursor shape:	The hardware cursor image data. XGI Volari Family locate the hardware cursor shape on frame buffer.
X Position:	The pixel offset of the cursor left-up corner from the screen left.
Y Position:	The pixel offset of the cursor left-up corner from the screen top.
X Preset:	The pixel offset from the left edge of hardware cursor shape to the left edge of display region.
Y Preset:	The pixel offset from the top edge of hardware cursor shape to the top edge of display region.

Hardware Cursor Shape



3-2 Associated Register

The programming of hardware cursor on XGI Volari Family is via MMIO registers. The associated register are listed below:

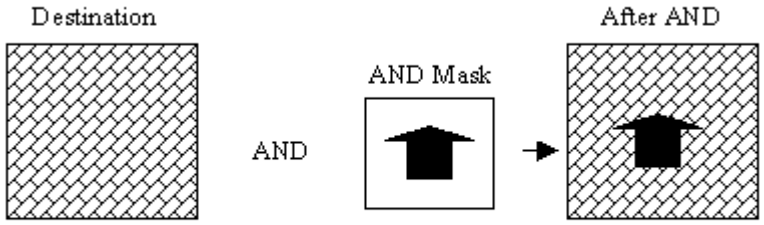
Register	Description
MMIO8500 CRT1 Hardware cursor pattern address	Default value: 00xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxb D31 Enable CRT1 color cursor D30 Enable CRT1 hardware cursor D[29:28] CRT1 Color cursor mode selection [1:0] 00: 15 bpp mode 01: 16 bpp mode 10: 32 bpp mode with alpha 11: 32 bpp mode D[27:24] CRT1 Hardware cursor pattern selection [3:0] D[23:17] reserved D[16:0] CRT1 Hardware cursor starting address [26:10]
MMIO8504 CRT1 Hardware cursor color 0 register	Default value: xxxxxxxxh D[31:24] reserved D[23:0] CRT1 Hardware cursor color 0 [23:0]
MMIO8508 CRT1 Hardware cursor color 1 register	Default value: xxxxxxxxh D[31:24] reserved D[23:0] CRT1 Hardware cursor color 1 [23:0]
MMIO850C CRT1 Hardware cursor horizontal location register	Default value: xxxxxxxxh D[31:22] reserved D[21:16] CRT1 Hardware cursor horizontal preset [5:0] D[15:12] reserved D[11:0] CRT1 Hardware cursor horizontal start [11:0]
MMIO8510 CRT1 Hardware cursor vertical location register	Default value: xxxxxxxxh D[31:22] reserved D[21:16] CRT1 Hardware cursor vertical preset [5:0] D[15:11] reserved D[10:0] CRT1 Hardware cursor vertical start [10:0]
MMIO8520 CRT2 Hardware cursor pattern address	Default value: 00xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxb D31 Enable CRT2 color cursor D30 Enable CRT2 hardware cursor D[29:28] CRT2 Color cursor mode selection [1:0]

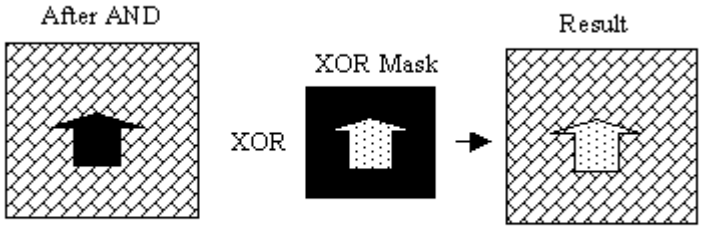
	00: 15 bpp mode 01: 16 bpp mode 10: 32 bpp mode with alpha 11: 32 bpp mode D[27:24] CRT2 Hardware cursor pattern selection [3:0] D[23:16] reserved D[15:0] CRT2 Hardware cursor starting address [26:10]
MMIO8524 CRT2 Hardware cursor color 0 register	Default value: xxxxxxxxh D[31:24] reserved D[23:0] CRT2 Hardware cursor color 0 [23:0]
MMIO8528 CRT2 Hardware cursor color 1 register	Default value: xxxxxxxxh D[31:24] reserved D[23:0] CRT2 Hardware cursor color 1 [23:0]
MMIO852C CRT2 Hardware cursor horizontal location register	Default value: xxxxxxxxh D[31:22] reserved D[21:16] CRT2 Hardware cursor horizontal preset [5:0] D[15:12] reserved D[11:0] CRT2 Hardware cursor horizontal start [11:0]
MMIO8530 CRT2 Hardware cursor vertical location register	Default value: xxxxxxxxh D[31:22] reserved D[21:16] CRT2 Hardware cursor vertical preset [5:0] D[15:11] reserved D[10:0] CRT2 Hardware cursor vertical start [10:0]
SR37 Extended CRT starting address register	Default value: 00h D[7] CRT2 Hardware cursor starting address [26] D[6:1] reserved D0 Screen starting address overflow [24] (in unit of DW)

3-3 Cursor Shape Format

You must store the cursor shape bitmap at dedicated location with dedicated format, then the XGI Volari Family can display hardware cursor correctly.

There are two logical cursor bitmap, named "AND Mask" and "XOR Mask". AND mask is an one-bit bitmap to and the destination, and the XOR mask is an n-bit bitmap to XOR with the destination after AND operation with AND mask. The figure below shows the action:





i.) Cursor Shape for Monochrome Hardware Cursor

The hardware cursor have a one-bit AND mask and one-bit XOR mask, each one are 64×64 pixel size. On XGI Volari Family, we place AND mask and XOR mask interlaced row by row. The figure below shows the placement.

byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12	byte 13	byte 14	byte 15
AND Mask Byte 0-7								XOR Mask Byte 0-7							
AND Mask Byte 8-15								XOR Mask Byte 8-15							
AND Mask Byte 16-23								XOR Mask Byte 16-23							
AND Mask Byte 24-31								XOR Mask Byte 24-31							
AND Mask Byte 32-39								XOR Mask Byte 32-39							
AND Mask Byte 40-47								XOR Mask Byte 40-47							
AND Mask Byte 48-55								XOR Mask Byte 48-55							
AND Mask Byte 56-63								XOR Mask Byte 56-63							

The cursor shape must locate on KByte boundary of frame buffer. In the case, each position have two bits, one is AND mask and the other is XOR mask. The combination for the cursor are listed in the table below:

AND Mask	XOR Mask	Pixel Color
0	0	Color 0 (Background Color)
0	1	Color 1 (Foreground Color)
1	0	Original color of destination (Transparent)
1	1	Inverse color of destination

ii.) Cursor Shape for 15 Bit-Color Hardware Cursor

The format of 15 Bit-Color is a continuous 64×64 word array for 64×64 pixels. The AND mask exist on the bit D[15] of each word, and the XOR mask exist on the bit D[14:0] with 5 bits red, 5 bits green, and 5 bits blue. The format of each word is shown below:

15	14 10	9 5	4 0
AND	Red	Green	Blue

In general, where AND mask bit is 1 the XOR mask opposite should be zero, then the AND mask usually called transparent bit.

iii.) Cursor Shape for 16 Bit-Color Hardware Cursor

The format of 16 Bit-Color is a continuous 64×64 word array for 64×64 pixels. The AND mask exist on the bit D[15] of each word, and the XOR mask exist on the bit D[15:11] with 5 bits red, D[10:6] with 5 bits green, and D[4:0] with 5 bits blue. Original XOR mask have 5 bits red, 6 bits green and 5 bits

blue, the lowest green bit is truncated to reserve one bit for AND mask. The format of each word is shown below:

15 11	10 6	5	4 0
Red	Green	AND	Blue

In general, where AND mask bit is 1 the XOR mask opposite should be zero, then the AND mask usually called transparent bit.

iv.) Cursor Shape for 32 Bit-Color Hardware Cursor without Alpha Blending

The format for 32 bit-color hardware cursor shape is a continuous 64×64 dword array to describe 64×64 pixels cursor. Each dword contains 8 bits red, 8 bits green, and 8 bits blue XOR mask and one bit AND mask located on D[24]. The format of each dword is shown below:

31	24	23 16	15 8	7 0
	AND Mask	Red	Green	Blue

v.) Cursor Shape for 32 Bit-Color Hardware Cursor with Alpha Blending

The format for 32 bit-color hardware cursor shape with alpha blending is a continuous 64×64 dword array to describe 64×64 pixels cursor. The type of cursor have no XOR cursor or AND cursor, the cursor shape has 8 bits red, 8 bits green, 8 bits blue, and 8 bits alpha value. The dword format is shown below:

31 24	23 16	15 8	7 0
Alpha Blending Value	Red	Green	Blue

To display this type of cursor, each pixel should blend with destination on the formula:

α=(Alpha Blending Value)/255
Rd = The red value of destination
Rc = The red value of cursor red part
Rs = The red value result shown on screen
Gd = The green value of destination
Gc = The green value of cursor red part
Gs = The green value result shown on screen
Bd = The blue value of destination
Bc = The blue value of cursor red part
Bs = The blue value result shown on screen
Rs = Rc×α+Rd×(1-α)
Gs = Gc×α+Gd×(1-α)
Bs = Bc×α+Bd×(1-α)

3-4 CRT1 Hardware Cursor Programming

To program hardware cursor on CRT1, you must follow the steps below:

To prepare cursor shape location.

To enable CRT1 hardware cursor.

To decide which type of hardware cursor.

To decide cursor position

To decide cursor preset

i.) Prepare Cursor Shape Location

At initialize phase, software must allocate a region to locate hardware cursor shape. There are different cursor types. If we assume that color cursor must be the same color depth as the desktop color depth, we can decide the hardware cursor region size as 64×64×2 = 8K bytes on 15/16 bpp color mode, and as 64×64×4 = 16K bytes on 32 bpp color mode.

The available bits to set color shape are D[26:10], it means you must locate hardware cursor on K bytes boundary location.

Use the way below to set CRT1 hardware cursor location:

```
mov   es, wMMIOBase
mov   eax, es:[8500h]
and   eax, 0FFFE0000h
mov   ecx, dHWCursorShapeAddr
shr   ecx, 10
or    eax, ecx
mov   es:[8500h], eax
```

MMIO 8500 D[27:24] have pattern selection to support multiple cursor shape switch, but I (ps, jjtseng) suggest you to modify cursor shape address to meet the requirement because the cost of IO is the same. And change cursor shape starting address is more easy to remember. We usually set the four bits zero.

ii.) Enable Hardware Cursor and Set Cursor Type

To enable CRT1 hardware cursor, you must set MMIO 8500 D[30] 1 to enable CRT1 hardware cursor.

If the hardware cursor is monochrome, set MMIO 8500 D[31] zero to disable color cursor. If the hardware cursor is color cursor, set the bit 1 and set D[29:28] to set the mode:

D[29:28]	Color Cursor Mode
00	15 bpp color cursor mode
01	16 bpp color cursor mode
10	32 bpp color cursor with alpha blending mode
11	32 bpp color cursor mode

If you choose monochrome cursor, you must set the color 0 (background color) and color 1 (foreground color) of the monochrome cursor. To write the 24 bits RGB color value into MMIO 8504 and MMIO 8508 register can program the color 0 and color 1 of CRT1 monochrome hardware cursor.

There are some code segments about set hardware cursor shape:

To set monochrome hardware cursor

```
mov   es, wMMIOBase
mov   eax, 40000000h    ; hardware cursor, but not color cursor
mov   ecx, dCursorShapeAddr
shr   ecx, 10
or    eax, ecx
mov   es:[8500h], eax
```

To set 15-bits color hardware cursor

```
mov   es, wMMIOBase
mov   eax, (1 shl 30)+(1 shl 31)+(00b shl 28)     ; hardware cursor, 15 bits color cursor
mov   ecx, dCursorShapeAddr
shr   ecx, 10
or    eax, ecx
```

```
mov  es:[8500h], eax
```

To set 16-bits color hardware cursor

```
mov  es, wMMIOBase
mov  eax, (1 shl 30)+(1 shl 31)+(01b shl 28)     ; hardware cursor, 16 bits color cursor
mov  ecx, dCursorShapeAddr
shr  ecx, 10
or   eax, ecx
mov  es:[8500h], eax
```

To set 32-bits color hardware cursor

```
mov  es, wMMIOBase
mov  eax, (1 shl 30)+(1 shl 31)+(11b shl 28)     ; hardware cursor, 32 bits color cursor
mov  ecx, dCursorShapeAddr
shr  ecx, 10
or   eax, ecx
mov  es:[8500h], eax
```

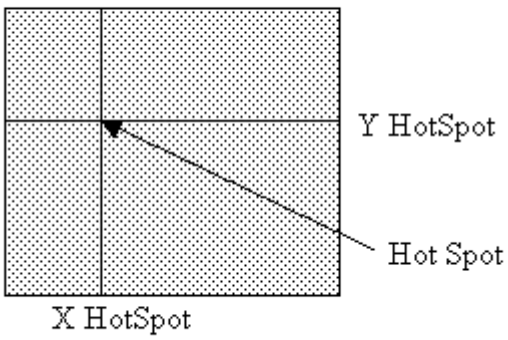
To set 32-bits color hardware cursor with alpha blending

```
mov  es, wMMIOBase
mov  eax, (1 shl 30)+(1 shl 31)+(10b shl 28)     ; hardware cursor, 32 bits color cursor with
alpha
mov  ecx, dCursorShapeAddr
shr  ecx, 10
or   eax, ecx
mov  es:[8500h], eax
```

iii.) Set Position And Preset

The position registers of CRT1 hardware cursor are MMIO 8514 and MMIO 8518. XGI Volari Family do not support negative position, therefore you must use preset value to control the cursor display region. For example, if the X position is -3, then you must write horizontal position zero and horizontal preset 3 (to display cursor shape from the third horizontal pixel). The vertical preset and position should be treated with the same way.

In many OS environment, the cursor shape data structure has the "Hot Spot" field to tell software which position is the focus. See the figure below:



To display it correctly, you must subtract the X HotSpot value from X position, and subtract the Y HotSpot value from Y position, the put into MMIO registers.

For example, if an cursor shape X HotSpot is 3, Y HotSpot is 5, and move focus to (123,333), the code to set the position is listed below:

```
        mov  es, wMMIOBase
        xor  cx, cx
        xor  dx, dx

        mov  ax, 123      ; X position
        mov  bx, 333      ; Y position
        sub  ax, 3
        jge  @f
        sub  cx, ax; if X position is negative, set it zero, and set preset value
        xor  ax, ax
        cmp  cx, 63
        jl   @f
        mov  cx, 63
@@:
        sub  bx, 5
        jge  @f
        sub  dx, bx
        xor  bx, bx       ; if Y position is negative, set it zero, and set preset value
        cmp  dx, 63
        jl   @f
        mov  dx, 63
@@:
        shl  ecx, 16
        shl  edx, 16
        mov  cx, ax
        mov  dx, bx
        mov  es:[850Ch], ecx
        mov  es:[8510h], edx
```

3-5 CRT2 Hardware Cursor Programming

To program hardware cursor on CRT2, you must follow the steps below:

To prepare cursor shape location.

To enable CRT2 hardware cursor.

To decide which type of hardware cursor.

To decide cursor position

To decide cursor preset

i.) Prepare Cursor Shape Location

At initialize phase, software must allocate a region to locate hardware cursor shape. There are different cursor types, if we assume color cursor must be the same color depth as the desktop color depth, we can decide the hardware cursor region size as 64×64×2 = 8K bytes on 15/16 bpp color mode, as 64×64×4 = 16K bytes on 32 bpp color mode.

The available bits to set color shape are D[26:10], it means you must locate hardware cursor on K bytes boundary location. The XGI Volari Family have a bug on MMIO 8520 register that D[16] is fail. Thus, the D[26] of cursor shape is put on SR37 D[7].

Use the way below to set CRT2 hardware cursor location:

```
mov  es, wMMIOBase
mov  eax, es:[8500h]
```

```
and   eax, 0FFFE0000h
mov   ecx, dHWCursorShapeAddr
shr   ecx, 10
mov   ax, cx
mov   es:[8520h], eax    ; set the cursor shape address D[25:10].
mov   dx, wExtendedIOBase
mov   al, 37h
out   dx, al
inc   dx
in    al, dx
and   al, 7fh
shr   ecx, 16
and   cl, 1
shl   cl, 7
or    al, cl  ; set the D[26] to SR37 D[7]
out   dx, al
```

MMIO 8520 D[27:24] have pattern selection to support multiple cursor shape switch, but I (ps, jjtseng) suggest you to modify cursor shape address to meet the requirement because the cost of IO is the same. And change cursor shape starting address is more easy to remember. We usually set the four bits zero.

ii.) Enable Hardware Cursor and Set Cursor Type

To enable CRT2 hardware cursor, you must set MMIO 8520 D[30] 1 to enable CRT2 hardware cursor.

If the hardware cursor is monochrome, set MMIO 8520 D[31] zero to disable color cursor. If the hardware cursor is color cursor, set the bit 1 and set D[29:28] to set the mode:

D[29:28]	Color Cursor Mode
00	15 bpp color cursor mode
01	16 bpp color cursor mode
10	32 bpp color cursor with alpha blending mode
11	32 bpp color cursor mode

If you choose monochrome cursor, you must set the color 0 (background color) and color 1 (foreground color) of the monochrome cursor. To write the 24 bits RGB color value into MMIO 8524 and MMIO 8528 register can program the color 0 and color 1 of CRT2 monochrome hardware cursor.

There are some code segments about set hardware cursor shape:

To set monochrome hardware cursor

```
mov   es, wMMIOBase
mov   eax, 40000000h   ; hardware cursor, but not color cursor
mov   ecx, dCursorShapeAddr
shr   ecx, 10
or    eax, ecx
mov   es:[8520h], eax
```

To set 15-bits color hardware cursor

```
mov   es, wMMIOBase
mov   eax, (1 shl 30)+(1 shl 31)+(00b shl 28)    ; hardware cursor, 15 bits color cursor
mov   ecx, dCursorShapeAddr
shr   ecx, 10
or    eax, ecx
mov   es:[8520h], eax
```

To set 16-bits color hardware cursor

```
mov   es, wMMIOBase
mov   eax, (1 shl 30)+(1 shl 31)+(01b shl 28)    ; hardware cursor, 16 bits color cursor
```

```
mov  ecx, dCursorShapeAddr
shr  ecx, 10
or   eax, ecx
mov  es:[8520h], eax
```

To set 32-bits color hardware cursor

```
mov  es, wMMIOBase
mov  eax, (1 shl 30)+(1 shl 31)+(11b shl 28)     ; hardware cursor, 32 bits color cursor
mov  ecx, dCursorShapeAddr
shr  ecx, 10
or   eax, ecx
mov  es:[8520h], eax
```

To set 32-bits color hardware cursor with alpha blending

```
mov  es, wMMIOBase
mov  eax, (1 shl 30)+(1 shl 31)+(10b shl 28)     ; hardware cursor, 32 bits color cursor with
alpha
mov  ecx, dCursorShapeAddr
shr  ecx, 10
or   eax, ecx
mov  es:[8520h], eax
```

iii.) Set Position And Preset

The position registers of CRT2 hardware cursor are MMIO 8534 and MMIO 8538. XGI Volari Family do not support negative position, therefore you must use preset value to control the cursor display region. For example, if the X position is -3, then you must write horizontal position zero and horizontal preset 3 (to display cursor shape from the third horizontal pixel). The vertical preset and position should be treated with the same way.

To display it correctly, you must subtract the X HotSpot value from X position, and subtract the Y HotSpot value from Y position, the put into MMIO registers.

Because the display delay, the CRT2 hardware cursor will shift to left 17 pixel on XGI Volari Family. We must add 17 pixels offset on CRT2 abstract X position.

For example, if an cursor shape X HotSpot is 3, Y HotSpot is 5, and move focus to (123,333), the code to set the position is listed below:

```
        mov  es, wMMIOBase
        xor  cx, cx
        xor  dx, dx

        mov  ax, 123      ; X position
        add  ax, 17       ; add CRT2 X placement delay.
        mov  bx, 333      ; Y position
        sub  ax, 3
        jge  @f
        sub  cx, ax; if X position is negative, set it zero, and set preset value
        xor  ax, ax
        cmp  cx, 63
        jl   @f
        mov  cx, 63
@@:
        sub  bx, 5
        jge  @f
        sub  dx, bx
        xor  bx, bx       ; if Y position is negative, set it zero, and set preset value
        cmp  dx, 63
        jl   @f
```

```
        mov  dx, 63
@@:
        shl  ecx, 16
        shl  edx, 16
        mov  cx, ax
        mov  dx, bx
        mov  es:[852Ch], ecx
        mov  es:[8530h], edx
```

iv.) Hardware Programming Cursor Limitation of XG40/XG42

	Item	Limitation
CRT1 Cursor	Disable/Enable	Whenever the cursor disable/enable issued, the issue will be triggered on the next vertical retrace time.
	Cursor Shape Base Address	Whenever the cursor shape base address is changed, it will active on the next vertical retrace time.
	Pattern Selection	Whenever the cursor pattern selection is changed, the issue will be active on the next vertical retrace time.
	Cursor Format	Whenever the cursor format is changed, the issue will be active on the next vertical retrace time.
	X/Y Position	Positive word value. Because all cursor parameter (color format, disable/enable, base address, pattern selection, position, preset) will be latched and not update until the vertical retrace, the Y position update will be treat as the update sequence trigger owner. All data will be update before last Y position update.
	X/Y Preset	Positive word value < 64.
CRT2 Cursor	Disable/Enable	Whenever the cursor disable/enable issued, the issue will be triggered on the next vertical retrace time.
	Cursor Shape Base Address	Whenever the cursor shape base address is changed, it will active on the next vertical retrace time.
	Pattern Selection	Whenever the cursor pattern selection is changed, the issue will be active on the next vertical retrace time.
	Cursor Format	Whenever the cursor format is changed, the issue will be active on the next vertical retrace time.
	X/Y Position	Positive word value. Because all cursor parameter (color format, disable/enable, base address, pattern selection, position, preset) will be latched and not update until the vertical retrace, the Y position update will be treat as the update sequence trigger owner. All data will be update before last Y position update.
	X/Y Preset	Positive word value < 64.

v.) Programming Method

Disable CRT1 HW Cursor	Issue a HW cursor disabled command immediately. The issue will be taken action while next vertical retrace.
Set CRT1 HW Cursor base	Modify a HW cursor base address immediately. It will change while next

address	vertical retrace.
Set CRT1 HW Cursor pattern selection	Modify a HW cursor pattern selection immediately. It will change while next vertical retrace.
Change CRT1 HW Cursor shape with color format changed.	1. Fill new cursor shape to a dedicated cursor pattern address for the color format (CRT1/CRT2). 2. Set new base address and new color format. 3. Set X position, X preset. 4. Set Y position, Y preset.
Change CRT1 HW Cursor shape without color format changed.	1. Fill new cursor shape to a dedicated cursor pattern address for the color format (CRT1/CRT2). 2. Set X position, X preset. 3. Set Y position, Y preset.

Disable CRT2 HW Cursor	Issue a HW cursor disabled command immediately. The issue will be taken action while next vertical retrace.
Set CRT2 HW Cursor base address	Modify a HW cursor base address immediately. It will change while next vertical retrace.
Set CRT2 HW Cursor pattern selection	Modify a HW cursor pattern selection immediately. It will change while next vertical retrace.
Change CRT2 HW Cursor shape with color format changed.	1. Fill new cursor shape to a dedicated cursor pattern address for the color format (CRT1/CRT2). 2. Set new base address and new color format. 3. Set X position, X preset. 4. Set Y position, Y preset.
Change CRT2 HW Cursor shape without color format changed.	1. Fill new cursor shape to a dedicated cursor pattern address for the color format (CRT1/CRT2). 2. Set X position, X preset. 3. Set Y position, Y preset.

vi.) Software Notice

For CRT1/CRT2 have the issue that CRT1 cursor and CRT2 cursor cannot set in the same base address, the CRT1 cursor base address and CRT2 cursor base address should be separated even though they shows the same data under mirror mode.

To avoid cursor position noise, we set Y as the cursor parameter trigger point. When the HW cursor update the parameter during the vertical retrace, the parameter before last Y position update will be taken active, and last parameters will merge with following Y position update.

vii.) Hardware Notice

To consider the Y position update sequence.

PART. II. 2D Graphic Engine

Chap 4. 2D Engine Initialization

The 2D engine initialization is to do something let 2D engine programmable. Those action items are listed below:

To enable memory mapped IO

To enable 2D engine module

To enable suitable command queue programming style

The command queue programming is a large topic, It will describe in later chapter.

4-1 Enable Memory Mapped IO

It is necessary to get/set information from engine with memory mapped I/O (MMIO), therefore, you must ensure MMIO enabled before you setup engine. The MMIO enable switch is in SR20 D[0]. Whenever you want to enable it, set the register as 1. The register description is listed below:

Register	Description
SR 20 PCI address decoder setting register	Default value: 20h D7 PCI linear address mode enable 0: disable 1: enable D6 reserved D5 Relocated VGA I/O port decoding enable 0: disable 1: enable D4 Standard VGA I/O port decoding disable 0: enable 1: disable D3 Enable video playback MMIO address decoding 0: disable 1: enable D2 Disable standard memory address (A000-Bfff) access 0: enable 1: disable D1 reserved D0 Memory mapped I/O enable 0: disable 1: enable

The method to enable MMIO is:

```
mov  dx, wExtendedIOBase
mov  al, 20h
out  dx, al
inc  dx
in   al, dx
or   al, 1   ; set MMIO enabled.
out  dx, al
```

4-2 Enable 2D Engine Module

To enable the 2D module bit in SR 1E (SR1E D[6]) can enable 2D engine. To ensure all engine is idle before you programming this registers. The register description is listed below:

Register	Description
SR1E Module enable register	Default value: 00h D7 Enable 3D G/L transformation engine 0: disable 1: enable D6 Enable 2D 0: disable 1: enable D5 Enable CRT 2 0: disable 1: enable D4 Enable 3D AGP vertex command fetch 0: disable 1: enable D3 Enable 3D command parser 0: disable 1: enable D2 Enable VGA BIOS flash memory data write 0: disable 1: enable D1 Enable 3D accelerator 0: disable 1: enable D0 Enable hardware MPEG 0: disable 1: enable

You can use the following steps to enable 2D engine module:

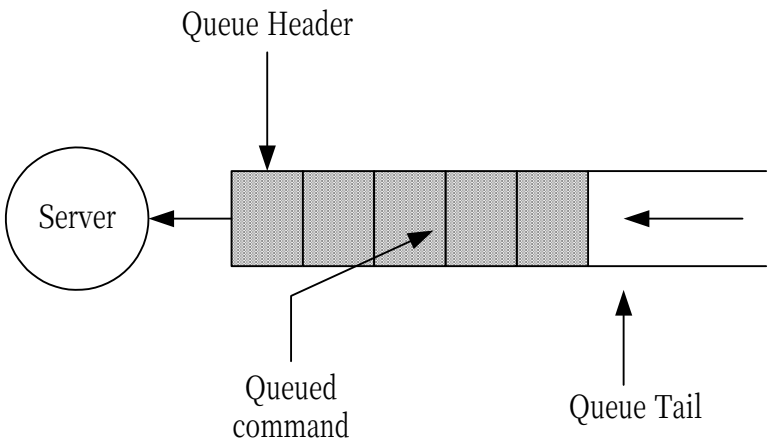
```
mov  dx, wExtendedIOBase
mov  al, 1Eh
out  dx, al
inc  dx
in   al, dx
or   al, 1 shl 6    ; to enable 2D engine module
out  dx, al
```

Chap 5. Command Queue

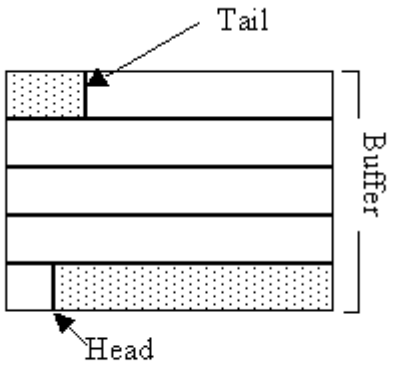
5-1 Initialization

XGI Volari Family use a new command queue mechanism to the graphics chips. The cost to program engine register via MMIO is very large during execution. Therefore, the design group let command located on AGP memory area and fetched by XGI Volari Family chip via AGP cycle or to put command on frame buffer with CPU write, and fetched by XGI Volari Family.

Before the description of command queue initialization, we must know some terms about command queue. In general queuing model, all requests are first-in-first-out (FIFO), there is a head of queue to get request, and a tail to put request. The figure below shows that.



In physical implement, we usually implement queue with fixed buffer as a ring queue. That is, when head or tail round to the start of buffer whenever they go to end. Like the figure below:



The terms listed below are those term we should initialize:

Term	Description
Queue Base	The physical address of the queue buffer base.
Write point	The port to write request into queue, the term "Tail" of the above queuing model.
Read point	The port to read request from queue, the term "Head" of the above queuing model.
Queue Size	The valid buffer size.

There are two types of command queue programming for XGI Volari Family. One put the command on AGP buffer, the other put the command on frame buffer. The initialize of command queue include those items:

Decide which type of queue place.

Decide queue size.

Decide queue base.

Reset command queue.

The command queue associated register are listed below:

Register	Description
SR26 Command queue control register	Default value: 00h D7 Enable AGP buffer command mode 0: disable 1: enable D6 Enable Video-RAM buffer command mode 0: disable 1: enable D5 Enable MMIO command mode 0: disable 1: enable D4 Bypass H/W queue mode

	0: normal mode 1: bypass mode D[3:2] Command queue length [1:0] 00: 512K queue length 01: 1M queue length 10: 2M queue length 11: 4M queue length D1 Enable command queue auto-correction control function 0: disable 1: enable D0 MCLK synchronous reset generation register for command queue 0: normal operation 1: reset
SR27 Command queue threshold value	Default value: 00h D[7:5] Reserved D[4:0] CQ threshold value; max value is “11111”
MMIO85C0 command queue based address	Default value: 00000000h D[31:0] Command queue based address [31:0]
MMIO85C4 command queue writer pointer	Default value: 00000000h D[31:22] Reserved D[21:4] Command queue writer pointer [31:0] D[3:0] Reserved
MMIO85C8 command queue read pointer	Read only D[31:22] Reserved D[21:4] Command queue read pointer [31:0] D[3:0] Reserved
MMIO85CC command queue read pointer	D31 2D idle, 3D idle, HQ & SQ empty 0: bussy or not empty 1: idle and empty D30 HQ empty (include H/W CQ, 2D queue, & 3D queue) 0: not empty 1: empty D29 2D idle 0: busy 1: idle D28 3D idle 0: busy 1: idle D27 H/W CQ empty 0: not empty 1: empty D26 2D queue empty 0: not empty 1: empty D25 3D queue empty 0: not empty 1: empty D24 S/W CQ queue empty 0: not empty 1: empty D[23:16] 2D counter 3 D[15:8] 2D counter 2

	D[7:0] 2D counter 1

The following steps are to initialize the command queue.

Set SR26 D[0] 1 to reset command queue.

Set SR26 D[7:4] to decide if the command queue AGP command queue or frame buffer queue.

Set SR26 D[3:2] to set the command queue size.

Set SR26 D[1] to prepare command queue set. The time queue will not be set until the queue base updated.

Get current read point from register 85C8, and write it to write point register 85C4 to synchronize new write point.

Write the register 85C0 to update the command queue base and trigger command queue set. On AGP command mode, the physical base of AGP buffer address should write into the register. On frame buffer command mode, the frame buffer offset should write into the register.

For example, if the command queue is located on AGP buffer with physical memory base 0xD8000000 and size is 2MB, the setting procedure is below:

```
mov  dx, wExtendedIOBase
mov  al, 26
out  dx, al
inc  dx
in   al, dx
or   al, 1   ; to reset command queue
out  dx, al
or   al, 80h       ; set AGP command queue
or   al, 10b shl 2       ; set command queue size 2MB
out  dx, al
and  al, (0ffh-1)  ; prepare to set command queue
out  dx, al
mov  es, wMMIOBase
mov  eax, dword ptr es:[85c8h]
mov  dword ptr es:[85c4h], eax      ; to synchronize read/write point
mov  eax, 0D8000000h
mov  es:[85c0h], eax    ; to update queue base, set command queue
```

If we want to set the command queue located on frame buffer offset 0x380000 with queue size 512KB, the procedure is below:

```
mov  dx, wExtendedIOBase
mov  al, 26
out  dx, al
inc  dx
in   al, dx
or   al, 1   ; to reset command queue
out  dx, al
or   al, 40h       ; set frame buffer command queue
or   al, 00b shl 2       ; set command queue size 512KB
out  dx, al
and  al, (0ffh-1)  ; prepare to set command queue
out  dx, al
mov  es, wMMIOBase
mov  eax, dword ptr es:[85c8h]
mov  dword ptr es:[85c4h], eax      ; to synchronize read/write point
mov  eax, 380000h
```

```
mov   es:[85c0h], eax     ; to update queue base, set command queue
```

Note: The AGP buffer base is given from AGP service, and it is different from each OS.

5-2 Command Queue Programming

The command queue of XGI Volari Family locates on AGP memory or local frame buffer. XGI Volari Family will issue a request to fetch the required command packet from the command queue buffer area. However, software must maintain the read point and write point of command queue, the mechanism is more complex than the queue mechanism in previous generations.

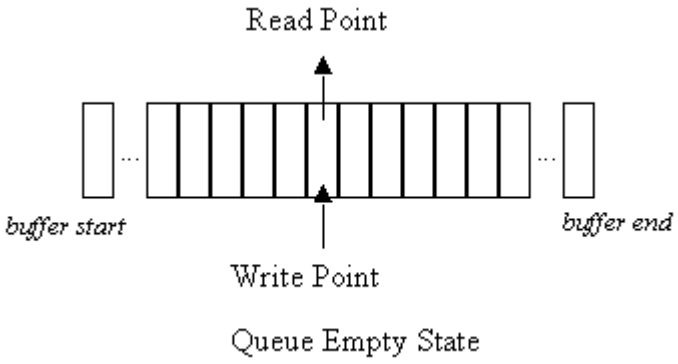
i.) Queue Empty/Full Definition

We must define the "Empty" and "Full" of command queue at first.

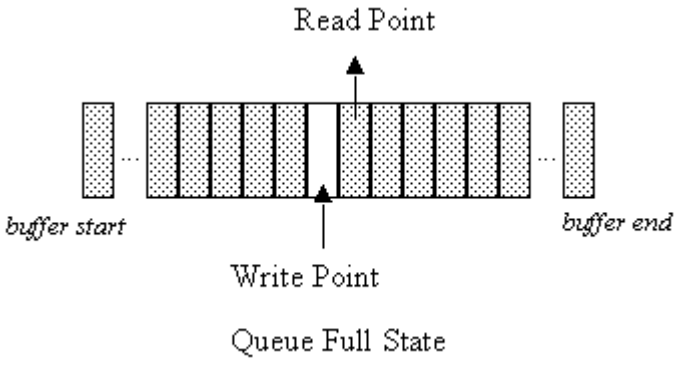
Queue empty: if read point equal to write point.

Queue full: if read point equal to (write point + 1)%QueueSize.

See the figure it describe the empty and full state:



Queue Empty State



Queue Full State

Note: There is always one bobble on queue full state.

ii.) Update Write Point and Read Point

When read point is equal to write point, it means the queue is empty. Whenever the command queue is not empty, the read point is not equal to write point, and the read point will move to fetch the command until the read point equal to the write point.

Thus, whenever the software update the write point, it means new commands are given into command queue.

The content of read point (register 85C8) and write point (register 85C4) are both the offset from the command queue base, thus the programming method under AGP queue and frame buffer queue is the same way. The unit to update write point or hardware to update read point is one AGP request slot (16 bytes or 4 dwords). The bit definition is shift to byte boundary, thus the bits [3:0] of register 85C4 and 85C8 are both zero for alignment.

This is an example to update write point:

```
mov  gs, wCmdSelectorBase ; gs:0 point to the command queue buffer
mov  es, wMMIOBase
mov  edi, es:[85C4h]    ; load current write point
mov  eax, 168F8200h
mov  ebx, 0
mov  ecx, 168F8204h
mov  edx, 0
mov  gs:[edi], eax
mov  gs:[edi+4], ebx
mov  gs:[edi+8], ecx
mov  gs:[edi+0ch], edx ; fill a slot of data
add  edi, 10h      ; increase one slot
and  edi, QueueMask   ; edi = edi % QueueSize, QueueMask = QueueSize - 1.
; To prevent overwrite buffer.
mov  es:[85C4h], edi    ; update write point.
```

iii.) Command Queue Maintenance

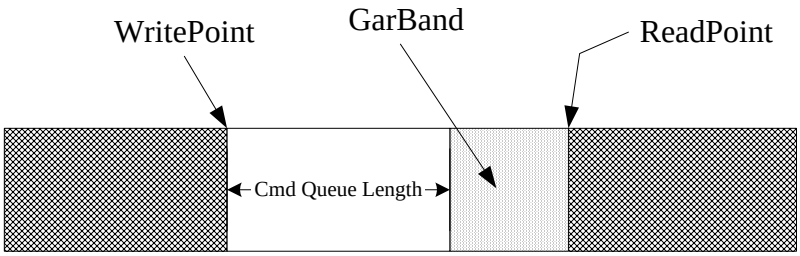
The most important work to maintain command queue is to prevent overwrite command queue. These terms are defined:

Command Queue Length: the available length to insert command into queue.

GarBand: the fixed length buffer for safely between read point and write point.

Therefore, the command queue length formula is given:

CommandQueueLength = ([ReadPoint] - [WritePoint] - Gardband) % QueueSize



5-3 Command Format

Now we know the command maintenance mechanism, but we do not know how to program the engine with the command queue.

There are four catalogs of command packet, to packet the register and data to program 2D/3D engine. They are

NULL Command: For hardware issue, each command must be located on 64 bits boundary in command queue, and each write point update must be 128 bits boundary with the AGP limitation. However, not all command packet can fit all the above requirement, a null type command is required. This catalog contains only one command, a 32 bits with value 0x168A0000, can be the bobble of command queue.

Single Command: The catalogue is a two double word format, the first dword have the tag and the register ID of engine, the second dword contains the value want to write.

Burst Command: The catalogue is a dword string with variable length. The first dword contains the first tag and starting register ID of engine, the second dword contains the second tag and following data length (larger than 1) with dword unit. The following dword sequence are the value list that we want to program into registers.

Packet Command: The catalogue contains a set of fixed format packets. Those packets have different definition described in following section, and they are defined for special usage on 2D/3D engine function.

i.) NULL Command

31 31	29	28	27 24	23 20	19 16	15 0
0 0	0	1	0 1 1 0	1 0 0 0	1 1 1 1	0

ii.) Single Command Mode

31 30	29	28	27 24	23 20	19 16	15 0
0 0	X	1	0 1 1 0	1 0 0 0	CBE	Engine Register Address

63 32
Data

X: 0 = 2D / 1 = 3D

iii.) Burst Command Mode

31 30	29	28	27 24	23 20	19 16	15 0
0 1	X	1	0 1 1 0	1 0 0 0	1 0 1 0	Start Engine Register Address

63 60	59 56	55 52	51 32
0 1 1 0	0 0 1 0	0 0 0 1	Total Followed Data Length

X: 0 = 2D / 1 = 3D

iv.) Packet Command Mode

31 30	29	28	27 24	23 20	19 16	15 5	4 0
1 0	X	1	0 1 1 0	1 0 0 0	1 0 1 0	0	Type Number

63 60	59 56	55 52	51 32
0 1 1 0	0 0 1 0	0 0 0 1	Total Followed Data Length

X: 0 = 2D / 1 = 3D

v.) Source Data Command Mode

31 30	29	28	27 24	23 20	19 16	15 0
1 1	X	1	0 1 1 0	1 0 0 0	1 0 1 0	(Reserved)

63 60	59 56	55 52	51 32
0 1 1 0	0 0 1 0	0 0 0 1	Total Followed Data Length

Total Followed Data must be 16 bytes (4 double words or 128 bits) times value.

X: 0 = 2D / 1 = 3D

Note:

The burst mode/packet mode as following style:

Burst Mode Header/Packet Mode Header
DWORD CMD 1
DWORD CMD 2
: *:*
DWORD CMD N

The read/write point should be 128 bits alignment.

Each command packet should be 64 bits alignment, programmer must add NULL to complete boundary.

vi.) All 2D Command Packet Type

Type 1: Source Copy Without Pattern (Length = 8)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00001
0 1 1 0	0 0 1 0	0 0 0 1	0	01000
Src Base Addr (8200)				
Src Pitch (8204)				
Src X (820A)			Src Y (8208)	
Dst X (820E)			Dst Y (820C)	
Dst Base Addr (8210)				
Dst Height (8216)			Dst Pitch (8214)	
Rect Height (821A)			Rect Width (8218)	
Command (823C)				

Type 2: Source Copy With Color Solid Brush (Length = 9)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00010
0 1 1 0	0 0 1 0	0 0 0 1	0	01001
Src Base Addr (8200)				

Src Pitch (8204)	
Src X (820A)	Src Y (8208)
Dst X (820E)	Dst Y (820C)
Dst Base Addr (8210)	
Dst Height (8216)	Dst Pitch (8214)
Rect Height (821A)	Rect Width (8218)
Pat Foreground Color (821C)	
Command (823C)	

Type 3: Source Copy With Mono Pattern (Length = 12)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00011
0 1 1 0	0 0 1 0	0 0 0 1	0	01100
Src Base Addr (8200)				
Src Pitch (8204)				
Src X (820A)				Src Y (8208)
Dst X (820E)				Dst Y (820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Rect Height (821A)				Rect Width (8218)
Pat Foreground Color (821C)				
Pat Background Color (8220)				
Mono Mask 3 (822F)	Mono Mask 2 (822E)	Mono Mask 1 (822D)	Mono Mask 0 (822C)	
Mono Mask 7 (8233)	Mono Mask 6 (8232)	Mono Mask 5 (8231)	Mono Mask 4 (8230)	
Command (823C)				

Type 4: No Source No Pattern/Pattern Blt With Pattern Register (Length = 5)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00100
0 1 1 0	0 0 1 0	0 0 0 1	0	00101
Dst X (820E)				Dst Y (820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Rect Height (821A)				Rect Width (8218)
Command (823C)				

Type 5: Pattern Blt with Color Solid Brush (length=6)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00101
0 1 1 0	0 0 1 0	0 0 0 1	0	00110
Dst X (820E)				Dst Y (820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Rect Height (821A)				Rect Width (8218)

Pat Foreground Color (821C)
Command (823C)

Type 6: Pattern Blt With Mono Mask (Length = 9)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00110
0 1 1 0	0 0 1 0	0 0 0 1	0	01001
Dst X (820E)				Dst Y (820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Rect Height (821A)				Rect Width (8218)
Pat Foreground Color (821C)				
Pat Background Color (8220)				
Mono Mask 3 (822F)		Mono Mask 2 (822E)		Mono Mask 1 (822D)	Mono Mask 0 (822C)
Mono Mask 7 (8233)		Mono Mask 6 (8232)		Mono Mask 5 (8231)	Mono Mask 4 (8230)
Command (823C)				

Type 7: Mono Source Expansion Without Pattern(Length = 10)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 00111
0 1 1 0	0 0 1 0	0 0 0 1	0	01010
Src Base Addr (8200)				
Src Pitch (8204)				
Src X (820A)				Src Y (8208)
Dst X (820E)				Dst Y (820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Rect Height (821A)				Rect Width (8218)
Src Foreground Color (8224)				
Src Background Color (8228)				
Command (823C)				

Type 8: Mono Source Expansion With Color Solid Brush (length = 11)

31 30	29	28	27 24	23 20	19 16	15 0
1 0	0	1	0 1 1 0	1 0 0 0	1 0 1 0	0 01000
0 1 1 0	0 0 1 0		0 0 0 1	0		01011
Src Base Addr (8200)						
Src Pitch (8204)						
Src X (820A)						Src Y (8208)
Dst X (820E)						Dst Y (820C)
Dst Base Addr (8210)						
Dst Height (8216)						Dst Pitch (8214)
Rect Height (821A)						Rect Width (8218)
Pat Foreground Color (821C)						
Src Foreground Color (8224)						
Src Background Color (8228)						
Command (823C)						

Type 9: Gradient Fill (Length = 11)

31 30	29	28	27 24	23 20	19 16	15 0
1 0	0	1	0 1 1 0	1 0 0 0	1 0 1 0	0 01001
0 1 1 0	0 0 1 0		0 0 0 1	0		01011
Dst X (820E)						Dst Y (820C)
Dst Base Addr (8210)						
Dst Height (8216)						Dst Pitch (8214)
Rect Height (821A)						Rect Width (8218)
Initial Color (RGB) (821C)						
Delta R (8220)						
Delta G (8224)						
Delta B (8228)						
Clip Top (8236)						Clip Left (8234)
Clip Bottom (823A)						Clip Right (8238)
Command (823C)						

Type 10: Dest Device Initialization (Length = 6)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0	01010
0 1 1 0	0 0 1 0	0 0 0 1	0	00110
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Src Foreground Color (8224)				
Src Background Color (8228)				
Clip Top (8236)				Clip Left (8234)
Clip Bottom (823A)				Clip Right (8238)

Type 11: Each Character Output (With Enhance Color Expansion) (Length = 6)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0	01011
0 1 1 0	0 0 1 0	0 0 0 1	0	00110
Src Base Addr (8200)				
Src Pitch (8204)				
Src X (820A)				Src Y (8208)
Dst X (820E)				Dst Y (820C)
Rect Height (821A)				Rect Width (8218)
Command (823C)				

Type 12: For New Functions (Length = 16)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0	01100
0 1 1 0	0 0 1 0	0 0 0 1	0	10000
Src Base Addr (8200)				
Src Pitch (8204)				
Src X (820A)				Src Y (8208)
Dst X (820E)				Dst Y (820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Rect Height (821A)				Rect Width (8218)
Pat Foreground Color (821C)				
Pat Background Color (8220)				
Src Foreground Color (8224)				

Src Background Color (8228)			
Mono Mask 3 (822F)	Mono Mask 2 (822E)	Mono Mask 1 (822D)	Mono Mask 0 (822C)
Mono Mask 7 (8233)	Mono Mask 6 (8232)	Mono Mask 5 (8231)	Mono Mask 4 (8230)
Clip Top (8236)		Clip Left (8234)	
Clip Bottom (823A)		Clip Right (8238)	
Command (823C)			

Type 13: Each Character Out (With Color Expansion) (Length = 5)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 01101
0 1 1 0	0 0 1 0	0 0 0 1	0	00101
Src Pitch (8204)				
Src X (820A)				Src Y (8208)
Dst X (820E)				Dst Y (820C)
Rect Height (821A)				Rect Width (8218)
Command (823C)				

5-4 Command Packet for Output Function

Type 14: Polyline With First line segment (Length = 12)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 01110
0 1 1 0	0 0 1 0	0 0 0 1	0	01100
Y0 (820A)				X0 (8208)
Y1 (820E)				X1(820C)
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Style Period (821A)				Line Count (8218)
Foreground Color (821C)				
Background Color (8220)				
Line Style 0 (822C)				
Line Style 1 (8230)				
Clip Top (8236)				Clip Left (8234)
Clip Bottom (823A)				Clip Right (8238)
Command (823C)				

Below package is for fill region/scanline

Type 15: Initialized Data for Monochrome Pattern Brush (length=6)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 01111
0 1 1 0	0 0 1 0	0 0 0 1	0	00110
Dst Base Addr (8210)				
Dst Height (8216)				Dst Pitch (8214)
Pat Foreground Color (821C)				

Pat Background Color (8220)			
Mono Mask 3 (822F)	Mono Mask 2 (822E)	Mono Mask 1 (822D)	Mono Mask 0 (822C)
Mono Mask 7 (8233)	Mono Mask 6 (8232)	Mono Mask 5 (8231)	Mono Mask 4 (8230)

Chap 6. The 2D Engine Programming

Method

It is easy to program the XGI Volari Family 2D engine. When you want to program a specified XGI Volari Family 2D engine function, you just fill the 2D engine registers except register 823C with those value you want to program, then write register 823C with specified command, the engine will be fired.

How to program XGI Volari Family 2D engine function? There are three modes:

MMIO mode. The mode have only short HW queue FIFO, each written of engine register must keep pair of writing and must wait for engine idle at first for the queue issue. It is simple to program but only usable on debugging mode.

AGP command queue mode. The mode use an AGP buffer as queue area, write command with specified format that queue dispatcher know. The mode was described on the chapter Command Queue Programming.

Local frame buffer (LFB) queue mode. The mode is similar to AGP command queue mode but use frame buffer instead AGP surface.

If you want to use MMIO mode to program 2D engine, you must set SR26 D[7:4] as 0011, turn off AGP command queue and LFB command queue, and enable MMIO mode. Then you can write the register with MMIO way.

For example:

```
        mov   es, wMMIOBase
@@:
        test    dword ptr es:[85C8h+3], (1 shl 31)
        jz      @b    ; engine is not idle
        mov   eax, 12345678h
        mov   es:[8200h], eax
        mov   eax, 0
        mov   es:[8260h], eax    ; write with pair
```

If you use AGP command queue mode or LFB command queue mode, you can use single command mode to write a single engine register:

```
; PKT_SINGLE_CMD_HEADER   EQU     16800000h
;
; dsPktSingleCmd  struct
;       PkSC_dwHeader   dd      PKT_SINGLE_CMD_HEADER
;       PkSC_dwData     dd      ?
; dsPktSingleCmd  ends

        mov   es, wMMIOBase
        mov   gs, wCmdQueueSelector        ; gs:0 point to queue buffer start
        mQueryCmdQueueLength 2   ; to query two dword length.
        mov   edi, es:[85C4h]    ; gs:[edi] = the current write point
        mov   eax, PKT_SINGLE_CMD_HEADER      ; 16800000h
```

```
        or      eax, 8200h  ; to programming
        mov     ebx, 12345678h
        mov     gs:[edi].PkSC_dwHeader, eax
        mov     gs:[edi].PkSC_dwData, ebx
        mov     eax, 8260h
        mov     ebx, 0
        mov     gs:[edi+8].PkSC_dwHeader, eax
        mov     gs:[edi+8].PkSC_dwData, ebx
        add     edi, 10h
        and     edi, QueueMask
        mov     es:[85C4h], edi
```

If you want to program continuous registers, you can choose the burst packet mode, such as to program pattern register:

```
; PKT_BURST_CMD_HEADER0 EQU     568A0000h
; PKT_BURST_CMD_HEADER1 EQU     62100000h
;
; dsPktBurstCmd struct
;       PkBC_dwHeader0  dd      PKT_BURST_CMD_HEADER0
;       PkBC_dwHeader1  dd      PKT_BURST_CMD_HEADER1
;       PkBC_dwData     dd      ?
; dsPktBurstCmd ends


        mov     es, wMMIOBase
        mov     gs, wCmdQueueSelector       ; gs:0 point to queue buffer start
        mQueryCmdQueueLength 64+4       ; to query two dword length.
        mov     edi, es:[85C4h]     ; gs:[edi] = the current write point
        mov     eax, PKT_BURST_CMD_HEADER0
        mov     ecx, PKT_BURST_CMD_HEADER1
        add     eax, 8300h  ; begin from 8300h
        add     ecx, 64         ; to write 64 pattern dword
        mov     ds, lpPBrush.sel
        movzx       esi, lpPBrush.off  ; ds:esi point to pattern source
        mov     es:[edi].PkBC_dwHeader0, eax
        mov     es:[edi].PkBC_dwHeader1, ecx
        add     edi, 10h
        mov     ecx, 64
@@:
        mov     eax, ds:[esi]
        mov     ebx, ds:[esi+4]
        mov     es:[edi-8], eax
        mov     es:[edi-4], ebx
        and     edi, QueueMask
        mov     eax, ds:[esi+8]
        mov     ebx, ds:[esi+12]
        mov     es:[edi], eax
        mov     es:[edi+4], ebx
        add     edi, 10h
        sub     ecx, 4
        jnz     @b

        ; to insert NULL command to fit a slot requirement
        mov     es:[edi-8], 168F0000h
        mov     es:[edi-4], 168F0000h
        and     edi, QueueMask
        mov     es:[85C4h], edi     ; udpate write point
```

For more efficiency, in some special cases those data are fixed format, we use fixed packed

command. There are 15 different types of packed command, the detail format are listed in chapter "Command Queue Programming", here just list the name:

Type 1, BitBlt with source only.

Type 2, BitBlt with source and solid color pattern.

Type 3, BitBlt with source and monochrome pattern (mono mask).

Type 4, BitBlt with no source and no pattern.

Type 5, BitBlt with solid color pattern.

Type 6, BitBlt with monochrome pattern (mono mask).

Type 7, Mono source enhance color expansion without pattern.

Type 8, Mono source enhance color expansion with solid color pattern.

Type 9, Gradient Fill

Type 10, Destination Device Initialize (for large amount text string out initialization)

Type 11, Output each character with enhance color expansion.

Type 12, For general engine map.

Type 13, Output each character with color expansion

Type 14, First line of polyline

Type 15, Initialized Data for Monochrome Pattern Brush

These items can be treated as filling engine registers, do not concern their name very well. In experience, they used for special case with large benchmark performance issue. Sometime we combine several packet type to finish some work we need, such as implement BitBlt color pattern with type 4 and burst command mode. Whenever you have no good packet to use, you can choose type 12 or use single command mode instead.

Chap 7. The BitBlt Engine Programming

7-1 Bit Block Transfer (BitBlt)

BitBlt is an operation to transfer bits from a rectangle on a source device to a rectangle on a destination device. The transfer is controlled by a ternary raster operation value that specifies how corresponding bits from the source, destination, and pattern in a brush are combined to form the final bits in the destination.

In aspect of engine, BitBlt operations can be differentiated roughly to three kinds. If the source is color bitmap or operate with brush, you should use BitBlt command. If the source is mono-bitmap in pattern registers, you should use color expansion command. If the source is mono-bitmap in video memory, you should use extended color expansion command.

In aspect of source location, if the source is in system memory, we use the “move string instruction” (REP MOVS) to move the source data to the off-screen memory (CPU Blt Buffer) first, then complete ternary raster operation by hardware engine. This is called “CPU-driven BitBlt”

If the source and the destination both in on-screen memory, we should take care of the block transfer direction to avoid data lose. Integrated graphics controller could gain the advantage of its advanced engine design to solve the problems of memory overlapping during the block transfers. The only effort is to program the adequate parameter.

7-2 Ternary Raster Operation

Ternary raster-operation codes define how GDI combines the bits in a source bitmap with the bits in the destination bitmap. Each raster-operation code represents a Boolean operation in which the values of the pixels in the source, the selected brush, and the destination are combined.

ROP 3 Definition								
Pattern Bit	1	1	1	1	0	0	0	0
Source Bit	1	1	0	0	1	1	0	0
Destination Bit	1	0	1	0	1	0	1	0
Result Bit (ROP)	X	X	X	X	X	X	X	X

Hardware engine just accept the ternary raster operations (ROP3) with 256 operation codes from 0x00 to 0xFF. If you want to use binary raster operations, you must transfer binary raster operation codes (ROP2) from 01h to 10h to corresponding ROP3 code before filling into the command register.

The table below listed all ROP3 catalog:

ROP3 Catalog	ROP3
without Pattern Part, without Source Part, without Destination Part	00 FF
without Pattern Part, without Source Part, with Destination Part	55 AA
without Pattern Part, with Source Part, without Destination Part	33 CC
without Pattern Part,	11 22 44 66 77 88 99 BB DD EE

with Source Part, with Destination Part	
with Pattern Part, without Source Part, without Destination Part	0F F0
with Pattern Part, without Source Part, with Destination Part	05 0A 50 5A 5F A0 A5 AF F5 FA
with Pattern Part, with Source Part, without Destination Part	03 0C 30 3C 3F C0 C3 CF F3 FC
with Pattern Part, with Source Part, with Destination Part	01 02 04 06 07 08 09 0B 0D 0E 10 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 31 32 34 35 36 37 38 39 3A 3B 3D 3E 40 41 42 43 45 46 47 48 49 4A 4B 4C 4D 4E 4F 51 52 53 54 56 57 58 59 5B 5C 5D 5E 60 61 62 63 64 65 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 9A 9B 9C 9D 9E 9F A1 A2 A3 A4 A6 A7 A8 A9 AB AC AD AE B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BC BD BE BF C1 C2 C4 C5 C6 C7 C8 C9 CA CB CD CE D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EF F1 F2 F4 F6 F7 F8 F9 FB FD FE

7-3 BitBlt/Color Expansion/Enh-Color Expansion Engine Registers Format

The following table shows the register format for the general Graphics Engine functions. The general Graphics Engine functions are BitBlt, Color Expansion, Enhanced Color Expansion, Page Flipping and Clean Z-buffer.

D[31:24]	D[23:16]	D[15:08]	D[07:00]	I/O Address
Source Base Address				8200h
Reserved	AGP Base	Source Pitch		8204h
Source X		Source Y		8208h
Destination X		Destination Y		820Ch
Destination Base Address				8210h
Destination Height		Destination Pitch		8214h
Rectangular Height		Rectangular Width		8218h
Pattern FG (Foreground) Color				821Ch
Pattern BG (Background) Color				8220h
Source FG (Foreground) Color				8224h
Source BG (Background) Color				8228h
Mask3	Mask2	Mask1	Mask0	822Ch
Mask7	Mask6	Mask5	Mask4	8230h
Top Clipping		Left Clipping		8234h
Bottom Clipping		Right Clipping		8238h
Command				823Ch
Pattern 3	Pattern 2	Pattern 1	Pattern 0	8300h
Pattern 7	Pattern 6	Pattern 5	Pattern 4	8304h
Pattern 11	Pattern 10	Pattern 9	Pattern 8	8308h
Pattern 15	Pattern 14	Pattern 13	Pattern 12	830Ch
Pattern 19	Pattern 18	Pattern 17	Pattern 16	8310h
Pattern 23	Pattern 22	Pattern 21	Pattern 20	8314h

Pattern 27	Pattern 26	Pattern 25	Pattern 24	8318h
Pattern 31	Pattern 30	Pattern 29	Pattern 28	831Ch
Pattern 35	Pattern 34	Pattern 33	Pattern 32	8320h
Pattern 39	Pattern 38	Pattern 37	Pattern 36	8324h
Pattern 43	Pattern 42	Pattern 41	Pattern 40	8328h
Pattern 47	Pattern 46	Pattern 45	Pattern 44	832Ch
Pattern 51	Pattern 50	Pattern 49	Pattern 48	8330h
Pattern 55	Pattern 54	Pattern 53	Pattern 52	8334h
Pattern 59	Pattern 58	Pattern 57	Pattern 56	8338h
Pattern 63	Pattern 62	Pattern 61	Pattern 60	833Ch
Pattern 67	Pattern 66	Pattern 65	Pattern 64	8340h
Pattern 71	Pattern 70	Pattern 69	Pattern 68	8344h
Pattern 75	Pattern 74	Pattern 73	Pattern 72	8348h
Pattern 79	Pattern 78	Pattern 77	Pattern 76	834Ch
Pattern 83	Pattern 82	Pattern 81	Pattern 80	8350h
Pattern 87	Pattern 86	Pattern 85	Pattern 84	8354h
Pattern 91	Pattern 90	Pattern 89	Pattern 88	8358h
Pattern 95	Pattern 94	Pattern 93	Pattern 92	835Ch
Pattern 99	Pattern 98	Pattern 97	Pattern 96	8360h
Pattern 103	Pattern 102	Pattern 101	Pattern 100	8364h
Pattern 107	Pattern 106	Pattern 105	Pattern 104	8368h
Pattern 111	Pattern 110	Pattern 109	Pattern 108	836Ch
Pattern 115	Pattern 114	Pattern 113	Pattern 112	8370h
Pattern 119	Pattern 118	Pattern 117	Pattern 116	8374h
Pattern 123	Pattern 122	Pattern 121	Pattern 120	8378h
Pattern 127	Pattern 126	Pattern 125	Pattern 124	837Ch
Pattern 131	Pattern 130	Pattern 129	Pattern 128	8380h
Pattern 135	Pattern 134	Pattern 133	Pattern 132	8384h
Pattern 139	Pattern 138	Pattern 137	Pattern 136	8388h
Pattern 143	Pattern 142	Pattern 141	Pattern 140	838Ch
Pattern 147	Pattern 146	Pattern 145	Pattern 144	8390h
Pattern 151	Pattern 150	Pattern 149	Pattern 148	8394h
Pattern 155	Pattern 154	Pattern 153	Pattern 152	8398h
Pattern 159	Pattern 158	Pattern 157	Pattern 156	839Ch
Pattern 163	Pattern 162	Pattern 161	Pattern 160	83A0h
Pattern 167	Pattern 166	Pattern 165	Pattern 164	83A4h
Pattern 171	Pattern 170	Pattern 169	Pattern 168	83A8h
Pattern 175	Pattern 174	Pattern 173	Pattern 172	83ACh
Pattern 179	Pattern 178	Pattern 177	Pattern 176	83B0h
Pattern 183	Pattern 182	Pattern 181	Pattern 180	83B4h
Pattern 187	Pattern 186	Pattern 185	Pattern 184	83B8h
Pattern 191	Pattern 190	Pattern 189	Pattern 188	83BCh
Pattern 195	Pattern 194	Pattern 193	Pattern 192	83C0h
Pattern 199	Pattern 198	Pattern 197	Pattern 196	83C4h
Pattern 203	Pattern 202	Pattern 201	Pattern 200	83C8h
Pattern 207	Pattern 206	Pattern 205	Pattern 204	83CCh
Pattern 211	Pattern 210	Pattern 209	Pattern 208	83D0h
Pattern 215	Pattern 214	Pattern 213	Pattern 212	83D4h
Pattern 219	Pattern 218	Pattern 217	Pattern 216	83D8h
Pattern 223	Pattern 222	Pattern 221	Pattern 220	83DCh
Pattern 227	Pattern 226	Pattern 225	Pattern 224	83E0h
Pattern 231	Pattern 230	Pattern 229	Pattern 228	83E4h
Pattern 235	Pattern 234	Pattern 233	Pattern 232	83E8h
Pattern 239	Pattern 238	Pattern 237	Pattern 236	83ECh
Pattern 243	Pattern 242	Pattern 241	Pattern 240	83F0h

Pattern 247	Pattern 246	Pattern 245	Pattern 244	83F4h
Pattern 251	Pattern 250	Pattern 249	Pattern 248	83F8h
Pattern 255	Pattern 254	Pattern 253	Pattern 252	83FCh
Pattern 259	Pattern 258	Pattern 257	Pattern 256	8400h
Pattern 263	Pattern 262	Pattern 261	Pattern 260	8404h
Pattern 267	Pattern 266	Pattern 265	Pattern 264	8408h
Pattern 271	Pattern 270	Pattern 269	Pattern 268	840Ch
Pattern 275	Pattern 274	Pattern 273	Pattern 272	8410h
Pattern 279	Pattern 278	Pattern 277	Pattern 276	8414h
Pattern 283	Pattern 282	Pattern 281	Pattern 280	8418h
Pattern 287	Pattern 286	Pattern 285	Pattern 284	841Ch
Pattern 291	Pattern 290	Pattern 289	Pattern 288	8420h
Pattern 295	Pattern 294	Pattern 293	Pattern 292	8424h
Pattern 299	Pattern 298	Pattern 297	Pattern 296	8428h
Pattern 303	Pattern 302	Pattern 301	Pattern 300	842Ch
Pattern 307	Pattern 306	Pattern 305	Pattern 304	8430h
Pattern 311	Pattern 310	Pattern 309	Pattern 308	8434h
Pattern 315	Pattern 314	Pattern 313	Pattern 312	8438h
Pattern 319	Pattern 318	Pattern 317	Pattern 316	843Ch
Pattern 323	Pattern 322	Pattern 321	Pattern 320	8440h
Pattern 327	Pattern 326	Pattern 325	Pattern 324	8444h
Pattern 331	Pattern 330	Pattern 329	Pattern 328	8448h
Pattern 335	Pattern 334	Pattern 333	Pattern 332	844Ch
Pattern 339	Pattern 338	Pattern 337	Pattern 336	8450h
Pattern 343	Pattern 342	Pattern 341	Pattern 340	8454h
Pattern 347	Pattern 346	Pattern 345	Pattern 344	8458h
Pattern 351	Pattern 350	Pattern 349	Pattern 348	845Ch
Pattern 355	Pattern 354	Pattern 353	Pattern 352	8460h
Pattern 359	Pattern 358	Pattern 357	Pattern 356	8464h
Pattern 363	Pattern 362	Pattern 361	Pattern 360	8468h
Pattern 367	Pattern 366	Pattern 365	Pattern 364	846Ch
Pattern 371	Pattern 370	Pattern 369	Pattern 368	8470h
Pattern 375	Pattern 374	Pattern 373	Pattern 372	8474h
Pattern 379	Pattern 378	Pattern 377	Pattern 376	8478h
Pattern 383	Pattern 382	Pattern 381	Pattern 380	847Ch

BitBlt/Color Expansion/Enhance Color Expansion/Color to Monochrome engine register definition:

Name	Description											
Source Base Address	Port	8200h~8203h										
	Register Type	Read/Write										
	D[31:0]	Base Linear Address of Source Bitmap in Byte										
		Limit: Source Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary in BitBlt, Byte Boundary in Enhanced Color Expansion.										
AGP Base	Port	8206h~8207h										
	Register Type	Read/Write										
	D[15:10]	Reserved										
	D[9:0]	AGP Base Address in Byte.										
		Limit: The accessibility of the Bit[9:0] are controlled by graphic window size.										
		Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Size
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	4M
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	8M

		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	0	16M
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	0	0	32M
		R/W	R/W	R/W	R/W	R/W	R/W	0	0	0	0	64M
		R/W	R/W	R/W	R/W	R/W	0	0	0	0	0	128 M
		R/W	R/W	R/W	R/W	0	0	0	0	0	0	256 M
Source Pitch	Port	8204h~8205h										
	Register Type	Read/Write										
	D[15:14]	Reserved										
	D[13:0]	Row Pitch of Source Bitmap in Byte.										
		Limit: Double Word Boundary in BitBlt, Byte Boundary in Color Expansion and Enhanced Color Expansion.										
Rectangle Source Y	Port	8208h~8209h										
	Register Type	Read/Write										
	D[15:13]	Reserved										
	D[12:0]	Started Y Coordinate of Source Bitmap in Pixel										
		Limit: It is in the S12.0 format										
Rectangle Source X	Port	820Ah~820Bh										
	Register Type	Read/Write										
	D[15:13]	Reserved										
	D[12:0]	Started X Coordinate of Source Bitmap in Pixel										
		Limit: It is in the S12.0 format										
Rectangle Destination Y	Port	820Ch~820Dh										
	Register Type	Read/Write										
	D[15:13]	Reserved										
	D[12:0]	Started Y Coordinate of Destination Bitmap in Pixel										
		Limit: It is in the S12.0 format										
Rectangle Destination X	Port	820Eh~820Fh										
	Register Type	Read/Write										
	D[15:13]	Reserved										
	D[12:0]	Started X Coordinate of Destination Bitmap in Pixel										
		Limit: It is in the S12.0 format										
Destination Base Address	Port	8210h~8213h										
	Register Type	Read/Write										
	D[31:0]	Base Linear Address of Destination Bitmap in Byte										
		Limit: Destination Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary.										
Destination Pitch	Port	8214h~8215h										
	Register Type	Read/Write										
	D[15:14]	Reserved										
	D[13:0]	Row Pitch of Destination Bitmap in Byte										
		Limit: Double Word Boundary.										
Destination Height	Port	8216h~8217h										
	Register Type	Read/Write										
	D[15:13]	Reserved										
	D[12:0]	Device Height of Destination Bitmap in Pixel										
Rectangular Width	Port	8218h~8219h										
	Register Type	Read/Write										
	D[15:13]	Reserved										
	D[12:0]	Destination Rectangular Drawing Width of Bitmap in Pixel										
Rectangular	Port	821Ah~821Bh										

Height	Register Type	Read/Write
	D[15:13]	Reserved
	D[12:0]	Destination Rectangular Drawing Height of Bitmap in Pixel
Pattern Foreground Color	Port	821Ch~821Fh
	Register Type	Read/Write
	D[31:0]	Foreground Color Register of Pattern
Pattern Background Color	Port	8220h~8223h
	Register Type	Read/Write
	D[31:0]	Background Color Register of Pattern
Source Foreground Color	Port	8224h~8227h
	Register Type	Read/Write
	D[31:0]	Foreground Color Register of Source
Source Background Color	Port	8228h~822Bh
	Register Type	Read/Write
	D[31:0]	Background Color Register of Source
Mono Mask Register	Port	822Ch~8233h
	Register Type	Read/Write
	D[63:0]	Monochrome Mask Register of Pattern
Left Clipping	Port	8234h~8235h
	Register Type	Read/Write
	D[15:0]	Left Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Top Clipping	Port	8236h~8237h
	Register Type	Read/Write
	D[15:0]	Top Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Right Clipping	Port	8238h~8239h
	Register Type	Read/Write
	D[15:0]	Right Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Bottom Clipping	Port	823Ah~823Bh
	Register Type	Read/Write
	D[15:0]	Bottom Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Command Register	Port	823Ch~823Fh
	Register Type	Read/Write
	D31	Scan Line Trigger Function 0: Disable 1: Enable
	D30	M-BitBlt Control 0: M-BitBlt function turn off 1: Fire engine need to wait for MframeRdyN is low
	D29	ID control 0: No need to count the global ID 1: Send a signal (M2dFireEnd) to count global ID
	D[28:27]	Counter decrease indictor 00: All counter no decrease 01: Counter 1 will be decrease 10: Counter 2 will be decrease 11: Counter 3 will be decrease
	D26	Rectangular Clipping Merge Control 0: Merge clipping bound with screen bound 1: Do not merge clipping bound with screen bound
	D25	Destination Location Control 0: Destination is to video memory 1: Destination is to AGP memory no define in 310

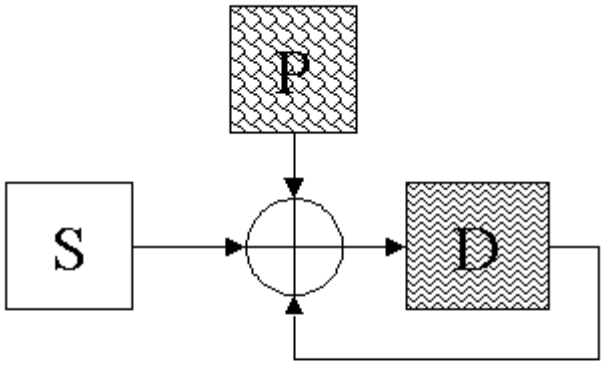
	D24	Scan Line Trigger Function for CRT1 or CRT2 0: CRT1 1: CRT2
	D[23:21]	Reserved
	D20	Transparent Control 0: Opaque 1: Transparent
	D19	Reserved
	D18	Rectangular Clipping Control 0: Disable rectangular clipping logic 1: Enable rectangular clipping logic
	D[17:16]	Enhanced graphics mode selection Bit[1:0] 00: 8 bpp 256 color mode 01: 16 bpp 64k color mode 10: 32 bpp True color mode 11: Reserved
	D[15:8]	256 Raster Operations
	D[7:6]	Pattern select Bit[1:0] 00: Pattern is from pattern foreground color register 01: Pattern is from pattern register 10: Pattern is from monochrome mask register 11: Reserved
	D[5:4]	Source select Bit[1:0] 00: Source is from video memory 01: Source is from CPU-driven BitBlt buffer 10: Source is from AGP memory 11: Reserved
	D[3:0]	Command type select Bit[3:0] 0000: BitBlt 0001: Color Expansion 0010: Enhanced Color Expansion 0011: Multiple Scanline 0100: Line Draw 0101: Trapezoid Fill 0110: Transparent BitBlt 0111: Alpha Blending 1000: 3D Function 1001: Clean Z-buffer 1010: Gradient Fill 1011: Stretch BitBlt 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved
Pattern Register n	Port	8300h~833Fh for 256-color 8300h~837Fh for high-color 8300h~83FFh for true-color 8300h~847Fh for color expansion source
	Register Type	Read/Write
	D[7:0]	For BitBlt and Enhanced Color Expansion function, these registers store the 8x8 color pattern bitmap. For Color Expansion function, these register (till 847Fh) stored monochrome source.
		Limit: Double Word Boundary, Double Word Access Only

For Color Expansion function, the registers from 8300h to 847Fh store the monochrome bitmap, thus it can expand 3072 pixels at one command.

7-4 Programming XGI Volari Family BitBlt Engine

XGI Volari Family support BitBlt engine to block transfer data from frame buffer to frame buffer. That is, XGI Volari Family do not support CPUBlt engine to do ROP3 when CPU move data to frame buffer. Whenever you want to BitBlt a rectangle from system memory to frame buffer, you must copy the source to a buffer on frame buffer then call BitBlt engine.

Before we discuss the programming problem, there are something should be talked at first. The "Source", "Destination", and "Pattern" in XGI Volari Family BitBlt engine are orthogonal. All operation must have "Destination" part to write result (if not, we can bypass the action). The Destination can be treated as one part as operation source, and the "Source" part, and the pattern, looked like the figure below:

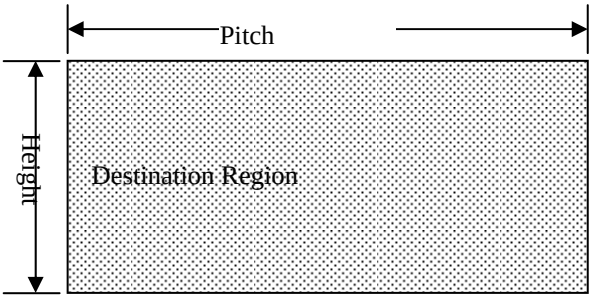


What part should join into the operation is decided by ROP3, look the ROP description. For example, ROP 0xCC have no pattern part, the destination will not be read back on operation.

7-5 Destination Device Region Definition

Before you draw an action, you must define which region you can draw. The parameter to definition a drawing region are listed below:

Parameter	Register	Description
Height	8216	The pixel height of the destination region
Pitch	8214	The byte distance between two vertical neighbor pixels.
Base Address	8210	The address of the destination region. Must be dword alignment. Suggest align to 128 bits.
Byte per pixel	in command bit field	defined in engine command 823C. Source and Destination region must be equal in Bitblt engine.



If the option "Automatic clipping merge" do not disabled, any drawing by hardware engine cannot draw out of the defined rectangle.

7-6 The Source Device Region Definition

As the destination device, the source must define the device region. The difference between source device and destination device is that source device does not have the height. That means you do not need to consider the region limit of source, because it is "source device", never be drawn when the device is source device.

The associated information is stored in

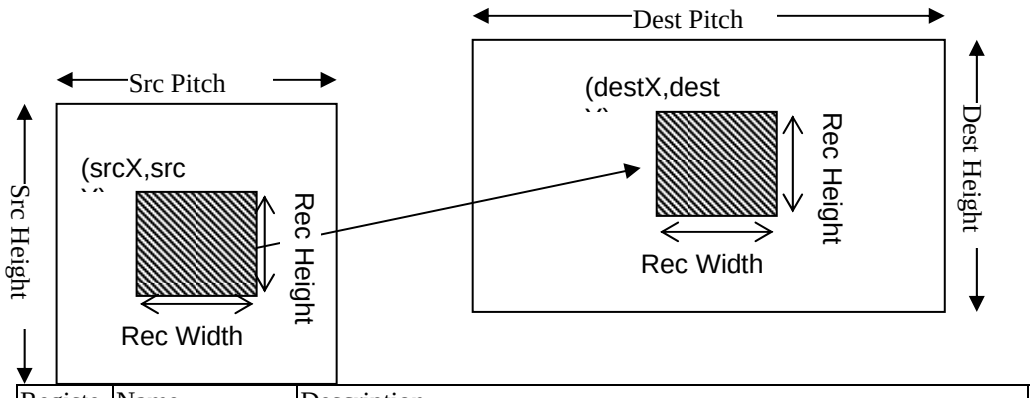
Parameter	Register	Description
Pitch	8204	The byte distance between two vertical neighbor pixels.
Base Address	8200	The address of the destination region. Must be dword alignment. Suggest align to 128 bits.
Byte per pixel	in command bit field	defined in engine command 823C. Source and Destination region must be equal in Bitblt engine.

7-7 The Coordinate and bit-block transfer rectangle

Look the figure below:

There are two pairs of coordinates, one is for the source rectangle left-up corner coordinate, and the other is for the destination rectangle left-up corner. The source pitch and destination pitch can be different on XGI Volari Family 2D engine.

To set these information, you must programming the values in those registers listed below:



Register	Name	Description

8208	Src Y	The Y coordinate of source rectangle up-left corner
820A	Src X	The X coordinate of source rectangle up-left corner
820C	Dest Y	The Y coordinate of destination rectangle up-left corner
820E	Dest X	The X coordinate of destination rectangle up-left corner
8218	Rec Width	The width of rectangle
821A	Rec Height	The height of rectangle

For example, we can see the sample to move a rectangle without pattern settng. The sample use the packet type 1, source copy.

```
;------------------------------------------------------------------------
; BS_SVNP
; Source Is Frame Buffer
; No Pattern
; ds:[0] = lpDstDev
; fs:[0] = lpSrcDev
;------------------------------------------------------------------------
align 16
PLABEL    BS_SVNP
BeginBltEntry SVNP
sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE01_LENGTH
jl     SVNP_WaitCmdQueue
SVNP_WaitCmdQueueDone:

mov  edi, gs:[0].sd_dwQueueWritePort

mov  eax, dptr fs:[0].deBits    ; eax = the source device base address
mov  ecx, dptr fs:[0].deDeltaScan   ; ecx = the pitch of source device

mov  dptr gs:[edi+QOFFSET].PkC01_dwHeader0, PKT_TYPE01_HEADER0
mov  dptr gs:[edi+QOFFSET].PkC01_dwHeader1, PKT_TYPE01_HEADER1
mov  dptr gs:[edi+QOFFSET].PkC01_dwSrcBaseAddr, eax
mov  dptr gs:[edi+QOFFSET].PkC01_wSrcPitch, ecx

mov  eax, dptr wSrcYOrg      ; eax = [SrcX:SrcY]
mov  ecx, dptr wDstYOrg      ; ecx = [DestX:DestY]
mov  ebx, dptr ds:[0].deBits  ; ebx = the base address of destination device

mov  edx, dptr ds:[0].deHeight
shl    edx, 16
or     edx, ds:[0].deDeltaScan ; edx = [DestHeight:DestPitch]

mov  dptr gs:[edi+QOFFSET].PkC01_wSrcY, eax
mov  dptr gs:[edi+QOFFSET].PkC01_wDstY, ecx
mov  dptr gs:[edi+QOFFSET].PkC01_dwDstBaseAddr, ebx
mov  dptr gs:[edi+QOFFSET].PkC01_wDstPitch, edx

mov  eax, dptr wYExt
ror    eax, 16

mov  ebx, dwCurColorMode ; each color depth must program int register.
mov  bh, bptr dwRop+2        ;

; or    ebx, CMD_BITBLT+CMD_SRC_VRAM; or ebx,0

mov  dptr gs:[edi+QOFFSET].PkC01_wRecWidth, eax
mov  dptr gs:[edi+QOFFSET].PkC01_dwCommand, ebx
mov  dptr gs:[edi+QOFFSET].PkC01_dwNullData, PKT_NULLCMD
mov  dptr gs:[edi+QOFFSET].PkC01_dwNullData+4, PKT_NULLCMD
```

```
add    edi, PKT_TYPE01_LENGTH
and    edi, gs:[0].sd_dwQueueMask

mUpdateWritePort edi   ; write edi to write point

and    bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
SVNP_WaitCmdQueue:
mQueryCmdQueueLength <PKT_TYPE01_LENGTH>
jmp    SVNP_WaitCmdQueueDone
EndBltEntry

; In this sample there are some macro.
; mQueryCmdQueueLength is an macro to query the actual command queue available queue
length.
; mUpdateWritePort is an macro to update the write point of command queue.
```

If there is no need to support source part for a given ROP, you need the destination device information only. With the same programming way, you can use packet type 4,5,or 6 to program this.

The programming sample for destination only without source and pattern are listed below:

```
; BS_NSXXX, for No Source
;-------------------------------------------------------------------------
; BS_NSNP
; No Source
; No Pattern
;-------------------------------------------------------------------------
PLABEL    BS_NSNP
BeginBltEntry NSNP
; INT3        <'NSNP'>

BLT_NSNP_CMD = CMD_BITBLT

; movgs, wShareDataSelector
sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE04_LENGTH
jl       NSNP_WaitCmdQueue
NSNP_WaitCmdQueueDone:
mov   edi, gs:[0].sd_dwQueueWritePort

mov   ebx, dwCurColorMode
mov   bh, byte ptr dwRop+2

mov   edx, dword ptr ds:[0].deHeight - 2
and    edx, 0ffff0000h
or      edx, dword ptr ds:[0].deDeltaScan

mov   esi, dword ptr wDstYOrg
mov   ecx, dword ptr ds:[0].deBits

mov   eax, dword ptr wYExt
ror     eax, 16

mov   dptr gs:[edi+QOFFSET].PkC04_dwHeader0, PKT_TYPE04_HEADER0
mov   dptr gs:[edi+QOFFSET].PkC04_dwHeader1, PKT_TYPE04_HEADER1
mov   dptr gs:[edi+QOFFSET].PkC04_wDstY, esi
mov   dptr gs:[edi+QOFFSET].PkC04_dwDstBaseAddr, ecx
```

```
mov   dptr gs:[edi+QOFFSET].PkC04_wDstPitch, edx
mov   dptr gs:[edi+QOFFSET].PkC04_wRecWidth, eax
mov   dptr gs:[edi+QOFFSET].PkC04_dwCommand, ebx
mov   dptr gs:[edi+QOFFSET].PkC04_dwNullData, PKT_NULLCMD

add    edi, PKT_TYPE04_LENGTH
and    edi, gs:[0].sd_dwQueueMask
mUpdateWritePort        edi

and    bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
NSNP_WaitCmdQueue:
mQueryCmdQueueLength      PKT_TYPE04_LENGTH
jmp    NSNP_WaitCmdQueueDone

EndBltEntry
```

We will describe how to implement the programming with pattern.

7-8 Programming Pattern

There are several types of patterns, brushes, any term. They can separate into three major types:

Solid Color Pattern: The pattern is a solid color.

Monochrome Pattern: The pattern is a 8×8 one bpp bitmap (thus it have 8 bytes), with foreground color and background color.

Color Pattern: The pattern is a 8×8 colored bitmap with the same color depth to destination target. The size is 64 bytes when 256 color mode, 128 bytes when high-color mode, and 256 bytes when 32 bits true color mode.

You can decide the pattern type in the bit D[7:6] of the command register 823C. The values are listed below:

Bit Value	Descrption
00	Pattern is from pattern foreground color register
01	Pattern is from pattern register
10	Pattern is from monochrome mask register
11	Reserved

The 8×8 bitmap of pattern is repeat drawing and the up-left corner is always match the position (8m,8n) of destination device.

The monochrome mask registers are 8 bytes size registers, 822Ch and 8230h. Each byte is a row of 8×8 bitmap, with little ending order. The pattern registers are a large area of storage for 8×8 pattern bitmap, and the bit-per-pixel is decided by register 823C D[17:16]. To use monochrome mask on engine, the pattern foreground color (821C) and the pattern background color (8220) is used to expand the bit 1 and 0 on mask register.

To program the pattern register with MMIO mode, no byte/word access in the register. In general, we do not need to program engine with MMIO mode, the limitation is minor to our program.

The sample is to brush an solid color to destination without source:

```
; BS_NSXXX, for No Source
;-----------------------------------------------------------------------
; BS_NSCSB
; No Source
```

```
; Color Solid Brush
; fs:esi = lpBrush
;---------------------------------------------------------------------
align 16
PLABEL    BS_NSCSB
BeginBltEntry NSCSB
; INT3        <'NSCSB'>

; movgs, wShareDataSelector

sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE05_LENGTH
jl     NSCSB_WaitCmdQueue
NSCSB_WaitCmdQueueDone:


mov   edi, gs:[0].sd_dwQueueWritePort
mov   ecx, dword ptr wDstYOrg
mov   edx, dword ptr ds:[0].deBits
mov   eax, dword ptr fs:[esi].dp8BrushBits ; FGColor
mov   ebx, dwCurColorMode


mov   dptr gs:[edi+QOFFSET].PkC05_dwHeader0, PKT_TYPE05_HEADER0
mov   dptr gs:[edi+QOFFSET].PkC05_dwHeader1, PKT_TYPE05_HEADER1
mov   dptr gs:[edi+QOFFSET].PkC05_wDstY, ecx
mov   dptr gs:[edi+QOFFSET].PkC05_dwDstBaseAddr, edx

mov   ecx, dword ptr ds:[0].deHeight - 2
and   ecx, 0ffff0000h
or    ecx, dword ptr ds:[0].deDeltaScan

mov   edx, dptr wYExt
ror   edx, 16
mov   bh, bptr dwRop+2


mov   dptr gs:[edi+QOFFSET].PkC05_dwPatFGColor, eax
mov   dptr gs:[edi+QOFFSET].PkC05_wDstPitch, ecx
mov   dptr gs:[edi+QOFFSET].PkC05_wRecWidth, edx
mov   dptr gs:[edi+QOFFSET].PkC05_dwCommand, ebx

add    edi, PKT_TYPE05_LENGTH
and    edi, gs:[0].sd_dwQueueMask

mUpdateWritePort edi

and    bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
NSCSB_WaitCmdQueue:
mQueryCmdQueueLength <PKT_TYPE05_LENGTH>
jmp    NSCSB_WaitCmdQueueDone
EndBltEntry
```

The sample is to brush a monochrome pattern to destination without source:

```
;---------------------------------------------------------------------
; BS_NSMP
```

```
; No Source
; Monochrome Pattern
; fs:[esi] = lpPBrush
; Mono Pattern , 0 mapped to lpDrawMode->TextColor
;              1 mapped to lpDrawMode->bkColor
;-----------------------------------------------------------------------
align 16
PLABEL    BS_NSMP
BeginBltEntry NSMP
; INT3        <'NSMP'>
; mov gs, wShareDataSelector
sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE06_LENGTH
jl     NSMP_WaitCmdQueue
NSMP_WaitCmdQueueDone:

mov  edi, gs:[0].sd_dwQueueWritePort

mov  eax, dptr wDstYOrg
mov  edx, dptr ds:[0].deBits

mov  dptr gs:[edi+QOFFSET].PkC06_dwHeader0, PKT_TYPE06_HEADER0
mov  dptr gs:[edi+QOFFSET].PkC06_dwHeader1, PKT_TYPE06_HEADER1
mov  dptr gs:[edi+QOFFSET].PkC06_wDstY, eax
mov  dptr gs:[edi+QOFFSET].PkC06_dwDstBaseAddr, edx

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask

mov  ecx, dptr ds:[0].deHeight -2
and    ecx, 0ffff0000h
or     ecx, dptr ds:[0].deDeltaScan

mov  ebx, dptr wYExt
ror    ebx, 16

mov  dptr gs:[edi+QOFFSET-10h].PkC06_wDstPitch, ecx
mov  dptr gs:[edi+QOFFSET-10h].PkC06_wRecWidth, ebx

mov  al, bptr fs:[esi].dp8BrushMono+8
mov  dl, bptr fs:[esi].dp8BrushMono+24
mov  ah, bptr fs:[esi].dp8BrushMono+12
mov  dh, bptr fs:[esi].dp8BrushMono+28

ror    eax, 16
ror    edx, 16

mov  al, bptr fs:[esi].dp8BrushMono
mov  dl, bptr fs:[esi].dp8BrushMono+16
mov  ah, bptr fs:[esi].dp8BrushMono+4
mov  dh, bptr fs:[esi].dp8BrushMono+20

movzx       esi, off_lpDrawMode
mov  fs, seg_lpDrawMode

mov  ebx, fs:[esi].bkColor
mov  ecx, fs:[esi].TextColor

mov  gs:[edi+QOFFSET-10h].PkC06_dwPatFGColor, ebx
```

```
mov  gs:[edi+QOFFSET-10h].PkC06_dwPatBGColor, ecx

add   edi, 10h
and   edi, gs:[0].sd_dwQueueMask

mov  ebx, dwCurColorMode
mov  bh, bptr dwRop+2
or     ebx, CMD_BITBLT+CMD_PAT_MONOMASK

; eax = BrushMono 12:08:04:00
; edx = BrushMono 28:24:20:16
; ebx = command

mov  dptr gs:[edi+QOFFSET-20h].PkC06_bMask, eax
mov  dptr gs:[edi+QOFFSET-20h].PkC06_bMask+4, edx

mov  dptr gs:[edi+QOFFSET-20h].PkC06_dwCommand, ebx
mov  dptr gs:[edi+QOFFSET-20h].PkC06_dwNullData, PKT_NULLCMD

add   edi, 10h
and   edi, gs:[0].sd_dwQueueMask

mUpdateWritePort       edi

and   bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
NSMP_WaitCmdQueue:
mQueryCmdQueueLength <PKT_TYPE06_LENGTH>
jmp   NSMP_WaitCmdQueueDone
EndBltEntry
```

The sample is to brush a color pattern to destination without source:

```
;---------------------------------------------------------------------
; BS_NSCP
; No Source
; Color Pattern
; fs:esi = lpPBrush
;---------------------------------------------------------------------
align 16
PLABEL    BS_NSCP
BeginBltEntry NSCP
; INT3       <'NSCP'>

; movgs, wShareDataSelector
mov  ecx, dwCurPatBytes
add   ecx, PKT_TYPE04_LENGTH+8+8 ; burst header
sub   gs:[0].sd_dwCmdQueueLen, ecx
jl      NSCP_WaitCmdQueue
NSCP_WaitCmdQueueDone:

; Using Burst Packet Mode to assign the pattern register
mov  edi, gs:[0].sd_dwQueueWritePort
mov  ecx, dwCurPatBytes
mov  eax, ecx
shr    eax, 2; dword count
add   eax, PKT_BURST_CMD_HEADER1
add   esi, OFFSET dp8BrushBits
```

```
Header0 = PKT_BURST_CMD_HEADER0+REG_ENGINE_PATTERN
mov  gs:[edi+QOFFSET].PkBC_dwHeader0, Header0
mov  gs:[edi+QOFFSET].PkBC_dwHeader1, eax
@@:
mov  eax, fs:[esi]
mov  edx, fs:[esi+04h]
mov  dptr gs:[edi+QOFFSET+8], eax
mov  dptr gs:[edi+QOFFSET+0Ch], edx

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  eax, fs:[esi+08h]
mov  edx, fs:[esi+0Ch]
mov  dptr gs:[edi+QOFFSET], eax
mov  dptr gs:[edi+QOFFSET+4], edx

add  esi, 10h
sub  ecx, 10h
jnz  @b
; fill pattern done, must fill the last 2 dword NULL command
mov  dptr gs:[edi+QOFFSET+08h], PKT_NULLCMD
mov  dptr gs:[edi+QOFFSET+0Ch], PKT_NULLCMD
add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  eax, dptr wDstYOrg
mov  ecx, dptr ds:[0].deBits

mov  dptr gs:[edi+QOFFSET].PkC04_dwHeader0, PKT_TYPE04_HEADER0
mov  dptr gs:[edi+QOFFSET].PkC04_dwHeader1, PKT_TYPE04_HEADER1
mov  dptr gs:[edi+QOFFSET].PkC04_wDstY, eax
mov  dptr gs:[edi+QOFFSET].PkC04_dwDstBaseAddr, ecx

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  edx, dptr ds:[0].deHeight - 2
and  edx, 0ffff0000h
or   edx, dptr ds:[0].deDeltaScan

mov  eax, dptr wYExt
ror  eax, 16

mov  ebx, dwCurColorMode
mov  bh, bptr dwRop+2
or   ebx, CMD_PAT_PATREG+CMD_BITBLT

mov  dptr gs:[edi+QOFFSET-10h].PkC04_wDstPitch, edx
mov  dptr gs:[edi+QOFFSET-10h].PkC04_wRecWidth, eax
mov  dptr gs:[edi+QOFFSET-10h].PkC04_dwCommand, ebx
mov  dptr gs:[edi+QOFFSET-10h].PkC04_dwNullData, PKT_NULLCMD

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mUpdateWritePort edi
```

```
and    bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
NSCP_WaitCmdQueue:
mQueryCmdQueueLength <ecx>
jmp    NSCP_WaitCmdQueueDone
EndBltEntry
```

The sample is to brush a solid color brush to destination with source:

```
;-------------------------------------------------------------------------
; BS_SVCSB
; Source Is Frame Buffer
; Color Solid Brush
; ds:[0] = lpDstDev
; fs:[0] = lpSrcDev
; gs = seg_lpPBrush
;-------------------------------------------------------------------------
align 16
PLABEL    BS_SVCSB
BeginBltEntry SVCSB
; INT3        <'SVCSB'>

; mov gs, wShareDataSelector
sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE02_LENGTH
jl       SVCSB_WaitCmdQueue
SVCSB_WaitCmdQueueDone:

mov   edi, gs:[0].sd_dwQueueWritePort
mov   eax, dptr fs:[0].deBits
mov   edx, dptr fs:[0].deDeltaScan

mov   dptr gs:[edi+QOFFSET].PkC02_dwHeader0, PKT_TYPE02_HEADER0
mov   dptr gs:[edi+QOFFSET].PkC02_dwHeader1, PKT_TYPE02_HEADER1
mov   dptr gs:[edi+QOFFSET].PkC02_dwSrcBaseAddr, eax
mov   dptr gs:[edi+QOFFSET].PkC02_wSrcPitch, edx

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask

mov   eax, dptr wSrcYOrg
mov   ecx, dptr wDstYOrg
mov   edx, dptr ds:[0].deBits
mov   esi, dptr ds:[0].deHeight
shl    esi, 16
or      esi, ds:[0].deDeltaScan

mov   dptr gs:[edi+QOFFSET-10h].PkC02_wSrcY, eax
mov   dptr gs:[edi+QOFFSET-10h].PkC02_wDstY, ecx
mov   dptr gs:[edi+QOFFSET-10h].PkC02_dwDstBaseAddr, edx
mov   dptr gs:[edi+QOFFSET-10h].PkC02_wDstPitch, esi

mov   fs, lpPBrush.sel
movzx        esi, lpPBrush.off

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask
```

```
mov  eax, dptr wYExt
mov  ebx, dwCurColorMode
mov  ecx, dptr fs:[esi].dp8BrushBits

ror    eax, 16
mov  bh, bptr dwRop+2
; or    ebx, CMD_BITBLT+CMD_PAT_FGCOLOR ; or ebx, 0

mov  dptr gs:[edi+QOFFSET-20h].PkC02_wRecWidth, eax
mov  dptr gs:[edi+QOFFSET-20h].PkC02_dwPatFGColor, ecx
mov  dptr gs:[edi+QOFFSET-20h].PkC02_dwCommand, ebx
mov  dptr gs:[edi+QOFFSET-20h].PkC02_dwNullData, PKT_NULLCMD

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask

mUpdateWritePort edi

and    bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
SVCSB_WaitCmdQueue:
mQueryCmdQueueLength <PKT_TYPE02_LENGTH>
jmp   SVCSB_WaitCmdQueueDone
EndBltEntry
```

The sample is to brush a monochrome pattern to destination with source:

```
;---------------------------------------------------------------------
; BS_SVMP
; Source Is Frame Buffer
; Monochrome Pattern
;---------------------------------------------------------------------
align 16
PLABEL    BS_SVMP
BeginBltEntry SVMP
; INT3        <'SVMP'>

; movgs, wShareDataSelector
sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE03_LENGTH
jl      SVMP_WaitCmdQueue
SVMP_WaitCmdQueueDone:

mov  edi, gs:[0].sd_dwQueueWritePort

mov  eax, dptr fs:[0].deBits
mov  edx, fs:[0].deDeltaScan

mov  dptr gs:[edi+QOFFSET].PkC03_dwHeader0, PKT_TYPE03_HEADER0
mov  dptr gs:[edi+QOFFSET].PkC03_dwHeader1, PKT_TYPE03_HEADER1
mov  dptr gs:[edi+QOFFSET].PkC03_dwSrcBaseAddr, eax
mov  dptr gs:[edi+QOFFSET].PkC03_wSrcPitch, edx

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask
```

```
mov  eax, dptr wSrcYOrg
mov  edx, dptr wDstYOrg
mov  ecx, dptr ds:[0].deBits
mov  ebx, dptr ds:[0].deHeight
shl  ebx, 16
or   ebx, dptr ds:[0].deDeltaScan

mov  dptr gs:[edi+QOFFSET-10h].PkC03_wSrcY, eax
mov  dptr gs:[edi+QOFFSET-10h].PkC03_wDstY, edx
mov  dptr gs:[edi+QOFFSET-10h].PkC03_dwDstBaseAddr, ecx
mov  dptr gs:[edi+QOFFSET-10h].PkC03_wDstPitch, ebx

mov  fs, lpDrawMode.sel
movzx     esi, lpDrawMode.off

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  eax, dptr wYExt
ror  eax, 16

mov  edx, fs:[esi].bkColor
mov  ecx, fs:[esi].TextColor

mov  dptr gs:[edi+QOFFSET-20h].PkC03_wRecWidth, eax
mov  dptr gs:[edi+QOFFSET-20h].PkC03_dwPatFGColor, edx
mov  dptr gs:[edi+QOFFSET-20h].PkC03_dwPatBGColor, ecx

mov  fs, lpPBrush.sel
movzx     esi, lpPBrush.off

mov  bl, bptr fs:[esi].dp8BrushMono+8
mov  cl, bptr fs:[esi].dp8BrushMono+24
mov  bh, bptr fs:[esi].dp8BrushMono+12
mov  ch, bptr fs:[esi].dp8BrushMono+28
ror  ebx, 16
ror  ecx, 16
mov  bl, bptr fs:[esi].dp8BrushMono+0
mov  cl, bptr fs:[esi].dp8BrushMono+16
mov  bh, bptr fs:[esi].dp8BrushMono+4
mov  ch, bptr fs:[esi].dp8BrushMono+20

mov  dptr gs:[edi+QOFFSET-20h].PkC03_bMask, ebx

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  ebx, dwCurColorMode
or   ebx, CMD_BITBLT+CMD_PAT_MONOMASK
mov  bh, bptr dwRop+2

mov  dptr gs:[edi+QOFFSET-30h].PkC03_bMask+4, ecx
mov  dptr gs:[edi+QOFFSET-30h].PkC03_dwCommand, ebx
mov  dptr gs:[edi+QOFFSET-30h].PkC03_dwNullData, PKT_NULLCMD
mov  dptr gs:[edi+QOFFSET-30h].PkC03_dwNullData+4, PKT_NULLCMD

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask
```

```
mUpdateWritePort edi

and    bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
SVMP_WaitCmdQueue:
mQueryCmdQueueLength <PKT_TYPE03_LENGTH>
jmp   SVMP_WaitCmdQueueDone
EndBltEntry
```

The sample is to brush a color pattern to destination with source:

```
;---------------------------------------------------------------------
; BS_SVCP
; Source Is Frame Buffer
; Color Pattern
;---------------------------------------------------------------------
align 16
PLABEL    BS_SVCP
BeginBltEntry SVCP
; INT3        <'SVCP'>
mov   ecx, dwCurPatBytes
add    ecx, PKT_TYPE01_LENGTH + 8 - 8
; add 2 dword for burst header
; remove 2 dword for the merge NULL

; movgs, wShareDataSelector
sub    gs:[0].sd_dwCmdQueueLen, ecx
jl       SVCP_WaitCmdQueue
SVCP_WaitCmdQueueDone:

mov   fs,  lpPBrush.sel
movzx        esi, lpPBrush.off
add    esi, OFFSET dp8BrushBits

mov   edi, gs:[0].sd_dwQueueWritePort

mov   eax, dwCurPatBytes
mov   ecx, dwCurPatBytes
shr    eax, 2
add    eax, PKT_BURST_CMD_HEADER1

Header0 =  PKT_BURST_CMD_HEADER0+REG_ENGINE_PATTERN
mov   dptr gs:[edi+QOFFSET].PkBC_dwHeader0, Header0
mov   dptr gs:[edi+QOFFSET].PkBC_dwHeader1, eax
@@:
mov   eax, fs:[esi]
mov   edx, fs:[esi+4]
mov   dptr gs:[edi+QOFFSET+8], eax
mov   dptr gs:[edi+QOFFSET+12], edx

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask

mov   eax, fs:[esi+8]
mov   edx, fs:[esi+12]
```

```
mov  dptr gs:[edi+QOFFSET], eax
mov  dptr gs:[edi+QOFFSET+4], edx
add  esi, 10h
sub  ecx, 10h
jnz  @b

mov  dptr gs:[edi+QOFFSET+8].PkC01_dwHeader0, PKT_TYPE01_HEADER0
mov  dptr gs:[edi+QOFFSET+8].PkC01_dwHeader1, PKT_TYPE01_HEADER1

mov  fs, seg_lpSrcDev

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  eax, dptr fs:[0].deBits
mov  ecx, fs:[0].deDeltaScan
mov  edx, dptr wSrcYOrg
mov  esi, dptr wDstYOrg

mov  dptr gs:[edi+QOFFSET-8].PkC01_dwSrcBaseAddr, eax
mov  dptr gs:[edi+QOFFSET-8].PkC01_wSrcPitch, ecx
mov  dptr gs:[edi+QOFFSET-8].PkC01_wSrcY, edx
mov  dptr gs:[edi+QOFFSET-8].PkC01_wDstY, esi

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mov  eax, dptr ds:[0].deBits

mov  ecx, dptr ds:[0].deHeight
shl  ecx, 16
or   ecx, ds:[0].deDeltaScan

mov  edx, dptr wYExt
ror  edx, 16

mov  ebx, dwCurColorMode
or   ebx, CMD_BITBLT+CMD_PAT_PATREG
mov  bh, bptr dwRop+2

mov  dptr gs:[edi+QOFFSET-18h].PkC01_dwDstBaseAddr, eax
mov  dptr gs:[edi+QOFFSET-18h].PkC01_wDstPitch, ecx
mov  dptr gs:[edi+QOFFSET-18h].PkC01_wRecWidth, edx
mov  dptr gs:[edi+QOFFSET-18h].PkC01_dwCommand, ebx

add  edi, 10h
and  edi, gs:[0].sd_dwQueueMask

mUpdateWritePort edi

and  bptr ds:[0].deFlags, not BUSY
mBltExitAndUnExcludeCursor TRUE

align 16
SVCP_WaitCmdQueue:
mQueryCmdQueueLength <ecx>
jmp  SVCP_WaitCmdQueueDone
EndBltEntry
```

7-9 CPUBlt Buffer

In previous sections, we talked about that source can be frame buffer, therefore we need a method to BitBlt those source on system memory. For performance issue, we do not hope whenever we BitBlt a block on system memory to frame buffer we must wait for engine idle at first.

There are three types of special frame buffer surface, named *CPUBlt Buffer*. Driver must original these buffers in a array, then read back the information on MMIO 85CC will know how many buffers are in command queue.

For example, we describe how CPUBlt buffer work:

Assume there are n buffers for CPUBlt buffer type 1. At initial time, you can read the "used buffer count of type 1" from the byte MMIO 85CC D[7:0] is zero, means there is no command with source type as CPUBlt buffer type 1. When you BitBlt a surface on system memory to frame buffer, you must copy it to one of the CPUBlt buffer. Write 0x11 to MMIO 85CC register, the 85CC D[7:0] value will increase to notify a new buffer is used. Then when you fire an command, you must program the D[28:27] of engine register 823C as 01 to identify when engine finish the command, engine must decrease the value of MMIO 85CC D[7:0] to notify software there is a buffer released.

CPUBlt Buffer Identify in 823C

D[28:27] Value	Description
00	All counter no decrease
01	Counter 1 (85CC [7:0]) decrease
10	Counter 2 (85CC [15:8]) decrease
11	Counter 3 (85CC [23:16]) decrease

CPUBlt Buffer Identify in 823C

Value write to 85CC	Description
0x00000011	Counter 1 (85CC [7:0]) increase
0x00000022	Counter 2 (85CC [15:8]) increase
0x00000033	Counter 3 (85CC [23:16]) increase

Therefore, you must configure your CPUBlt buffer as a ring queue with N (<255) same size buffers. You can control how many buffer is not wait for engine process, then you can wait for CPUBlt buffer free instead of wait for engine idle to synchronize frame buffer and CPU read/write data.

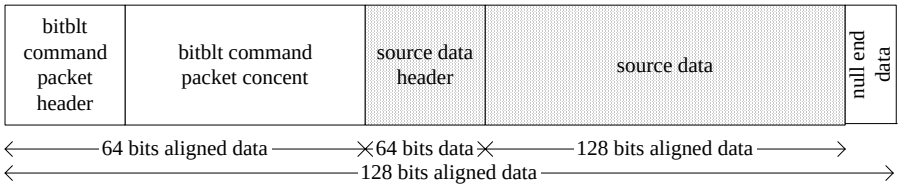
7-10 BitBlt with the Source from Command Queue[1]

To support CPUBlt function, to keep command order and data coherence is a major issue. Therefore, if we put blt data in command queue we can solve the problem and get better performance.

i.) Command and data order as using source in queue

When we using the source in queue feature, the data sequence is different from the original bitblt. The original command use data from and to frame buffer, therefore, when command is fetched by 2D accelerator, the data must be ready on frame buffer. When the source is in command queue, the 2D accelerator will fetch the bitblt command .

[1] Update date on 2001/02/08



For example:

```
CPUBlt(
        PDEV *pDst, LONG lDstX, LONG lDstY,
        PDEV *pSrc, LONG lSrcX, LONG lSrcY,
        ULONG ulRecWidth, ULONG ulRecHeight,
        ULONG ulROP)
{

        ULONG ulSrcDataPitch ;
        ULONG ulSrcDataSize ;
        LONG lDataSrcX, lDataSrcY ;
        PDSPkt01 *dsTempPkt01 ;

        if( lDstX < 0 ){
                lSrcX -= lDstX ;
                lDstX = 0 ;
        }

        if( lDstY < 0 ){
                lSrcY -= lDstY ;
                lDstY = 0 ;
        }

        if( lSrcX < 0 ){
                lDstX -= lSrcX ;
                ulRecWidth += lSrcX ;
                lSrcX = 0 ;
        }

        if( lSrcY < 0 ){
                lDstY -= lSrcY ;
                ulRecHeight += lSrcY ;
                lSrcY = 0 ;
        }

        ulSrcDataPitch = (ulRecWidth*pDst->ulPixelSizeInByte+15) & (ULONG)(~15) ;
        ulSrcDataSize = ulRecHeight * ulSrcDataPitch ;

        (PBYTE)dsTempPkt01 = (PBYTE)QueryCmdQueueLength( (ULONG)(ulSrcDataSize +
                                        PKT_TYPE01_LENGTH +
                                        PKT_SRCDATA_HEADER_LENGTH +
                                        15) & (ULONG)(~15)) ;
        // Fill dsTempPkt01 data ....

        // Fill Src in queue data

        (ULONG)dsTempPkt01 += (ULONG)PKT_TYPE01_LENGTH ;
        (PDSPktSrcDataHeader)dsTempPkt01->ulHeader0 = 0xD68A0000 ;
        (PDSPktSrcDataHeader)dsTempPkt01->ulHeader0 = (ulSrcDataSize/4) | 0x62100000 ;
        (ULONG)dsTempPkt01 += PKT_SRCDATA_HEADER_LENGTH ;
```

```
        for( y = 0 ; y < ulRecHeight ; y++ ){
            for( x = 0 ; x < ulRecWidth ; x++ ){

        (PBYTE)((ULONG)(dsTempPkt01)+y*ulSrcDataPitch+x*pDst->ulPixelSizeInByte)=
                    pSrc->ulBits[...][...] ;
            }
        }

        // Fill Null Data End ;

        (ULONG)dsTempPkt01 += ulSrcDataSize ;
        (PULONG)dsTempPkt01[0] = 0x168a0000 ; /* NULL DATA */
        (PULONG)dsTempPkt01[1] = 0x168a0000 ; /* NULL DATA */
        (ULONG)dsTempPkt01 += 8 ;

        vUpdateWirtePoint( dsTempPkt01 ) ;
    }
```

ii.) Constraint of bitblt function as using source in queue

SrcBase (reg 0x8200) must be zero.

SrcX must be larger than zero.

SrcX * BytePerPixel must be less than 16.

SrcPitch must be the multiple of 16 bytes (128 bits).

Src Y must be zero.

Source data must be the multiple of 16 bytes (128 bits) and just as the blted data size.

SrcPitch - (RecWidth+SrcX)*BytePerPixel must be less than 16. Each 128-bit couple of the source data should contain valid data.

On hardware constrain, the maximum source data size is 1Mega double words (4 Mega bytes). It is suggested to limited it as smaller than minimum command queue size (512KB).

Destination X and Y must be larger than zero.

Destination clip cannot work. It is always disabled. Software should take the clipping overhead of source in queue.

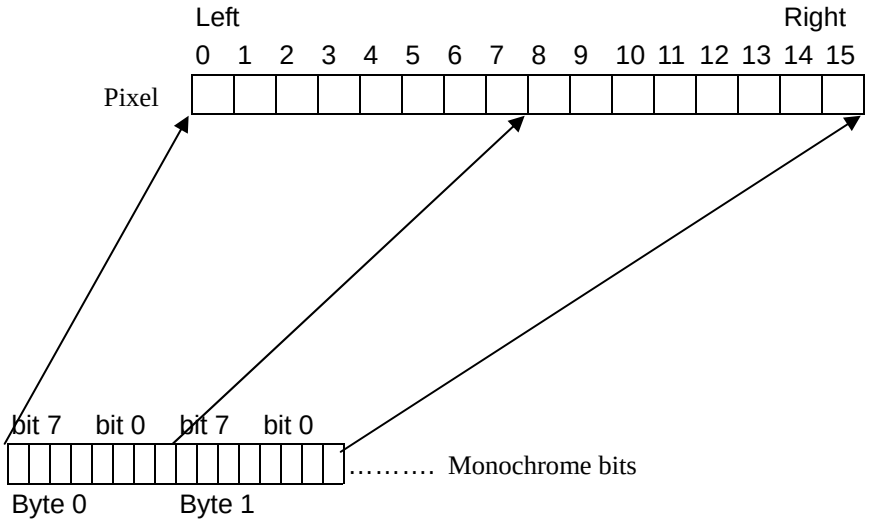
Chap 8. The Color Expansion Engine

Programming

The color expansion engine is to expand monochrome bitmap to color bitmap with pattern register. The source of color expansion engine is dedicated on pattern registers, thus, the maximum size of color expansion source cannot over then 384 bytes, or 384×8=3072 pixels. If the original source is larger than the size, you must split the source in the constrain of color expansion size issue.

8-1 The monochrome bitmap format

Before we describe how to program the color expansion engine, we must discuss the format of monochrome bitmap. Look the figure:



For those bytes on the same row, they keep the lower order on left. Each byte means 8 pixel, the higher bit on left. Thus, the byte 0 bit 7 is the up-left corner pixel of bitmap.

If the monochrome source in some OS is not the same as the hardware format, you must translate them to the display format.

8-2 The programming of color expansion

Because the source used the pattern registers of 2D engine, the color expansion engine do not

support those ROP with color pattern. If you must implement expansion with color pattern, you must use enhance color expansion described in next chapter. The table below describes those parameter an color expansion used:

Parameter	Register	Description
Destination Device	8210 8214 8216	To define which region you want to draw to.
Rectangle Dimension	8218 821A	The rectangle of the expanded bitmap
Source Coordinate	8208 820A	The source X, source Y with bit coordinate of the source bitmap.
Destination Coordinate	820C 820E	The destination X, destination Y of the up-left corner of the target corner.
Source Foreground Color	8224	The color which the bit 1 on source expand to.
Source Background Color	8228	The color which the bit 0 on source expand to.
Clip Left	8234	Clip rectangle left
Clip Top	8236	Clip rectangle top
Clip Right	8238	Clip rectangle right
Clip Bottom	823A	Clip rectangle bottom

The steps below describe how to program the color expansion.

Copy the source bitmap to pattern registers, 8300 ~ 847F.
Write 8210 to program the destination region base address. With frame buffer offset alignment on double word.
Write 8214 to program the pitch of destination region.
Write 8216 to program the height of destination region.
Write 820C to program the destination Y coordinate.
Write 820E to program the destination X coordinate.
Write 8208 to program the source Y coordinate.
Write 820A to program the source X coordinate.
Write 8224 to program the source foreground color.
Write 8228 to program the source background color.
Set the clip region
Prepare pattern if necessary.
Write 823C with command type Color Expansion (823C D[3:0] = 0001), with ROP and color depth to issue the command.

8-3 The Related Command Packet Type of Color Expansion

There is only one packet design for color expansion, type 13, output character. Color expansion usually be used on drawing text without font cache. In the case, color expansion is called several times to draw multiple text on the same destination. Therefore, the destination should be set once, we implement a packet type 10 to fill the destination base address, destination pitch and height, source foreground color and background color, and clip region.

Type 10: Dest Device Initialization (Length = 6)

31 30	29	28	27 24	23 20	19 16	15 0
1 0	0	1	0 1 1 0	1 0 0 0	1 0 1 0	0 / 01010
0 1 1 0	0 0 1 0		0 0 0 1	0		00110
Dst Base Addr (8210)						
Dst Height (8216)						Dst Pitch (8214)

Src Foreground Color (8224)	
Src Background Color (8228)	
Clip Top (8236)	Clip Left (8234)
Clip Bottom (823A)	Clip Right (8238)

After the initialization of destination device information, you can use burst command mode to write pattern register, and use packet type 13 to issue a color expansion command to draw the text.

Type 13: Each Character Out (With Color Expansion) (Length = 5)

31 30 29 28	27 24	23 20	19 16	15 0
1 0 0 1	0 1 1 0	1 0 0 0	1 0 1 0	0 01101
0 1 1 0	0 0 1 0	0 0 0 1	0	00101
Src Pitch (8204)				
Src X (820A)				Src Y (8208)
Dst X (820E)				Dst Y (820C)
Rect Height (821A)				Rect Width (8218)
Command (823C)				

In next chapter, we will discuss another expansion method, enhance color expansion. You can compare what different between them.

Chap 9. The Enhance Color Expansion

Engine Programming

Enhance color expansion engine is similar to the color expansion, they have the same source format, the similar programming method.

Enhance color expansion store the monochrome bitmap in frame buffer instead of the pattern registers on color expansion. Therefore, the pattern register is free on this engine, you can use enhance color expansion and pattern register to implement the monochrome source BitBlt with color pattern. By the way, the source is taken on frame buffer, the synchronization issue about frame buffer access by CPU and engine is important. If the source bitmap is fresh whenever the enhance color expansion used, you can use CPUBlt buffer to implement the source area. If the source bitmap is hardly to change such as often using font bitmap, you may put them on a area with simple synchronization method to keep the data consistence.

9-1 The parameter of enhance color expansion

The parameters are similar to the case under color expansion. The only difference is that the source base address is needed. Because the source bitmap is put on frame buffer, you need to give a base address to store the bitmap instead of the default address in pattern registers on color expansion case. The registers are listed below:

Parameter	Register	Description
Destination Device	8210 8214 8216	To define which region you want to draw to.
Source Device	8200 8204	The source is located on frame buffer, base address must give. The source pitch must be defined.
Rectangle Dimension	8218 821A	The rectangle of the expanded bitmap
Source Coordinate	8208 820A	The source X, source Y with bit coordinate of the source bitmap.
Destination Coordinate	820C 820E	The destination X, destination Y of the up-left corner of the target corner.
Source Foreground Color	8224	The color which the bit 1 on source expand to.
Source Background Color	8228	The color which the bit 0 on source expand to.
Clip Left	8234	Clip rectangle left
Clip Top	8236	Clip rectangle top
Clip Right	8238	Clip rectangle right
Clip Bottom	823A	Clip rectangle bottom

The steps below describe how to program the color expansion.

Copy the source bitmap to source bitmap area.
Write 8200 to give the source base address.
Write 8204 to give the source pitch
Write 8210 to program the destination region base address. With frame buffer offset alignment on double word.
Write 8214 to program the pitch of destination region.

Write 8216 to program the height of destination region.
Write 820C to program the destination Y coordinate.
Write 820E to program the destination X coordinate.
Write 8208 to program the source Y coordinate.
Write 820A to program the source X coordinate.
Write 8224 to program the source foreground color.
Write 8228 to program the source background color.
Set the clip region
Prepare pattern if necessary.
Write 823C with command type Color Expansion (823C D[3:0] = 0001), with ROP and color depth to issue the command.

9-2 The related packet types of enhance color expansion

There are several packet types are defined for enhance color expansion usage. There is a table listed those packet:

Type 7, monochrome source without pattern.
Type 8, monochrome source with color solid brush.
Type 10, Device initialize for textout
Type 11, Each Character Output (With Enhance Color Expansion)

For the monochrome source bitmap bitblt, you can use the type 7, some pattern initialization and burst packet mode to implement. The sample code for bitblt a monochrome bitmap without is listed below:

```
; Type 7, monochrome source without pattern.
PLABEL    BltMonoMem_FireEngine

sub    gs:[0].sd_dwCmdQueueLen, PKT_TYPE07_LENGTH
jl     BltMonoMem_WaitCmdQueueLength
BltMonoMem_WaitCmdQueueLengthDone:
mov  edi, gs:[0].sd_dwQueueWritePort

mov  eax, dwSrcBaseAddr
mov  edx, dwCPUBltBuffPitch
mov  dptr gs:[edi+QOFFSET].PkC07_dwHeader0, PKT_TYPE07_HEADER0
mov  dptr gs:[edi+QOFFSET].PkC07_dwHeader1, PKT_TYPE07_HEADER1
mov  dptr gs:[edi+QOFFSET].PkC07_dwSrcBaseAddr, eax
mov  dptr gs:[edi+QOFFSET].PkC07_wSrcPitch, edx
add   edi, 10h
and   edi, gs:[0].sd_dwQueueMask

mov  eax, dptr wSrcYOrg
and   eax, ( 7 shl 16 )
mov  ebx, dptr wDstYOrg
mov  ecx, dwDstBaseAddr
mov  edx, dwDstPitch

mov  dptr gs:[edi+QOFFSET-10h].PkC07_wSrcY, eax
mov  dptr gs:[edi+QOFFSET-10h].PkC07_wDstY, ebx
mov  dptr gs:[edi+QOFFSET-10h].PkC07_dwDstBaseAddr, ecx
mov  dptr gs:[edi+QOFFSET-10h].PkC07_wDstPitch, edx

add   edi, 10h
and   edi, gs:[0].sd_dwQueueMask

mov  eax, dwMaxSrcYExt
ror    eax, 16
```

```
mov   ax, wXExt

mov   ebx, dwSrcFGColor
mov   ecx, dwSrcBGColor
mov   edx, dwBltCommand

mov   dptr gs:[edi+QOFFSET-20h].PkC07_wRecWidth, eax
mov   dptr gs:[edi+QOFFSET-20h].PkC07_dwSrcFGColor, ebx
mov   dptr gs:[edi+QOFFSET-20h].PkC07_dwSrcBGColor, ecx
mov   dptr gs:[edi+QOFFSET-20h].PkC07_dwCommand, edx

add    edi, 10h
and    edi, gs:[0].sd_dwQueueMask
; mov gs:[0].sd_dwQueueWritePort, edi

mIncCPUBltBuffCount
mUpdateWritePort edi

mov   dx, wptr dwMaxSrcYExt
add    wDstYOrg, dx
sub    wYExt, dx
jg      BltMonoMem_GetBuffer

PLABEL    BltMonoMem_Exit
mov   ds, lpDstDev.sel
and    bptr ds:[0].deFlags, not BUSY
jmp    BS_Done

align 16
BltMonoMem_WaitCmdQueueLength:
mQueryCmdQueueLength     PKT_TYPE07_LENGTH
jmp    BltMonoMem_WaitCmdQueueLengthDone
```

You can reference the pattern description in the chapter Bitblt to find out how to program the enhance color expansion with pattern and ROP. The part is similar to the part in bitblt.

Textout is a big performance issue during benchmark execution. We try to minimize the cost when we give the textout parameter. For a long string, the destination device is unique and the source text will be modified but stored in fontcache, therefore, you can use type 10 to initialize the destination data, clipping rectangle, source color, then use continuous packet type 11 to drawout all of the text. Such as:

Fill packet type 10 (to prepare destination information)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Fill packet type 11 (to draw a text)
Update write point.

9-3 Enhance Color Expansion with the Source from Command Queue[2]

For monochrome source data, the bitblt software function should invoke enhance color expansion to expand the data into destination. For the implementation of source in queue, it also support the enhance color expansion.

Constraint of enhance color expansion as using

SrcBase (reg 0x8200) must be zero.

SrcX must be larger than zero.

SrcX must be less than 8.

SrcPitch is with the unit in byte without limitation.

Src Y must be zero.

Source data must be the multiple of 16 bytes (128 bits) and just as the blted data size.

SrcPitch must be equal to (RecWidth+SrcX+7)/8. Each byte of the source rectangle should contain valid data.

On hardware constrain, the maximum source data size is 1Mega double words (4 Mega bytes). It is suggested to limited it as smaller than minimum command queue size (512KB).

9-4 Destination X and Y must be larger than zero.

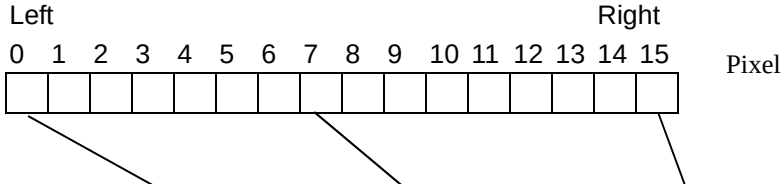
Destination clip can work. When destination X or destination Y is less than zero, you can shift the base address to let destination X/Y be positive and use destination clipping to complete the result. Software can use source clipping instead of the destination clipping.

The Color to Mono Enhance Color Expansion Engine Programming

Color to mono enhance color expansion engine is to transform color bitmap into monochrome bitmap according to a specific color key. When the color of some pixel in the source is equal to the color key, the value of the according pixel in the destination will be set as 1. If the color of the pixel in the source is different from the color key, the value of the according pixel in the destination will be set as 0. The source (color bitmap) should be on frame buffer (off-screen) and the destination (monochrome bitmap) should be not. But the source and destination of color to mono engine should be on frame buffer. You can use frame buffer off-screen region to implement the temporal destination, and then copy them to

[2] I

X(



the true destination. The figure describes the mapping from color bitmap to monochrome bitmap.

9-5 The parameter of enhance color expansion

The registers are listed below:

Parameter	Register	Description
Destination Device	8210 8214 8216	To define which region you want to draw to.
Source Device	8200 8204	The source is located on frame buffer, base address must give. The source pitch must be defined.
Rectangle Dimension	8218 821A	The rectangle of the expanded bitmap
Source Coordinate	8208 820A	The source X, source Y with bit coordinate of the source bitmap.
Destination Coordinate	820C 820E	The destination X, destination Y of the up-left corner of the target corner.
Color Key	821C	The criterion color.
Clip Left	8234	Clip rectangle left
Clip Top	8236	Clip rectangle top
Clip Right	8238	Clip rectangle right
Clip Bottom	823A	Clip rectangle bottom

The steps below describe how to program the color to mono.

Get a buffer from frame buffer region,
Clear the buffer by pattern Blt with color solid brush.
Write 8200 to give the source base address.
Write 8204 to give the source pitch
Write 8210 to program the destination region base address as the buffer start address. With frame buffer offset alignment on double word.
Write 8214 to program the pitch of destination region.
Write 8216 to program the height of destination region.
Write 820C to program the destination Y coordinate.
Write 820E to program the destination X coordinate.
Write 8208 to program the source Y coordinate.
Write 820A to program the source X coordinate.
Write 821C to program the color key
Set the clip region
Write 823C with command type Enhance Color Expansion (823C D[3:0] = 0010) and Color to Mono (823C D[23:21] = 001), with ROP and color depth to issue the command.
Wait engine idle then copy the mono bitmap form the buffer to the destination.

9-6 The Related Command Packet Type of Color to Mono

We use two packet types in color to mono. First is type 5 to clear the buffer in the frame buffer region and the second is type 2 for color to mono engine.

Type 5: Pattern Blt with Color Solid Brush (length=6)

Type 2: Source Copy With Color Solid Brush (Length = 9)

9-7 The Hardware Limitation of Color to Mono

Destination base address is 16 byte boundary

Destination pitch is 16 byte boundary
Destination X/Y should be zero
Source base address is 16 byte boundary
Source pitch is byte boundary

Chap 10. Line Drawing Engine

XGI Volari Family support an line drawing engine with negative coordinate supporting, multiple line segments supporting to 97 segment.

10-1 Register Format for Line Drawing

The register format for Line-Drawing is shown in the following table.

D[31:24]	D[23:16]	D[15:08]	D[07:00]	Register
AGP Base		Reserved		8204h
Y0		X0		8208h
Y1		X1		820Ch
Destination Base Address				8210h
Destination Height		Destination Pitch		8214h
Style Period		Line Count		8218h
FG (Foreground) Color				821Ch
BG (Background) Color				8220h
Line Style 0				822Ch
Line Style 1				8230h
Top Clipping		Left Clipping		8234h
Bottom Clipping		Right Clipping		8238h
Command				823Ch
Y2		X2		8300h
...		...		...
Y97		X97		847Ch

The Register Description:

Name	Description											
Source Base Address	Port	8200h~8203h										
	Register Type	Read/Write										
	D[31:0]	Base Linear Address of Source Bitmap in Byte										
		Limit: Source Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary in BitBlt, Byte Boundary in Enhanced Color Expansion.										
AGP Base	Port	8206h~8207h										
	Register Type	Read/Write										
	D[15:10]	Reserved										
	D[9:0]	AGP Base Address in Byte.										
		Limit: The accessibility of the Bit[9:0] are controlled by graphic window size.										
		Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Size
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	4M

		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	8M
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	0	16M
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	0	0	32M
		R/W	R/W	R/W	R/W	R/W	R/W	0	0	0	0	64M
		R/W	R/W	R/W	R/W	R/W	0	0	0	0	0	128M
		R/W	R/W	R/W	R/W	0	0	0	0	0	0	256M
X0	Port	8208h~8209h										
	Reg Type	Read/Write										
	D[15:0]	The X-start Coordinate of the First Line										
		Limit: For normal resolution mode it is in the S12.0 format,										
Y0	Port	820Ah~820Bh										
	Reg Type	Read/Write										
	D[15:0]	The Y-start Coordinate of the First Line										
		Limit: For normal resolution mode it is in the S12.0 format,										
X1	Port	820Ch~820Dh										
	Reg Type	Read/Write										
	D[15:0]	The X-end Coordinate of the First Line and X-start Coordinate of the Second Line										
		Limit: For normal resolution mode it is in the S12.0 format,										
Y1	Port	820Eh~820Fh										
	Reg Type	Read/Write										
	D[15:0]	The Y-end Coordinate of the First Line and Y-start Coordinate of the Second Line										
		Limit: For normal resolution mode it is in the S12.0 format,										
Destination Base Address	Port	8210h~8213h										
	Register Type	Read/Write										
	D[31:0]	Base Linear Address of Destination Bitmap in Byte										
		Limit: Destination Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary.										
Destination Pitch	Port	8214h~8215h										
	Register Type	Read/Write										
	D[15:14]	Reserved										
	D[13:0]	Row Pitch of Destination Bitmap in Byte										
		Limit: Double Word Boundary.										
Destination Height	Port	8216h~8217h										
	Register Type	Read/Write										

	D[15:13]	Reserved
	D[12:0]	Device Height of Destination Bitmap in Pixel
Line Count	Port	8218h~8219h
	Reg Type	Read/Write
	D[15:7]	Reserved
	D[6:0]	Total Line Count
		Limit: The range is from 1 to 97.
Style Period	Port	821Ah~821Bh
	Reg Type	Read/Write
	D[15:6]	Reserved
	D[5:0]	Period of the Line Style in Pixel
		Limit: A value of 0 equals 1 pixel.
Foreground Color	Port	821Ch~821Fh
	Register Type	Read/Write
	D[31:0]	Foreground Color Register of Pattern
Background Color	Port	8220h~8223h
	Register Type	Read/Write
	D[31:0]	Background Color Register of Pattern
Line Style	Port	822Ch~8233h
	Register Type	Read/Write
	D[63:0]	Line Style Pattern
Left Clipping	Port	8234h~8235h
	Register Type	Read/Write
	D[15:0]	Left Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Top Clipping	Port	8236h~8237h
	Register Type	Read/Write
	D[15:0]	Top Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Right Clipping	Port	8238h~8239h
	Register Type	Read/Write
	D[15:0]	Right Bound of Rectangular Clipping in Pixel

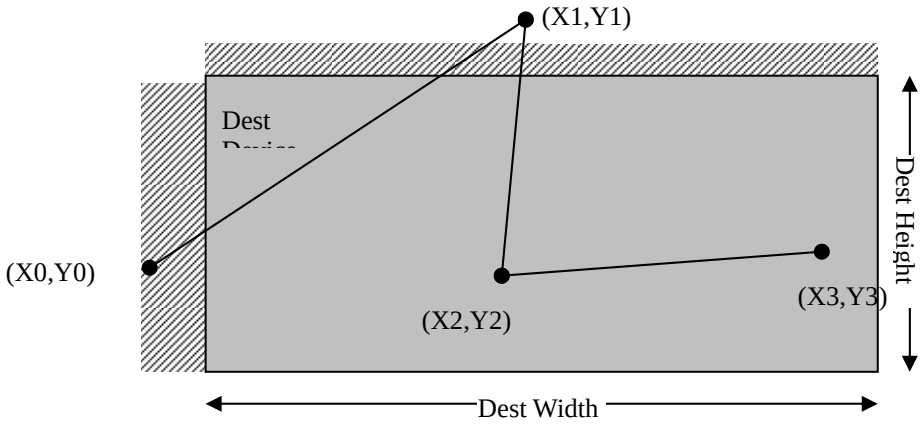
		Limit: In the 2's complement format.
Bottom Clipping	Port	823Ah~823Bh
	Register Type	Read/Write
	D[15:0]	Bottom Bound of Rectangular Clipping in Pixel
		Limit: In the 2's complement format.
Command Register	Reg Type	Read/Write
	Port	823Ch~823Fh
	D[31:27]	Reserved
	D[26]	Rectangular Clipping Merge Control 0: Merge clipping bound with screen bound 1: Do not merge clipping bound with screen bound
	D[25]	Destination Location Control 0: Destination is to video memory 1: Destination is to AGP memory no define in 310
	D[24]	Reserved
	D[23]	Line Style Control 0: Disable line style 1: Enable line style
	D[22]	Style counter reset control 0: Style counter will be reset 1: Style counter will not be reset
	D[21]	Line drawing last pixel control 0: Last pixel will be drawn 1: Last pixel will not be drawn
	D[20]	Transparent Control 0: Opaque 1: Transparent
	D[19]	Line Resolution Control 0: Disable high resolution line drawing 1: Enable high resolution line drawing

	D[18]	Rectangular Clipping Control 0: Disable rectangular clipping logic 1: Enable rectangular clipping logic
	D[17:16]	Enhanced graphics mode selection Bit[1:0] 00: 8 bpp 256 color mode 01: 16 bpp 64k color mode 10: 32 bpp True color mode 11: Reserved
	D[15:8]	256 Raster Operations
	D[7:4]	Reserved
	D[3:0]	Command type select Bit[3:0] 0000: BitBlt 0001: Color Expansion 0010: Enhanced Color Expansion 0011: Multiple Scanline 0100: Line Draw 0101: Trapezoid Fill 0110: Transparent BitBlt 0111: Alpha Blending 1000: Page Flipping 1001: Clean Z-buffer 1010: Gradient Fill 1011: 3D Stereo 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved
Points	Reg Type	Read/Write
	Port	8300h~847Fh
	D[31:16]	The Y coordinate of the N-th point
		Limit: For normal resolution mode it is in the S12.0 format,
	D[15:0]	The X coordinate of the N-th point

		Limit: For normal resolution mode it is in the S12.0 format,

10-2 Basic Line Drawing Support

See the figure below, we just fill the item to the associated register, then the line drawing is done.



The programming sequence are listed below:

Define the destination region. Program the destination base address (8210) , destination pitch (8214), and destination height (8216).

Define the foreground color (821C), background color of line(8220).

Give line segment count (8218).

Give the line style (822C and 8230). The line style is a monochrome pattern with foreground color in 821C and opaque (8220) or transparent background. The style runs with line pixel, contain 1 bits to 64 bits length. The style length is defined in style period (821A) which value 0 means period 1.

Give clipping rectangle (8234,8236,8238,823A).

Give Points (X0,Y0,X1,Y1,….X97,Y97)

Write command, set the clip enable/color depth/Disable high resolution line drawing, and decide if style count reset. Set D[3:0] 0100 for line drawing engine.

10-3 Register Format for Gradient Fill

The register format for Gradient Fill is shown in the following table.

D[31:24]	D[23:16]	D[15:08]	D[07:00]	I/O Address
Destination X		Destination Y		820Ch
Destination Base Address				8210h
Destination Height		Destination Pitch		8214h
Rectangular Height		Rectangular Width		8218h
Initial Color(RGB)				821Ch
Delta R				8220h
Delta G				8224h
Delta B				8228h
Top Clipping		Left Clipping		8234h
Bottom Clipping		Right Clipping		8238h
Command				823Ch

Rectangle Destination Y

Register Type: Read/Write

Read/Write Port: 820Ch~820Dh

D[15:13] Reserved

D[12:0] Started Y Coordinate of Destination Bitmap in Pixel

Limit: It is in the S12.0 format

Rectangle Destination X

Register Type: Read/Write

Read/Write Port: 820Eh~820Fh

D[15:13] Reserved

D[12:0] Started X Coordinate of Destination Bitmap in Pixel

Limit: It is in the S12.0 format

Destination Base Address

Register Type: Read/Write

Read/Write Port: 8210h~8213h

D[31:0] Base Linear Address of Destination Bitmap in Byte

Limit: Destination Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary.

Destination Pitch

Register Type: Read/Write

Read/Write Port: 8214h~8215h

D[15:14] Reserved

D[13:0] Row Pitch of Destination Bitmap in Byte

Limit: Double Word Boundary.

Destination Height

Register Type: Read/Write

Read/Write Port: 8216h~8217h

D[15:13] Reserved

D[12:0] Device Height of Destination Bitmap in Pixel

Rectangular Width

Register Type: Read/Write

Read/Write Port: 8218h~8219h

D[15:13] Reserved

D[12:0] Destination Rectangular Drawing Width of Bitmap in Pixel

Rectangular Height

Register Type: Read/Write

Read/Write Port: 821Ah~821Bh

D[15:13] Reserved

D[12:0] Destination Rectangular Drawing Height of Bitmap in Pixel

Initial Color

Register Type: Read/Write

Read/Write Port: 821Ch~821Fh

D[31:0] Initial Color for RGB when True Color

Delta R

Register Type: Read/Write

Read/Write Port: 8220h~8223h

D[31:24] Reserved

D[23:0] Delta R’s Color

Limit: It is in the S8.12 format

Delta G

Register Type: Read/Write

Read/Write Port: 8224h~8227h

D[31:24] Reserved

D[23:0] Delta G’s Color

Limit: It is in the S8.12 format

Delta B

Register Type: Read/Write

Read/Write Port: 8228h~822Bh

D[31:24] Reserved

D[23:0] Delta R’s Color

Limit: It is in the S8.12 format

Left Clipping

Register Type: Read/Write

Read/Write Port: 8234h~8235h

D[15:0] Left Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Top Clipping

Register Type: Read/Write

Read/Write Port: 8236h~8237h

D[15:0] Top Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Right Clipping

Register Type: Read/Write

Read/Write Port: 8238h~8239h

D[15:0] Right Bound of Rectangular Clipping in Pixel

Limit: In the 2's complement format.

Bottom Clipping

Register Type: Read/Write

Read/Write Port: 823Ah~823Bh

D[15:0] Bottom Bound of Rectangular Clipping in Pixel

Limit: In the 2's complement format.

Command Register

Register Type: Read/Write

Read/Write Port: 823Ch~823Fh

D31 Scan Line Trigger Function

0: Disable

1: Enable

D30 M-BitBlt Control

0: M-BitBlt function turn off

1: Fire engine need to wait for MframeRdyN is low

D29 ID control

0: No need to count the global ID

1: Send a signal (M2dFireEnd) to count global ID

D[28:27] Counter decrease indictor

00: All counter no decrease

01: Counter 1 will be decrease

10: Counter 2 will be decrease

11: Counter 3 will be decrease

D26 Rectangular Clipping Merge Control

0: Merge clipping bound with screen bound

1: Do not merge clipping bound with screen bound

D25 Destination Location Control

0: Destination is to video memory

1: Destination is to AGP memory no define in 310

D24 Scan Line Trigger Function for CRT1 or CRT2

0: CRT1

1: CRT2

D[23:21] Reserved

D20 RGB/BGR color mode control

0: RGB color mode

1: BGR color mode

D19 Gradient Fill Direction

0: Horizontal Direction

1: Vertical Direction

D18 Rectangular Clipping Control

0: Disable rectangular clipping logic

1: Enable rectangular clipping logic

D[17:16] Enhanced graphics mode selection Bit[1:0]

00: 8 bpp 256 color mode

01: 16 bpp 64k color mode

10: 32 bpp True color mode

11: Reserved

D[15:8] Reserved

D[7:6] Reserved

D[5:4] Reserved

D[3:0] Command type select Bit[3:0]

0000: BitBlt

0001: Color Expansion

0010: Enhanced Color Expansion

0011: Multiple Scanline

0100: Line Draw

0101: Trapezoid Fill

0110: Transparent BitBlt

0111: Alpha Blending

1000: 3D Function

1001: Clean Z-buffer

1010: Gradient Fill

1011: Stretch BitBlt

1100: Reserved

1101: Reserved

1110: Reserved

1111: Reserved

Gradient Fill shades the specified rectangles.

XGI Volari Family chip supports two shading modes:

GRADIENT_FILL_RECT_HORIZONTAL

Each rectangle is to be shaded from left to right. Specifically, the upper-left and lower-left pixels are the same color, as are the upper-right and lower-right pixels.

$R(V_1, V_2)$ is a rectangle

Color at point V_1 and V_2 are given by:

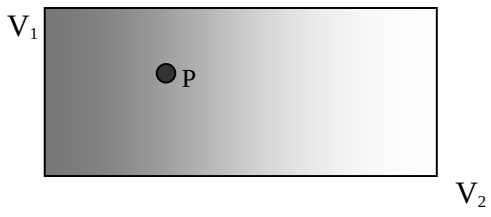
$C_{V1} = (R_{V1},G_{V1},B_{V1})$, $C_{V2} = (R_{V2},G_{V2},B_{V2})$

$P \in R(V_1, V_2)$, the color at point P is given by:

$R_P = R_{V1} + ((R_{V2} - R_{V1}) / (X_{V2} - X_{V1})) * (X_p - X_{V1})$

$G_P = G_{V1} + ((G_{V2} - G_{V1}) / (X_{V2} - X_{V1})) * (X_p - X_{V1})$

$B_P = B_{V1} + ((B_{V2} - B_{V1}) / (X_{V2} - X_{V1})) * (X_p - X_{V1})$



GRADIENT_FILL_RECT_VERTICAL

Each rectangle is to be shaded from top to bottom. Specifically, the upper-left and upper-right pixels are the same color, as are the lower-left and lower-right pixels.

$R(V_1, V_2)$ is a rectangle

Color at point V_1 and V_2 are given by:

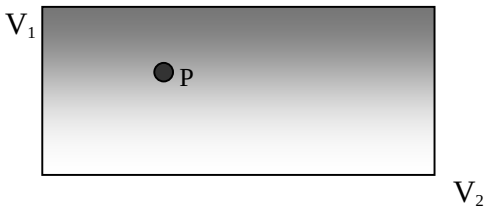
$C_{V1} = (R_{V1},G_{V1},B_{V1})$, $C_{V2} = (R_{V2},G_{V2},B_{V2})$

$P \in R(V_1, V_2)$, the color at point P is given by:

$R_P = R_{V1} + ((R_{V2} - R_{V1}) / (Y_{V2} - Y_{V1})) * (Y_p - Y_{V1})$

$G_P = G_{V1} + ((G_{V2} - G_{V1}) / (Y_{V2} - Y_{V1})) * (Y_p - Y_{V1})$

$B_P = B_{V1} + ((B_{V2} - B_{V1}) / (Y_{V2} - Y_{V1})) * (Y_p - Y_{V1})$



Example 1: shaded from left to right

PtlLr = lower-right point of destination rectangle

PtlUl = upper-left point of destination rectangle

DX = ptlLr.x - ptlUl.x

DY = ptlUl.y - ptlLr.y

RDiff = (ptlLr.R - ptlUl.R) << 12 // Transfer into S8.12 format

GDiff = (ptlLr.G - ptlUl.G) << 12 // Transfer into S8.12 format

BDiff = (ptlLr.B - ptlUl.B) << 12 // Transfer into S8.12 format

RDeltaX = RDiff / DX

GDeltaX = GDiff / DX

```
BDeltaX =  Bdiff / DX

// Set initial color in RGB888 format in spite of 16 or 32 bpp color mode
InitColor = (ptlUl.R << 16) | (ptlUl.G <<  8) | (ptlUl.B)

PkC09.dwHeader0     = PKT_TYPE09_HEADER0
PkC09.dwHeader1     = PKT_TYPE09_HEADER1
PkC09.dwDstBaseAddr = DstBaseAddress
PkC09.wDstPitch     = DstDelta
PkC09.wDstHeight    = DstHeight
PKC09.dwDeltaR      = RDeltaX           // S8.12 format
PkC09.dwDeltaG      = GDeltaX           // S8.12 format
PkC09.dwDeltaB      = BDeltaX           // S8.12 format
PkC09.wDstY         = ptlUl.y
PkC09.wDstX         = ptlLr.x
PkC09.wRecWidth     = DX
PkC09.wRecHeight    = DY
PkC09.dwInitColor   = InitColor
PkC09.dwCommand     = CMD_GRADIENTFILL |       // 823C D[3:0]
                      CMD_GF_HORIZONTAL_DIR |  // 823C D[19]
                      EngColorDepth            // 823C D[17:16]
PkC09.dwNullData[0] = PKT_NULL_CMD
PkC09.dwNullData[1] = PKT_NULL_CMD
PkC09.dwNullData[2] = PKT_NULL_CMD

// fire engine
Fire2DEngine(PktC09)
```

Register Format for Alpha Blending

The register format for Alpha Blending is shown in the following table.

D[31:24]	D[23:16]	D[15:08]	D[07:00]	I/O Address
Source Base Address				8200h
AGP Base		Source Pitch		8204h
Source X		Source Y		8208h
Destination X		Destination Y		820Ch
Destination Base Address				8210h
Destination Height		Destination Pitch		8214h
Rectangular Height		Rectangular Width		8218h
Alpha Value				821Ch
Top Clipping		Left Clipping		8234h
Bottom Clipping		Right Clipping		8238h
Command				823Ch

Source Base Address

Register Type: Read/Write

Read/Write Port: 8200h~8203h

D[31:0] Base Linear Address of Source Bitmap in Byte

Limit: Source Bitmap Addressing Range is from 0 to 256M,

Double Quad-Word Boundary in BitBlt,

Byte Boundary in Enhanced Color Expansion.

AGP Base

Register Type: Read/Write

Read/Write Port: 8206h~8207h

D[15:10] Reserved

D[9:0] AGP Base Address in Byte.

Limit: The accessibility of the Bit[9:0] are controlled by graphic window size.

Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Size
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	4M
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	8M
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	0	16M
R/W	R/W	R/W	R/W	R/W	R/W	R/W	0	0	0	32M
R/W	R/W	R/W	R/W	R/W	R/W	0	0	0	0	64M
R/W	R/W	R/W	R/W	R/W	0	0	0	0	0	128M
R/W	R/W	R/WR/W	0	0	0	0	0	0		256M

Source Pitch

Register Type: Read/Write

Read/Write Port: 8204h~8205h

D[15:14] Reserved

D[13:0] Row Pitch of Source Bitmap in Byte.

Limit: Double Word Boundary in BitBlt,

Byte Boundary in Color Expansion and Enhanced Color Expansion.

Rectangle Source Y

Register Type: Read/Write

Read/Write Port: 8208h~8209h

D[15:13] Reserved

D[12:0] Started Y Coordinate of Source Bitmap in Pixel

Limit: It is in the S12.0 format

Rectangle Destination Y

Register Type: Read/Write

Read/Write Port: 820Ch~820Dh

D[15:13] Reserved

D[12:0] Started Y Coordinate of Destination Bitmap in Pixel

Limit: It is in the S12.0 format

Rectangle Destination X

Register Type: Read/Write

Read/Write Port: 820Eh~820Fh

D[15:13] Reserved

D[12:0] Started X Coordinate of Destination Bitmap in Pixel

Limit: It is in the S12.0 format

Destination Base Address

Register Type: Read/Write

Read/Write Port: 8210h~8213h

D[31:0] Base Linear Address of Destination Bitmap in Byte

Limit: Destination Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary.

Destination Pitch

Register Type: Read/Write

Read/Write Port: 8214h~8215h

D[15:14] Reserved

D[13:0] Row Pitch of Destination Bitmap in Byte

Limit: Double Word Boundary.

Destination Height

Register Type: Read/Write

Read/Write Port: 8216h~8217h

D[15:13] Reserved

D[12:0] Device Height of Destination Bitmap in Pixel

Rectangular Width

Register Type: Read/Write

Read/Write Port: 8218h~8219h

D[15:13] Reserved

D[12:0] Destination Rectangular Drawing Width of Bitmap in Pixel

Rectangular Height

Register Type: Read/Write

Read/Write Port: 821Ah~821Bh

D[15:13] Reserved

D[12:0] Destination Rectangular Drawing Height of Bitmap in Pixel

Alpha Value

Register Type: Read/Write

Read/Write Port: 821Ch~821Fh

D[31:8] Reserved

D[7:0] Alpha Value

Left Clipping

Register Type: Read/Write

Read/Write Port: 8234h~8235h

D[15:0] Left Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Top Clipping

Register Type: Read/Write

Read/Write Port: 8236h~8237h

D[15:0] Top Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Right Clipping

Register Type: Read/Write

Read/Write Port: 8238h~8239h

D[15:0] Right Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Bottom Clipping

Register Type: Read/Write

Read/Write Port: 823Ah~823Bh

D[15:0] Bottom Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Command Register

Register Type: Read/Write

Read/Write Port: 823Ch~823Fh

D31	Scan Line Trigger Function
	0: Disable
	1: Enable
D30	M-BitBlt Control
	0: M-BitBlt function turn off
	1: Fire engine need to wait for MframeRdyN is low
D29	ID control
	0: No need to count the global ID
	1: Send a signal (M2dFireEnd) to count global ID
D[28:27]	Counter decrease indictor
	00: All counter no decrease
	01: Counter 1 will be decrease
	10: Counter 2 will be decrease
	11: Counter 3 will be decrease
D26	Rectangular Clipping Merge Control
	0: Merge clipping bound with screen bound
	1: Do not merge clipping bound with screen bound
D25	Destination Location Control
	0: Destination is to video memory
	1: Destination is to AGP memory no define in 310
D24	Scan Line Trigger Function for CRT1 or CRT2
	0: CRT1
	1: CRT2
D[23:22]	Reserved
D21	When Hi Color 3D Full Scene Antialiasing function, RGB/BGR Control
	0: BGR

	1: RGB
D[20:19]	When the command is Alpha Blending
	00 : Constant alpha value Blending
	01 : per-pixel alphavalue Blending
	10 : no dest. Alpha Blending
	11 : 3D Full Scene Antialiasing
D18	Rectangular Clipping Control
	0: Disable rectangular clipping logic
	1: Enable rectangular clipping logic
D[17:16]	Enhanced graphics mode selection Bit[1:0]
	00: 8 bpp 256 color mode
	01: 16 bpp 64k color mode
	10: 32 bpp True color mode
	11: Reserved
D[15:8]	Reserved
	D[7:6] Reserved
D[5:4]	Source select Bit[1:0]
	00: Source is from video memory
	01: Source is from CPU-driven BitBlt buffer
	10: Source is from AGP memory
	11: Reserved
D[3:0]	Command type select Bit[3:0]
	0000: BitBlt
	0001: Color Expansion
	0010: Enhanced Color Expansion
	0011: Multiple Scanline
	0100: Line Draw
	0101: Trapezoid Fill
	0110: Transparent BitBlt
	0111: Alpha Blending
	1000: 3D Function
	1001: Clean Z-buffer
	1010: Gradient Fill
	1011: Stretch BitBlt
	1100: Reserved
	1101: Reserved

1110: Reserved

1111: Reserved

Alpha Blend provides bit-block transfer capabilities with alpha blending.

The three possible cases for the blend function are:

The source bitmap has no per pixel alpha, so the blend is applied to the pixel's color channels based on the constant source alpha value specified in α as follows:

Dst.R = Round(((Src.R * α) + ((255 - α) * Dst.R)) / 255)

Dst.G = Round(((Src.G * α) + ((255 - α) * Dst.G)) / 255)

Dst.B = Round(((Src.B * α) + ((255 - α) * Dst.B)) / 255)

Dst.A = Round(((Src.A * α) + ((255 - α) * Dst.A)) / 255)

The source bitmap has per pixel alpha values. The blend is computed as follows:

Dst.R = Src.R + Round(((255 - Src.A) * Dst.R) / 255)

Dst.G = Src.G + Round(((255 - Src.A) * Dst.G) / 255)

Dst.B = Src.B + Round(((255 - Src.A) * Dst.B) / 255)

Dst.A = Src.A + Round(((255 - Src.A) * Dst.A) / 255)

The source bitmap has both per pixel alpha values, and the constant source alpha value specified in α. The blend is computed as follows:

Temp.R = Round((Src.R * α) / 255)

Temp.G = Round((Src.G * α) / 255)

Temp.B = Round((Src.B * α) / 255)

Temp.A = Round((Src.A * α) / 255)

Note that the following equations use the just-computed Temp.A value:

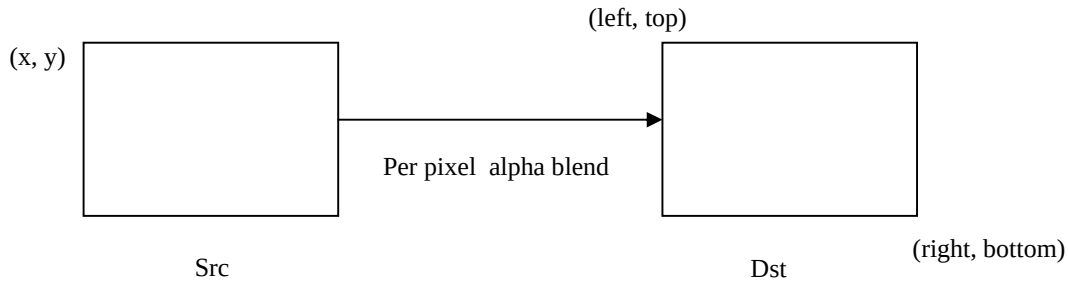
Dst.R = Temp.R + Round(((255 - Temp.A) * Dst.R) / 255)

Dst.G = Temp.G + Round(((255 - Temp.A) * Dst.G) / 255)

Dst.B = Temp.B + Round(((255 - Temp.A) * Dst.B) / 255)

Dst.A = Temp.A + Round(((255 - Temp.A) * Dst.A) / 255)

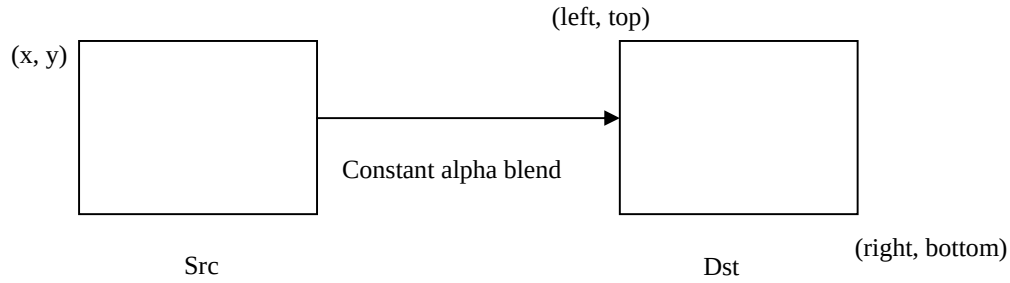
Example 1: do per-pixel alpha blending




```
PkC02_dwHeader0     = PKT_TYPE02_HEADER0
PkC02_dwHeader1     = PKT_TYPE02_HEADER1
PkC02.dwSrcBaseAddr = SrcBaseAddress
PkC02.wSrcPitch     = SrcDelta
PkC02.wSrcX         = ptlSrc.x
PkC02.wSrcY         = ptlSrc.y
PkC02.wDstX         = rclDst.left
PkC02.wDstY         = rclDst.top
PkC02.dwDstBaseAddr = DstBaseAddress
PkC02.wDstPitch     = DstDelta
PkC02.wDstHeight    = DstHeight
PkC02.wRecWidth     = rclDst.right  - rclDst.left
PkC02.wRecHeight    = rclDst.bottom – prclDst.top
PkC02.dwCommand     = (ROP3_S << 8) |               // 823C D[15:8]
                      CMD_ALPHABLENDING |           // 823C D[3:0]
                      CMD_PER_PIXEL_ALPHA_BLEND |   // 823C D[20:19]
                      EngColorDepth                 // 823C D[17:16]
PkC02.dwNullData[0] = PKT_NULL_CMD

// fire engine
```

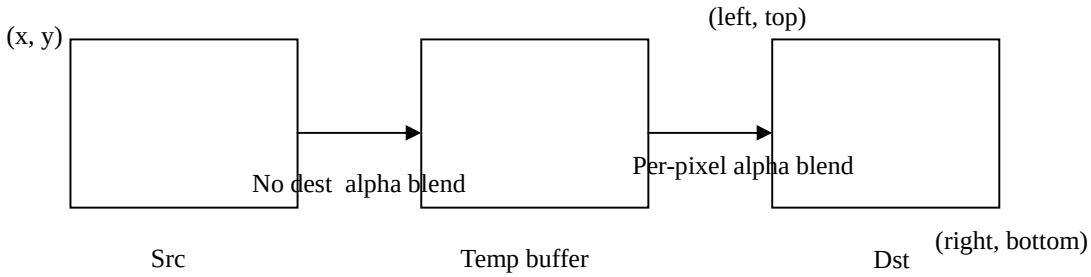
Example 2: do constant alpha blending




```
PkC02_dwHeader0     = PKT_TYPE02_HEADER0
PkC02_dwHeader1     = PKT_TYPE02_HEADER1
PkC02.dwSrcBaseAddr = SrcBaseAddress
PkC02.wSrcPitch     = SrcDelta
PkC02.wSrcX         = ptlSrc.x
PkC02.wSrcY         = ptlSrc.y
PkC02.wDstX         = rclDst.left
PkC02.wDstY         = rclDst.top
PkC02.dwDstBaseAddr = DstBaseAddress
PkC02.wDstPitch     = DstDelta
PkC02.wDstHeight    = DstHeight
PkC02.wRecWidth     = rclDst.right  - rclDst.left
PkC02.wRecHeight    = rclDst.bottom – prclDst.top
PkC02_dwAlphaValue  = ConstantAlpha
PkC02.dwCommand     = (ROP3_S << 8) |              // 823C D[15:8]
                      CMD_ALPHABLENDING |          // 823C D[3:0]
                      CMD_CONSTANT_ALPHA_BLEND |   // 823C D[20:19]
                      EngColorDepth                // 823C D[17:16]
PkC02.dwNullData[0] = PKT_NULL_CMD

// fire engine
Fire2DEngine(PktC02)
```

Example 3: do constant & per-pixel alpha blending



Width = rclDst.right - prclDst.left
Height = rclDst.bottom – prclDst.top
TempBuffer = CreateBuffer(Width, Height)

```
// do constant alpha blending for source surface
PkC02_1.dwHeader0     = PKT_TYPE02_HEADER0
PkC02_1.dwHeader1     = PKT_TYPE02_HEADER1
PkC02_1.dwSrcBaseAddr = SrcBaseAddress
PkC02_1.wSrcPitch     = SrcDelta
PkC02_1.wSrcY         = ptlSrc.y
PkC02_1.wSrcX         = ptlSrc.x
PkC02_1.wDstY         = 0
PkC02_1.wDstX         = 0
PkC02_1.dwDstBaseAddr = TempBuffer.BaseAddress
PkC02_1.wDstPitch     = TempBuffer.Delta
PkC02_1.wDstHeight    = Height
PkC02_1.wRecWidth     = Width
PkC02_1.wRecHeight    = Height
PkC02_1.dwAlphaValue  = α
PkC02_1.dwCommand     = ROP3_S << 8)     |          // 823C D[15:8]
                  CMD_ALPHABLENDING |          // 823C D[3:0]
                  CMD_NO_DEST_ALPHA_BLEND |    // 823C D[20:19]
```

```
                EngColorDepth              // 823C D[17:16]
PkC02_1.dwNullData[0] = PKT_NULL_CMD

// do per pixel alpha blending for temp buffer & destination surface
PkC02_2.dwHeader0    = PKT_TYPE02_HEADER0
PkC02_2.dwHeader1    = PKT_TYPE02_HEADER1
PkC02_2.dwSrcBaseAddr = TempBuffer.BaseAddress
PkC02_2.wSrcPitch    = TempBuffer.Delta
PkC02_2.wSrcY        = 0
PkC02_2.wSrcX        = 0
PkC02_2.wDstY        = rclDst.top
PkC02_2.wDstX        = rclDst.left
PkC02_2.dwDstBaseAddr = DstBaseAddress
PkC02_2.wDstPitch    = DstDelta
PkC02_2.wDstHeight   = DstHeight
PkC02_2.wRecWidth    = Width
PkC02_2.wRecHeight   = Height
PkC02_2.dwCommand    = ROP3_S << 8)    |          // 823C D[15:8]
                CMD_ALPHABLENDING |         // 823C D[3:0]
                CMD_PER_PIXEL_ALPHA_BLEND |   // 823C D[20:19]
                EngColorDepth              // 823C D[17:16]
PkC02_2.dwNullData[0] = PKT_NULL_CMD

// fire engine
Fire2DEngine(PktC02_1)
Fire2DEngine(PktC02_2)

// free temp buffer
FreeBuffer(TempBuffer)
```

10-4 Register Format for Color Expansion and Enhanced Color Expansion Functions

The following table shows the register format for the color expansion and enhanced color expansion functions.

D[31:24]	D[23:16]	D[15:08]	D[07:00]	I/O Address
Source Base Address				8200h
Reserved	AGP Base	Source Pitch		8204h
Source X		Source Y		8208h
Destination X		Destination Y		820Ch
Destination Base Address				8210h
Destination Height		Destination Pitch		8214h
Rectangular Height		Rectangular Width		8218h
Pattern FG (Foreground) Color				821Ch
Pattern BG (Background) Color				8220h
Source FG (Foreground) Color				8224h
Source BG (Background) Color				8228h
Mask3	Mask2	Mask1	Mask0	822Ch
Mask7	Mask6	Mask5	Mask4	8230h
Top Clipping		Left Clipping		8234h
Bottom Clipping		Right Clipping		8238h
Command				823Ch
Pattern 3	Pattern 2	Pattern 1	Pattern 0	8300h
Pattern 7	Pattern 6	Pattern 5	Pattern 4	8304h
Pattern 11	Pattern 10	Pattern 9	Pattern 8	8308h
Pattern 15	Pattern 14	Pattern 13	Pattern 12	830Ch
Pattern 19	Pattern 18	Pattern 17	Pattern 16	8310h
Pattern 23	Pattern 22	Pattern 21	Pattern 20	8314h
Pattern 27	Pattern 26	Pattern 25	Pattern 24	8318h
Pattern 31	Pattern 30	Pattern 29	Pattern 28	831Ch
Pattern 35	Pattern 34	Pattern 33	Pattern 32	8320h
Pattern 39	Pattern 38	Pattern 37	Pattern 36	8324h
Pattern 43	Pattern 42	Pattern 41	Pattern 40	8328h
Pattern 47	Pattern 46	Pattern 45	Pattern 44	832Ch

Pattern 51	Pattern 50	Pattern 49	Pattern 48	8330h
Pattern 55	Pattern 54	Pattern 53	Pattern 52	8334h
Pattern 59	Pattern 58	Pattern 57	Pattern 56	8338h
Pattern 63	Pattern 62	Pattern 61	Pattern 60	833Ch
Pattern 67	Pattern 66	Pattern 65	Pattern 64	8340h
Pattern 71	Pattern 70	Pattern 69	Pattern 68	8344h
Pattern 75	Pattern 74	Pattern 73	Pattern 72	8348h
Pattern 79	Pattern 78	Pattern 77	Pattern 76	834Ch
Pattern 83	Pattern 82	Pattern 81	Pattern 80	8350h
Pattern 87	Pattern 86	Pattern 85	Pattern 84	8354h
Pattern 91	Pattern 90	Pattern 89	Pattern 88	8358h
Pattern 95	Pattern 94	Pattern 93	Pattern 92	835Ch
Pattern 99	Pattern 98	Pattern 97	Pattern 96	8360h
Pattern 103	Pattern 102	Pattern 101	Pattern 100	8364h
Pattern 107	Pattern 106	Pattern 105	Pattern 104	8368h
Pattern 111	Pattern 110	Pattern 109	Pattern 108	836Ch
Pattern 115	Pattern 114	Pattern 113	Pattern 112	8370h
Pattern 119	Pattern 118	Pattern 117	Pattern 116	8374h
Pattern 123	Pattern 122	Pattern 121	Pattern 120	8378h
Pattern 127	Pattern 126	Pattern 125	Pattern 124	837Ch
Pattern 131	Pattern 130	Pattern 129	Pattern 128	8380h
Pattern 135	Pattern 134	Pattern 133	Pattern 132	8384h
Pattern 139	Pattern 138	Pattern 137	Pattern 136	8388h
Pattern 143	Pattern 142	Pattern 141	Pattern 140	838Ch
Pattern 147	Pattern 146	Pattern 145	Pattern 144	8390h
Pattern 151	Pattern 150	Pattern 149	Pattern 148	8394h
Pattern 155	Pattern 154	Pattern 153	Pattern 152	8398h
Pattern 159	Pattern 158	Pattern 157	Pattern 156	839Ch
Pattern 163	Pattern 162	Pattern 161	Pattern 160	83A0h
Pattern 167	Pattern 166	Pattern 165	Pattern 164	83A4h
Pattern 171	Pattern 170	Pattern 169	Pattern 168	83A8h
Pattern 175	Pattern 174	Pattern 173	Pattern 172	83ACh
Pattern 179	Pattern 178	Pattern 177	Pattern 176	83B0h
Pattern 183	Pattern 182	Pattern 181	Pattern 180	83B4h
Pattern 187	Pattern 186	Pattern 185	Pattern 184	83B8h

Pattern 191	Pattern 190	Pattern 189	Pattern 188	83BCh
Pattern 195	Pattern 194	Pattern 193	Pattern 192	83C0h
Pattern 199	Pattern 198	Pattern 197	Pattern 196	83C4h
Pattern 203	Pattern 202	Pattern 201	Pattern 200	83C8h
Pattern 207	Pattern 206	Pattern 205	Pattern 204	83CCh
Pattern 211	Pattern 210	Pattern 209	Pattern 208	83D0h
Pattern 215	Pattern 214	Pattern 213	Pattern 212	83D4h
Pattern 219	Pattern 218	Pattern 217	Pattern 216	83D8h
Pattern 223	Pattern 222	Pattern 221	Pattern 220	83DCh
Pattern 227	Pattern 226	Pattern 225	Pattern 224	83E0h
Pattern 231	Pattern 230	Pattern 229	Pattern 228	83E4h
Pattern 235	Pattern 234	Pattern 233	Pattern 232	83E8h
Pattern 239	Pattern 238	Pattern 237	Pattern 236	83ECh
Pattern 243	Pattern 242	Pattern 241	Pattern 240	83F0h
Pattern 247	Pattern 246	Pattern 245	Pattern 244	83F4h
Pattern 251	Pattern 250	Pattern 249	Pattern 248	83F8h
Pattern 255	Pattern 254	Pattern 253	Pattern 252	83FCh

Source Base Address

Register Type: Read/Write

Read/Write Port: 8200h~8203h

D[31:0] Base Linear Address of Source Bitmap in Byte

Limit: Source Bitmap Addressing Range is from 0 to 256M,

Double Quad-Word Boundary in BitBlt,

Byte Boundary in Enhanced Color Expansion.

AGP Base

Register Type: Read/Write

Read/Write Port: 8206h~8207h

D[15:10] Reserved

D[9:0] AGP Base Address in Byte.

Limit: The accessibility of the Bit[9:0] are controlled by graphic window size.

Bit9 Bit8 Bit7 Bit6 Bit5 Bit4 Bit3 Bit2 Bit1 Bit0 Size

R/W R/W R/W R/W R/W R/W R/W R/W R/W R/W 4M

R/W R/W R/W R/W R/W R/W R/W R/W R/W 0 8M

R/W R/W R/W R/W R/W R/W R/W R/W 0 0 16M

R/W R/W R/W R/W R/W R/W R/W 0 0 0 32M

R/W R/W R/W R/W R/W R/W 0 0 0 0 64M

R/W R/W R/W R/W R/W 0 0 0 0 0 128M

R/W R/W R/WR/W 0 0 0 0 0 0 256M

Source Pitch

Register Type: Read/Write

Read/Write Port: 8204h~8205h

D[15:14] Reserved

D[13:0] Row Pitch of Source Bitmap in Byte.

Limit: Double Word Boundary in BitBlt,

Byte Boundary in Color Expansion and Enhanced Color Expansion.

Rectangle Source Y

Register Type: Read/Write

Read/Write Port: 8208h~8209h

D[15:13] Reserved

D[12:0] Started Y Coordinate of Source Bitmap in Pixel

Limit: It is in the S12.0 format

Rectangle Source X

Register Type: Read/Write

Read/Write Port: 820Ah~820Bh

D[15:13] Reserved

D[12:0] Started X Coordinate of Source Bitmap in Pixel

Limit: It is in the S12.0 format

Rectangle Destination Y

Register Type: Read/Write

Read/Write Port: 820Ch~820Dh

D[15:13] Reserved

D[12:0] Started Y Coordinate of Destination Bitmap in Pixel

Limit: It is in the S12.0 format

Rectangle Destination X

Register Type: Read/Write

Read/Write Port: 820Eh~820Fh

D[15:13] Reserved

D[12:0] Started X Coordinate of Destination Bitmap in Pixel

Limit: It is in the S12.0 format

Destination Base Address

Register Type: Read/Write

Read/Write Port: 8210h~8213h

D[31:0] Base Linear Address of Destination Bitmap in Byte

Limit: Destination Bitmap Addressing Range is from 0 to 256M, Double Quad-Word Boundary.

Destination Pitch

Register Type: Read/Write

Read/Write Port: 8214h~8215h

D[15:14] Reserved

D[13:0] Row Pitch of Destination Bitmap in Byte

Limit: Double Word Boundary.

Destination Height

Register Type: Read/Write

Read/Write Port: 8216h~8217h

D[15:13] Reserved

D[12:0] Device Height of Destination Bitmap in Pixel

Rectangular Width

Register Type: Read/Write

Read/Write Port: 8218h~8219h

D[15:13] Reserved

D[12:0] Destination Rectangular Drawing Width of Bitmap in Pixel

Rectangular Height

Register Type: Read/Write

Read/Write Port: 821Ah~821Bh

D[15:13] Reserved

D[12:0] Destination Rectangular Drawing Height of Bitmap in Pixel

Pattern Foreground Color

Register Type: Read/Write

Read/Write Port: 821Ch~821Fh

D[31:0] Foreground Color Register of Pattern

Pattern Background Color

Register Type: Read/Write

Read/Write Port: 8220h~8223h

D[31:0] Background Color Register of Pattern

Source Foreground Color

Register Type: Read/Write

Read/Write Port: 8224h~8227h

D[31:0] Foreground Color Register of Source

Source Background Color

Register Type: Read/Write

Read/Write Port: 8228h~822Bh

D[31:0] Background Color Register of Source

Mono Mask Register

Register Type: Read/Write

Read/Write Port: 822Ch~8233h

D[63:0] Monochrome Mask Register of Pattern

Left Clipping

Register Type: Read/Write

Read/Write Port: 8234h~8235h

D[15:0] Left Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Top Clipping

Register Type: Read/Write

Read/Write Port: 8236h~8237h

D[15:0] Top Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Right Clipping

Register Type: Read/Write

Read/Write Port: 8238h~8239h

D[15:0] Right Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Bottom Clipping

Register Type: Read/Write

Read/Write Port: 823Ah~823Bh

D[15:0] Bottom Bound of Rectangular Clipping in Pixel

Limit: In the 2’s complement format.

Command Register

Register Type: Read/Write

Read/Write Port: 823Ch~823Fh

D31 Scan Line Trigger Function

0: Disable

1: Enable

D30 M-BitBlt Control

0: M-BitBlt function turn off

1: Fire engine need to wait for MframeRdyN is low

D29 ID control

0: No need to count the global ID

1: Send a signal (M2dFireEnd) to count global ID

D[28:27] Counter decrease indictor

00: All counter no decrease

01: Counter 1 will be decrease

10: Counter 2 will be decrease

11: Counter 3 will be decrease

D26 Rectangular Clipping Merge Control

0: Merge clipping bound with screen bound

1: Do not merge clipping bound with screen bound

D25 Destination Location Control

0: Destination is to video memory

1: Destination is to AGP memory no define in 310

D24 Scan Line Trigger Function for CRT1 or CRT2

0: CRT1

1: CRT2

D[23:21] The sub-function bits of the CE/HCE function

000 : reserved

001 : Color to Mono

010 : Antialiasing text

others: reserved

D20 Transparent Control

0: Opaque

1: Transparent

D19 Reserved

D18 Rectangular Clipping Control

0: Disable rectangular clipping logic

1: Enable rectangular clipping logic

D[17:16] Enhanced graphics mode selection Bit[1:0]

00: 8 bpp 256 color mode

01: 16 bpp 64k color mode

10: 32 bpp True color mode

11: Reserved

D[15:8] 256 Raster Operations

D[7:6] Pattern select Bit[1:0]

00: Pattern is from pattern foreground color register

01: Pattern is from pattern register

10: Pattern is from monochrome mask register

11: Reserved

D[5:4] Source select Bit[1:0]

00: Source is from video memory

01: Source is from CPU-driven BitBlt buffer

10: Source is from AGP memory

11: Reserved

D[3:0] Command type select Bit[3:0]

0000: BitBlt

0001: Color Expansion

0010: Enhanced Color Expansion

0011: Multiple Scanline

0100: Line Draw

0101: Trapezoid Fill

0110: Transparent BitBlt

0111: Alpha Blending

1000: 3D Function

1001: Clean Z-buffer

1010: Gradient Fill

1011: Stretch BitBlt

1100: Reserved

1101: Reserved

1110: Reserved

1111: Reserved

Pattern Register n

	Limit: Double Word Boundary
Register Type:	Read/Write
Read/Write Port:	8300h~833Fh for 256-color
	8300h~837Fh for high-color
	8300h~83FFh for true-color
D[7: 0]	For BitBlt and Enhanced Color Expansion function, these registers store the 8x8 color pattern bitmap.
	For Color Expansion function, the registers from 8300h to 847Fh store the monochrome bitmap, thus it can expand 3072 pixels at one command.

XGI Volari Family chip supports antialiased text

Data format:

Glyphs are provide as packed 4 bits per-pixel bitmap, specify the corresponding per-pixel alpha values.

Pixel in byte format:

high nibble : low nibble

| |

low pixel high pixel

Example of glyph bitmap

Width = 10

Height = 26

0 0 0 6 ad ff fc a5 0 0

0 0 7 ef ff ff ff ff f9 10

0 0 cf ff ff ff ff ff ff f0

0 c ff fd 51 0 3 8e ff f0

0 af ff 60 0 0 0 0 6e f0

4 ff f5 0 0 0 0 0 1 c0

c ff 90 0 0 0 0 0 0 0

2f ff 10 0 0 0 0 0 0 0

7f fa 0 0 0 0 0 0 0 0

bf f5 0 0 0 0 0 0 0 0

cf f3 0 0 0 0 0 0 0 0

ff f0 0 0 0 0 0 0 0 0

ff f0 0 0 0 0 0 0 0 0

ff f0 0 0 0 0 0 0 0 0

ff f0 0 0 0 0 0 0 0 0

df f2 0 0 0 0 0 0 0 0

bf f5 0 0 0 0 0 0 0 0

7f f9 0 0 0 0 0 0 0 0

2f ff 10 0 0 0 0 0 0 0

d ff a0 0 0 0 0 0 0 0

5 ff f5 0 0 0 0 0 0 c0

0 cf ff 60 0 0 0 0 6e f0

0 1e ff fd 51 0 3 8d ff f0

0 2 df ff ff ff ff ff f0

0 0 8 ff ff ff ff e9 10

0 0 0 16 be ff fc 95 0 0

Blending formula:

k = per-pixel's alpha value

C_0 = background color

C_1 = foreground color

b = blending value = (k+1)/16 // {k = 1,2,..,15}

0 // {k = 0}

d = (1 - b) * C_0 + b * C_1 // destination color

Example 1: Enhance color expansion for antialiased text

```
// initialize for destination
PkC10.dwHeader0     = PKT_TYPE10_HEADER0
PkC10.dwHeader1     = PKT_TYPE10_HEADER1
PkC10.dwDstBaseAddr = DstBaseAddress
PkC10.wDstPitch     = DstDelta
PkC10.wDstHeight    = DstHeight
PkC10.dwSrcFGColor  = FGColor
PkC10.wLeftClip     = rclClip.left
PkC10.wTopClip      = rclClip.top
PkC10.wRightClip    = rclClip.right
PkC10.wBottomClip   = rclClip.bottom

// fire engine
Fire2Dengine(PktC10)

// use PkC11 to draw each glyph
DstX   = Started X coordinate of destination bitmap in pixel
DstY   = Started Y coordinate of destination bitmap in pixel
```

```
Width  = Width of the glyph
Height = Height of the glyph
GlyphBaseAddress = Base address of each glyph in frame buffer

// 1 pixel = 4 bits, calculate the glyph bitmap size in byte
WidthByte = (Width + 1) / 2

// enhance color expansion
PkC11.dwHeader0     = PKT_TYPE11_HEADER0
PkC11.dwHeader1     = PKT_TYPE11_HEADER1
PkC11.wSrcY         = 0
PkC11.wSrcX         = 0
PkC11.dwSrcBaseAddr = GlyphBaseAddress
PkC11.wSrcPitch     = WidthByte
PkC11.wDstY         = DstY
PkC11.wDstX         = DstX
PkC11.wRecWidth     = Width
PkC11.wRecHeight    = Height
PkC11.dwCommand     = (ROP3_S << 8) |              // 823C D[15:8]
                      CMD_ENHCOLOREXP |           // 823C D[3:0]
                      CMD_TRANSPARENT |           // 823C D[20]
                      CMD_ENABLE_CLIP |           // 823C D[18]
                      CMD_ANTIALIAS_EXPANSION |   // 823C D[23:21]
                      EngColorDepth               // 823C D[17:16]

//Fire engine
Fire2DEngine(PktC11)
```