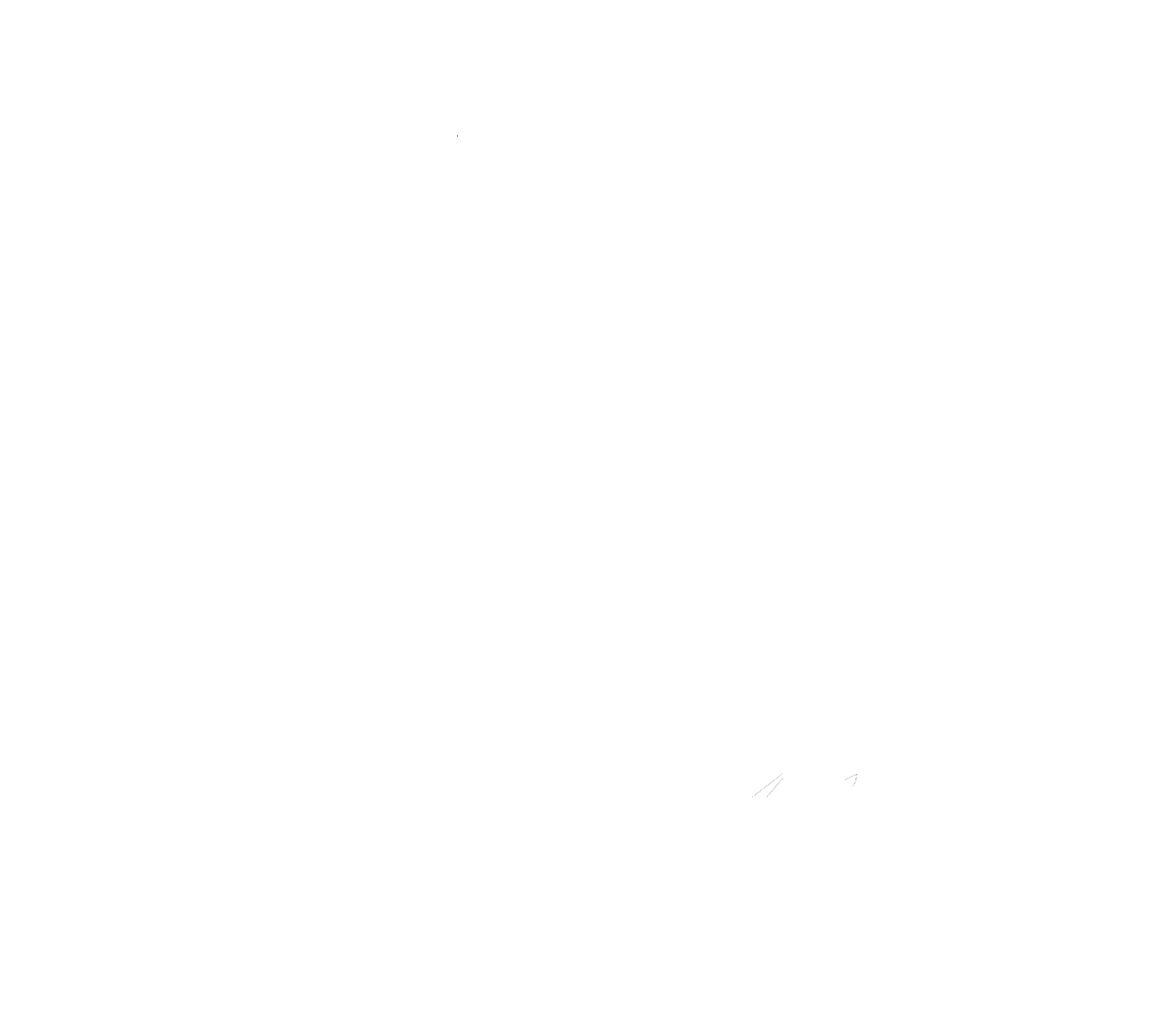
|  |
| --- |
| MICROCOM GPU v1.0 MANUAL |
|  |
| A hardware and software guide to using the Microcom GPU  10103  Authored by: J.Nock |





**Contents**

[Technical Specifications 4](#_Toc94626729)

[Memory Map & Registers 5](#_Toc94626730)

[Memory map for EP4CE10 FPGA: 5](#_Toc94626731)

[Verilog HDL notes: 6](#_Toc94626732)

[Hardware Registers (HW\_REGS) Map 7](#_Toc94626733)

[The TILE/FONT-enabled PDI layer 8](#_Toc94626734)

[Understanding Layer Order, priority, and swapping control logic. 9](#_Toc94626735)

[PALETTES 10](#_Toc94626736)

[FONTS 10](#_Toc94626737)

[Z80 Host IO ports 11](#_Toc94626738)

[Keyboard PS/2 Interface 12](#_Toc94626739)

[KEYBOARD DATA PORT 12](#_Toc94626740)

[KEYBOARD STATUS REGISTER - READ 12](#_Toc94626741)

[KEYBOARD STATUS REGISTER – TYPEMATIC SETTINGS 13](#_Toc94626742)

[Graphics Processing Unit 14](#_Toc94626743)

[COMMAND/DATA WRITE ORDER 14](#_Toc94626744)

[HARDWARE COMMANDS 15](#_Toc94626745)

[X & Y registers – (80-B0 for X[0-3], C0-F0 for Y[0-3]) 15](#_Toc94626746)

[Set destination screen memory pointer (7C-7F) 16](#_Toc94626747)

[Set source screen memory pointer (78-7B) 16](#_Toc94626748)

[Set destination raster width – X[2] (73) 16](#_Toc94626749)

[Set source raster width – X[2] (72) 17](#_Toc94626750)

[Set destination raster width – X[3] (71) 17](#_Toc94626751)

[Set source raster width – Y[3] (70) 17](#_Toc94626752)

[Set max width & height of screen – X[0]/Y[0] (5F) 18](#_Toc94626753)

[Set max width & height of screen – X[1]/Y[1] (5E) 18](#_Toc94626754)

[Set max width & height of screen – X[2]/Y[2] (5D) 18](#_Toc94626755)

[Set max width & height of screen – X[3]/Y[3] (5C) 19](#_Toc94626756)

[Clear pixel collision counter (5B) 19](#_Toc94626757)

[Clear blitter copy pixel collision counter (5A) 19](#_Toc94626758)

[Graphics Commands 20](#_Toc94626759)

[VWAIT – 0x07 21](#_Toc94626760)

[SET HW\_POSITION – 0x0F 22](#_Toc94626761)

[Blitter 23](#_Toc94626762)

[Blitter Copy Width & Height – X[2]/Y[2] & X[3]/Y[3] (74-75) 23](#_Toc94626763)

[Blitter Source Image Offset – X[2]/Y[2] & X[3]/Y[3] (76-77) 23](#_Toc94626764)

[Blitter Copy Source Pointer – X/Y[0-3] (78-7B) 24](#_Toc94626765)

[Blitter Copy Destination Pointer – X/Y[0-3] (7C-7F) 24](#_Toc94626766)

[Source/Destination X/Y Scaling (09) 24](#_Toc94626767)

[How to calculate a zoom 25](#_Toc94626768)

[Copy Colour Transform (08) 26](#_Toc94626769)

[Blitter Function – (00) 27](#_Toc94626770)

[Appendix A – Character Codepage 28](#_Toc94626771)

|  |
| --- |
| Technical Specifications Depending on the FPGA used in the uCOM GPU card, the following features will be available in varying capacities. Listed below are *minimum specifications* based on an Intel  Cyclone V 5CEBA2 FPGA (2nd-generation video card):   * 170KB FPGA graphics RAM, plus 1 KB for palettes * 16MB scratchpad RAM * Up to 20 graphics layers * 24-bit audio * 720x480 resolution in 256 colours at 60 fps (DVI-D/HDMI) * Full HDMI resolution at 60fps * 24-bit true-colour output |
|  |

|  |
| --- |
| Memory Map & Registers The uCOM GPU uses a vacant 512 KB slot in the host system’s memory space – see the uCOM’s manual for more information on this. This 512 KB window is filled with the GPU’s own RAM – if the GPU has less than 512 KB of RAM, accessing memory above the GPU’s upper memory limit will return FF values and be read-only (writes will be ignored).  *Only the last 16 bytes of the last page of the 512KB will return anything other than FF, and this will be the BANK\_ID as per the uCOM’s memory specifications. This is a pseudo-ROM location, with a value returned from a hardwired value in the HDL of the FPGA.*  For GPUs with access to scratchpad RAM, this will be presented as paged-RAM within the uCOM’s 512KB memory window, controlled via IO writes/reads to the GPU Memory Controller. Memory map for EP4CE10 FPGA:  * 0000-01FF – GPU HW registers:   + **00-07** = H&V triggers for 4 yellow test cursors   + **0C-0D** = H&V reset coordinates for all 15 MAGGIE\_Layer#s   + **10-13 + 4\*MAGGIE\_Layer#** = H&V top left edge of each MAGGIE\_Layer# window   + **60 + 10\*MAGGIE\_Layer#** = 16 byte controls for each of the 15 MAGGIE layers * 0200-11FF – Default IBM VGA 8x16 font. * 1200-9FFF – Video RAM (for text buffers / sprite data)   + **1200-1B5F** – Typical memory space for a full screen of text * A000-A3FF – Palettes:   + **A000-A1FF** – primary palette (ARGB4444)   + **A200-A3FF** – secondary palette (RGB565)   Without text you can mix 10 on-screen sprites, or 1 text layer and 8 sprites. Other than a RAM limitation, there are no other output limitations. Any mode and any resolution with any palette may all be mixed simultaneously onscreen. Verilog HDL notes: NUM\_LAYERS = 2 through 15 = 2 layers through 15 layers.  PALETTE\_ADDR = Sets the base address for the 2 palettes.  Palettes occupy the last 1024 bytes of GPU RAM. If the RAM ends on 2^ADDR\_SIZE (i.e. 32768), then (2\*\*ADDR\_SIZE)-1024 can be used to set PALETTE\_ADDR, otherwise PALETTE\_ADDR must be set explicitly to the last 1KB of available GPU RAM. Obviously, this depends on the amount of RAM available in the FPGA used. |
|  |

# Hardware Registers (HW\_REGS) Map

|  |  |  |
| --- | --- | --- |
| HWREG\_BASE\_ADDRESS + | | |
| Off | Bits | Value |
| 00 | 32 | Base address for screen memory. |
| 04 | 8 | Screen enable & bpp. Bit 7 = Enable. Bits 2-0 = bpp. |
| 05 | 8 | Alpha adjust = 100% Opaque *\*8-bit SIGNED value (+127 to -128)* |
| 06 | 16 | Bitmap width. In tile mode, # tiles wide. Align to every 4 bytes. |
| 08 | 16 | Bitmap X offset. |
| 0A | 16 | Bitmap Y offset. |
| 0C | 16 | Window X position. |
| 0E | 16 | Window Y position. |
| 10 | 16 | Window width. |
| 12 | 16 | Window height. |
| 14 | 8 | Scale width. |
| 15 | 8 | Scale height. |
| 16 | 16 | Tile/font base address |
| 18 | 8 | Tile Mode – bit 7 = Enable, Bits 2-0 = Tile bpp. |
| 19 | 8 | Tile Width x Height. Bits 5-4 = width, bits 1-0 = height (0,1,2,3 = 4,8,16,32 pixels) |
| 1A | 8 | Global system 24-bit background colour - RED |
| 1B | 8 | Global system 24-bit background colour - GREEN |
| 1C | 8 | Global system 24-bit background colour - BLUE |
| 1F | 8 | Bits 6-4 = VIDEO\_MODE, bits 2-0 = CLK\_DIVIDER |

|  |  |  |
| --- | --- | --- |
| Window Layer Address Offsets | | |
| Window | HWREG\_BASE\_ADDRESS + offset | Palette Address Offset (PALETTE+) |
| 0 | 000 | 0000h |
| 1 | 020 | 0400h |
| 2 | 040 | 0800h |
| 3 | 060 | 0C00h |
| 4 | 080 | 1000h |
| 5 | 0A0 | 1400h |
| 6 | 0C0 | 1800h |
| 7 | 0E0 | 1C00h |
| 8 | 100 | 2000h |
| 9 | 120 | 2400h |
| 10 | 140 | 2800h |
| 11 | 160 | 2C00h |
| 12 | 180 | 3000h |
| 13 | 1A0 | 3400h |
| 14 | 1C0 | 3800h |
| 15 | 1E0 | 3C00h |

|  |  |  |
| --- | --- | --- |
| HWREG\_BASE\_ADDR\_LSWAP + | | |
| Off | Bits | Value |
| 00 | 8 | On PDI\_layer 0 output, do not swap the sequential order of the SDI\_layers |
| 01 | 8 | On PDI\_layer 1 output, do not swap the sequential order of the SDI\_layers |
| 02 | 8 | On PDI\_layer 2 output, do not swap the sequential order of the SDI\_layers |
| 03 | 8 | On PDI\_layer 3 output, do not swap the sequential order of the SDI\_layers |
| 04 | 8 | On PDI\_layer 4 output, do not swap the sequential order of the SDI\_layers |
| 05 | 8 | On PDI\_layer 5 output, do not swap the sequential order of the SDI\_layers |
| 06 | 8 | On PDI\_layer 6 output, do not swap the sequential order of the SDI\_layers |
| 07 | 8 | On PDI\_layer 7 output, do not swap the sequential order of the SDI\_layers |
| 08 | 8 | During SDI\_layer phase 0, do not swap around any of the PDI\_layers |
| 09 | 8 | During SDI\_layer phase 1, do not swap around any of the PDI\_layers |
| 0A | 8 | During SDI\_layer phase 2, do not swap around any of the PDI\_layers |
| 0B | 8 | During SDI\_layer phase 3, do not swap around any of the PDI\_layers |
| 0C | 8 | During SDI\_layer phase 4, do not swap around any of the PDI\_layers |
| 0D | 8 | During SDI\_layer phase 5, do not swap around any of the PDI\_layers |
| 0E | 8 | During SDI\_layer phase 6, do not swap around any of the PDI\_layers |
| 0F | 8 | During SDI\_layer phase 7, do not swap around any of the PDI\_layers |

# The TILE/FONT-enabled PDI layer

The font/tile layer utilises on-chip FPGA Block RAM to hold its tiles/fonts.

Tile selection when using different ‘**CMD\_vid\_bpp**’ modes, 8/16a/32/16b bpp**\*** modes.

FGC = Foreground color. Adds this FGC value to any tile pixels whose color data is != 0.

BGC = Background color. Replace tile pixels whose color data = 0 with this BGC value.

MIR = Mirror the tile.

FLIP = Vertically flip the tile.

**CMD\_vid\_bpp' mode:**

8 bpp -> Each byte = 1 character, 0 through 255, no color, mirror or flip functions.

**BGC, FGC, Char 0-255** \* BGC & FGC are x16 in this mode

16a bpp -> {4'h0, 4'h0, 8'h00} = 16 bits / 256 possible tiles.

**FLIP, MIR, FGC, Char 0-1023** \* FGC is x16 in this mode

16b bpp -> {1'b0, 1'b0, 4'h0, 10'h000} = 16 bits / 1024 possible tiles.

**BGC, FGC, FLIP, MIR, N/A, Char 0-1023**

32 bpp -> {8'h00, 8'h00, 1'b0, 1'b0, 4'h0, 10'h000} = 32 bits / 1024 possible tiles.

Remember, the contents inside a tile set's '**CMD\_vid\_tile\_bpp**' can be 1/2/4/8/16a/32/16b bpp. **The tile set can only be as large as the reserved, fixed available FPGA Block RAM.** It is possible to have multiple tile layers when using the 'SDI\_LAYERS' feature where each layer may share or have different tile sets so long as there is enough room in the single reserved FPGA block ram.

\* On a tile layer, bpp will actually mean bpc -> Bits Per Character Tile

## Understanding Layer Order, priority, and swapping control logic.

For now, layer 0 is on top and each higher layer number progressively is below / lower priority. Don't forget that if the bottom layer has transparent pixels right through the top layer, then the BGC set will become the displayed colour.

With 16 layers, parameters are:

PDI\_LAYERS = 4

SDI\_LAYERS = 4

This means that the layers 0 through 15 =

PDI layer 0 -> SDI 0,1,2,3 = Layer 0,1,2,3

PDI layer 1 -> SDI 0,1,2,3 = Layer 4,5,6,7

PDI layer 2 -> SDI 0,1,2,3 = Layer 8,9,10,11

PDI layer 3 -> SDI 0,1,2,3 = Layer 12,13,14,15.

Note that to save on Block RAM resources, the TILE/FONT mode & memory is only available for PDI\_LAYER 0. This means the tile/font mode is only functional on layers 0, 1, 2 & 3. The available RAM for the font is set to 65536 bytes. This means a 256-character 16x16 pixel font with 8 bpp, i.e., 256 colours per pixel font is possible.

The 'CLK\_DIVIDER' must be at least =3 to have all SDI layers 0, 1, 2 & 3 functional.

# PALETTES

The palettes are held in GPU RAM at a location specified in the HDL, defaulting to 0x1000 – which equates to Bank 0x40, offset +0x1000. The palettes are 8-bit RBGA, so four bytes for each palette colour entry (with each channel – red, green, blue and alpha – having values from 0-255).

Each palette is 256x4 bytes/1KB/0x400 bytes in size and there are 16 in total, one for each Layer.

|  |  |  |
| --- | --- | --- |
| PDI Layer | Offset (Hex) | Address |
| 0 | +0000 | 1000 |
| 1 | +0400 | 1400 |
| 2 | +0800 | 1800 |
| 3 | +0C00 | 1C00 |
| 4 | +1000 | 2000 |
| 5 | +1400 | 2400 |
| 6 | +1800 | 2800 |
| 7 | +1C00 | 2C00 |
| 8 | +2000 | 3000 |
| 9 | +2400 | 3400 |
| 10 | +2800 | 3800 |
| 11 | +2C00 | 3C00 |
| 12 | +3000 | 4000 |
| 13 | +3400 | 4400 |
| 14 | +3800 | 4800 |
| 15 | +3C00 | 4C00 |

# FONTS

A default font is held in the GPU; however, a system font is also held in the host system’s ROM and is copied to the GPU’s RAM at startup, allowing custom fonts to be used. The font table can reside anywhere in the GPU’s RAM, with software specifying its location.

# Z80 Host IO ports

The Z80\_Bridge in the GPU exposes some specific IO addresses to the host to facilitate extra hardware functions, listed below. The IO addresses are separate from the GPU RAM address space and are easily customizable in Quartus via the parameter settings in the Z80\_bridge.

|  |  |  |  |
| --- | --- | --- | --- |
| **IO Address** | **Read/Write** | **Function** | More on page |
| **240 / F0** | Read-only | KEYBOARD DATA port (*keyboard interface*) | 15 |
| **241 / F1** | Read/Write | KEYBOARD STATUS register (*keyboard interface*) | 15 |
| **242 / F2** | Write-only | SPEAKER ENABLE (0 = off, >0 = on) |  |
| **243 / F3** | Write-only | VIDEO OUTPUT ENABLE (0 = off, >0 = on) |  |
| **244 / F4** | Write-only | TONE (frequency) REGISTER for speaker (0-255) |  |
| **245 / F5** | Write-only | DURATION REGISTER for speaker (0-255) |  |
| **246 / F6** | Write-only | GPU LBR – Low-Byte Register (0-255) | 17 |
| **247 / F7** | Write-only | GPU HBR – High-Byte Register (0-255) | 17 |
| **248 / F8** | Read-only | GPU FIFO Status (FIFO\_Full on bit 0) |  |

# Keyboard PS/2 Interface

The uCOM GPU provides far more than just a video output terminal. The GPU card also provides a keyboard terminal to fully release the uCOM from its dependency on a serial link to a host PC.

## KEYBOARD DATA PORT

This 8-bit data port returns the ASCII character with current MAKE or BREAK code (bit 7 set for BREAK, cleared for MAKE). ASCII characters are returned as a result of a key being pressed on the keyboard attached to the GPU card. Once read, the KEYBOARD DATA PORT is set to zero – any further reads will return 0x00 unless another MAKE or BREAK code is ready to be transmitted.

## KEYBOARD STATUS REGISTER - READ

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| TX | TX ACK | TX ERR | SHIFT | CAPS | EXT CODE | BREAK | VALID\_DATA |

When read, the KEYBOARD STATUS register provides information on the current state of the KEYBOARD DATA PORT and attached keyboard:

* Bit 0 – HIGH if valid data is available on the DATA port.
* Bit 1 – HIGH if the ASCII character at the DATA port is a BREAK code (i.e. the result of the key being released on the keyboard). *BREAK codes are generally filtered out by the DMI and CP/M BIOS (unless special code is written to make use of them) by ignoring characters with bit 7 set, or status bytes with bit 1 set.*
* Bit 2 – HIGH if the data is an EXTENDED CODE – another byte should be incoming.
* Bit 3 – HIGH if CAPS LOCK is active.
* Bit 4 – HIGH if SHIFT is depressed.
* Bit 5 – HIGH if no TX ACK received from keyboard in response to a command sent to it.
* Bit 6 – HIGH for TX acknowledge from keyboard
* Bit 7 – HIGH whilst data is transmitted to the keyboard

## KEYBOARD STATUS REGISTER – TYPEMATIC SETTINGS

Writing to the KEYBOARD STATUS register will set the keyboard’s *Typematic Delay* and *Repeat Rate*. In this way, the keyboard’s behaviour can be customized by the user as desired by simply writing a byte to the appropriate I/O address.

The delay and repeat rate are set in the same byte, which his composed as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | Delay | | Repeat Rate | | | | |

|  |  |
| --- | --- |
| Bits 5-6 | Delay (seconds) |
| 00b (00h) | 0.25 |
| 01b (20h) | 0.50 |
| 10b (40h) | 0.75 |
| 11b (60h) | 1.00 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bits 0-4 | Rate (cps) | Bits 0-4 | Rate (cps) | Bits 0-4 | Rate (cps) | Bits 0-4 | Rate (cps) |
| 00h | 30.0 | 08h | 15.0 | 10h | 7.5 | 18h | 3.7 |
| 01h | 26.7 | 09h | 13.3 | 11h | 6.7 | 19h | 3.3 |
| 02h | 24.0 | 0Ah | 12.0 | 12h | 6.0 | 1Ah | 3.0 |
| 03h | 21.8 | 0Bh | 10.9 | 13h | 5.5 | 1Bh | 2.7 |
| 04h | 20.7 | 0Ch | 10.0 | 14h | 5.0 | 1Ch | 2.5 |
| 05h | 18.5 | 0Dh | 9.2 | 15h | 4.6 | 1Dh | 2.3 |
| 06h | 17.1 | 0Eh | 8.6 | 16h | 4.3 | 1Eh | 2.1 |
| 07h | 16.0 | 0Fh | 8.0 | 17h | 4.0 | 1Fh | 2.0 |

To work out a byte value to write to the register, simply add the hexadecimal values of your chosen settings together from the tables above. For example, if you want a delay of 0.50 seconds and a repeat rate of 10.0, add 20h and 0Ch together to get 2Ch. Write this byte to the PS/2 STATUS register and off you go.

NOTE: Valid values range from 00h-7Fh. Bit 7 should always be zero.

# Graphics Processing Unit

The GPU is a powerful hardware module in the FPGA that performs basic graphics functions on behalf of the host system – including:

1. Pixel plotting (X[0], Y[0])
2. Line drawing (X[0], Y[0] to X[1], Y[1])
3. Filled/unfilled rectangle (X[0], Y[0] to X[1], Y[1])
4. Filled/unfilled triangle (X/Y[0], X/Y[1], X/Y[2])
5. Filled/unfilled quadrilaterals (X/Y points [0] to [3])
6. Blitter copy/paste with rotation & scaling

As well as the primary drawing functions, there are several support functions:

* Set 24-bit destination screen memory pointer
* Set 24-bit source screen memory pointer
* Set byte-width for destination raster (in x[n], y[n]) (n = 2 to 3)
* Set byte-width for source raster (in x[n], y[n]) (n = 2 to 3)
* Set maximum bitmap width & height to x[n], y[n] (n = 0 to 3)
* Clear pixel collision counter
* Clear blitter copy pixel collision counter

## COMMAND/DATA WRITE ORDER

The GPU has a 16-bit interface, driven by dual 8-bit IO writes from the host in little-endian format – the LOW byte is always written first to IO port 246, then the HIGH byte is written to IO port 247, which causes the GPU to act on the received 16-bit data/command. Data/commands must be written in 16-bit pairs – even if a value of zero is required for a command, the low byte should have zero written to it before writing to the High-Byte Register. This ensures the Low-Byte Register does not insert garbage values into the code.

## HARDWARE COMMANDS

### X & Y registers – (80-B0 for X[0-3], C0-F0 for Y[0-3])

There are four 12-bit X and Y registers, X[0-3], Y[0-3]. Their contents are set using the 8-bit Low-Byte Register (LBR) and High-Byte Register (HBR) which combine to create a 16-bit input. As with all data sent to the GPU, the low byte must be written first to the LBR at port 246, followed by the high byte to the HBR at port 247.

NOTE: The command bytes (sent to the High-Byte Register) may also include data in their lower nybble to allow passing 12-bit values with one command.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE (HBR) | | | | | | | | LOW BYTE (LBR) | | | | | | | |
| High Nybble | | | | Low Nybble | | | | High Nybble | | | | Low Nybble | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | R | n | n | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |

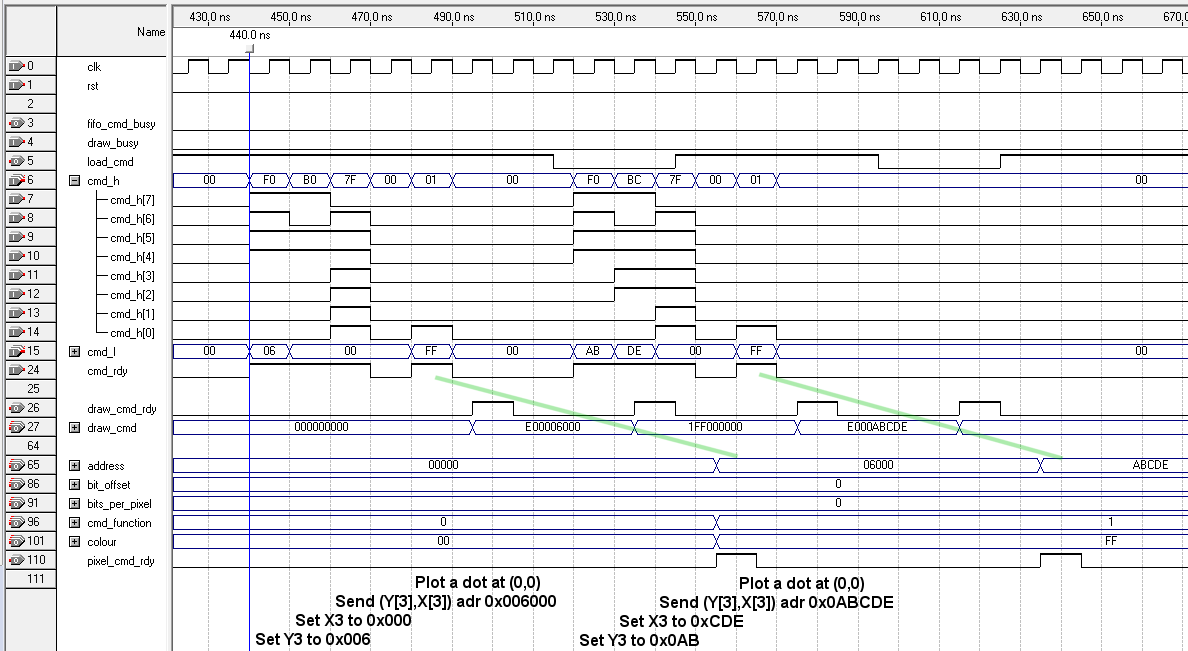
R – HBR bit 6 (register bit 14) – Register Select

This bit determines which register – X or Y – is being written to. 0 = X, 1 = Y.

nn – HBR bits 5 & 4 (register bits 13 & 12)

These bits specify which of the four X/Y registers to write to.

? – bits 0-11 are 12 data bits to be written to the specified register. Room for three nybbles (or three hex digits). Note that 4 bits in the HBR (green boxes) can also be used to include data, e.g., when setting a base memory address.

In the example below, a base address of ABCDE is set by writing 0x0AB to Y[3] with the F0AB command word, and 0xCDE to X[3] with the BCDE command word.

### Set destination screen memory pointer (7C-7F)

The following command sets the destination screen memory pointer to the data in the indicated 12-bit registers. LBR value is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | n | n | x | x | x | x | x | x | x | x |

* nn – specify which of the four X/Y registers to use the data from
* x – don’t care – these bits are ignored

### Set source screen memory pointer (78-7B)

The following command sets the source screen memory pointer to the data in the indicated 12-bit registers. LBR value is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | n | n | x | x | x | x | x | x | x | x |

* nn – specify which of the four X/Y registers to use the data from
* x – don’t care – these bits are ignored

### Set destination raster width – X[2] (73)

Sets the destination raster’s bytes-per-horizontal line to the data in the X[2] register. Y register is ignored. Low byte bits 2-0 specify bits-per-pixel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | x | x | x | x | x | bpp | | |

* x – don’t care – these bits are ignored
* bpp – bits per pixel

### Set source raster width – X[2] (72)

Sets the source raster’s pixels-per-horizontal line to the data in the X[2] register. Y register is ignored. Low byte bits 2-0 specify bits-per-pixel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | x | x | x | x | x | bpp | | |

* x – don’t care – these bits are ignored
* bpp – bits per pixel

### Set destination raster width – X[3] (71)

Sets the source raster’s pixels-per-horizontal line to the data in the X[3] register. Y register is ignored. Low byte bits 2-0 specify bits-per-pixel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | x | x | x | x | x | bpp | | |

* x – don’t care – these bits are ignored
* bpp – bits per pixel

### Set source raster width – Y[3] (70)

Sets the source raster’s bytes-per-horizontal line to the data in the Y[3] register. X register is ignored. Low byte bits 2-0 specify bits-per-pixel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | x | x | x | x | x | bpp | | |

* x – don’t care – these bits are ignored
* bpp – bits per pixel

### Set max width & height of screen – X[0]/Y[0] (5F)

Sets the bitmap dimensions to the data in the 0-index registers (X[0], Y[0]). LBR value is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | x | x | x | x | x | x | x | x |

x – don’t care – these bits are ignored.

### Set max width & height of screen – X[1]/Y[1] (5E)

Sets the bitmap dimensions to the data in the 1-index registers (X[1], Y[1]). LBR value is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | x | x | x | x | x | x | x | x |

x – don’t care – these bits are ignored.

### Set max width & height of screen – X[2]/Y[2] (5D)

Sets the bitmap dimensions to the data in the 0-index registers (X[0], Y[0]). LBR value is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | x | x | x | x | x | x | x | x |

x – don’t care – these bits are ignored.

### Set max width & height of screen – X[3]/Y[3] (5C)

Sets the bitmap dimensions to the data in the 3-index registers (X[3], Y[3]). LBR value is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | x | x | x | x | x | x | x | x |

x – don’t care – these bits are ignored.

### Clear pixel collision counter (5B)

Clears the pixel collision counter and sets all three transparent mask colours to one 8-bit colour in the source function data.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | ? | ? | ? | ? | ? | ? | ? | ? |

? – sets the mask colour.

### Clear blitter copy pixel collision counter (5A)

Clears the blitter copy pixel collision counter and sets all three transparent mask colours to one 8-bit colour in the source function data.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | ? | ? | ? | ? | ? | ? | ? | ? |

? – sets the mask colour.

# Graphics Commands

Following is a summary of the graphics commands that can be sent to the GPU using the hardware commands listed previously via IO calls, or whatever interface your system uses. All commands are 16-bit, with the low byte sent first.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Command | High Byte | Low Byte | Notes |
| 1 | Draw Pixel | 01 | Colour | Draws a pixel at X/Y[0]. |
| 2 | Draw Line | 02 | Colour | Draws a line from X/Y[0] to X/Y[1]. |
| 3 | Draw Triangle | 03 | Colour | Draws a triangle with points X/Y[0], [1] and [2]. |
| 4 | Draw Rectangle | 04 | Colour | Draws a rectangle, top-left corner at X/Y[0], bottom-right corner at X/Y[1]. |
| 5 | Draw Quad | 05 | Colour | Draws a quad, corners specified using all four X/Y registers. |
| 6 | Draw Arc | 06 | Colour | Draws an arc (quarter of an ellipse based on quadrant # set in next command). Uses X/Y[0] for centre, X/Y[1] to set X & Y radii. To draw a complete ellipse or circle, this must be executed four times, once for each quadrant. |
| 7 | Quadrant Select | 07 | 0x00-0x03 | Set the ellipse quadrant to draw. This must be set before Draw Arc is executed. To draw a complete ellipse or circle, this must be executed four times, once for each quadrant. |
| 7 | VWAIT | 07 | 0x8#-0xB#  (# = 1-F) | 0x8# .. 0x8# => Wait for hw\_position\*1 to pass by # frames.  0x9# .. 0x9# => Wait for hw\_position\*2 to pass by # frames.  0xA# .. 0xA# => Wait for hw\_position\*4 to pass by # frames.  0xB# .. 0xB# => Wait for hw\_position\*8 to pass by # frames. |
| … |  |  |  |  |
| 11 | Filled Triangle | 0B | Colour | Draws filled triangle, points at X/Y[0], [1] and [2]. |
| 12 | Filled Rectangle | 0C | Colour | Draws filled rectangle, top-left corner at X/Y[0], bottom-right corner at X/Y[1]. |
| 13 | Filled Quad1 | 0B  0D | Colour  Colour | Drawing a quad utilises the Filled Triangle command and the alternative Filled Triangle command that uses points from registers 0, 2 and 3. Point order must be clockwise. |
| 14 | Filled Arc | 0E | Colour | As Draw Arc, except filled. |
| 15 | Set hw\_position | 0F | hw\_position | Used with VWAIT. Default 0. |

1 – This hybrid command requires two sets of 16-bit commands to be sent to the IO port

## VWAIT – 0x07

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | Multiplier | | Multiplicand | | | |

The VWAIT command tells the GPU to wait a specified time (using a multiple of the *hw\_position* value) before executing the next instruction. *VWAIT shares the same command High Byte as Quadrant Select; Low Byte values 0x00-0x03 are strictly reserved for Quadrant Select, valid range for VWAIT’s Low Byte is 0x81-0xBF.*

The multiplier is set by bits 5 and 4 in the Low Byte of the command to x1, x2, x4 or x8 – with the multiplicand being the least-significant nybble (valid range: 0x01-0x0F).

|  |  |  |
| --- | --- | --- |
| **Multiplier** | **Multiplicand** |  |
| **0x8** | 0x81-0x8F | Wait for (*hw\_position* x 1) to pass by # frames. |
| **0x9** | 0x91-0x9F | Wait for (*hw\_position* x 2) to pass by # frames. |
| **0xA** | 0xA1-0xAF | Wait for (*hw\_position* x 4) to pass by # frames. |
| **0xB** | 0xB1-0xBF | Wait for (*hw\_position* x 8) to pass by # frames. |

## SET HW\_POSITION – 0x0F

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | hw\_position | | | | | | | |

Sets the *hw\_position* value used with the VWAIT command.

*hw\_position* defaults to zero, unless set to another value using this command. A value of zero means the vertical wait position is one line after the display frame has ended, giving the maximum time before the next frame begins to draw.

To request a delay before executing the next command, the multiplicand set with the VWAIT command needs to be 1 or higher to wait multiple frames. A *hw\_position* of 100 and using VWAIT with a Low Byte value of 0xA1 means the pause position will be located 400 line/h-syncs after the end of the display frame.

Values of *hw\_position* larger than the available lines in the frame will cause VWAIT to abort after a 2-frame pause.

# Mosaic,grid,pixel,pattern,decoration - free image from needpix.comBlitter

The GPU has a fully functional blitter to massively speed up sprite operations (block copy & paste). *All draw commands will blit.* If you use command '0x01' (plot pixel), the paste will begin at coordinates X/Y[0], or be center-shifted depending on set width & height when ‘Center Paste’ is enabled (see page 30). If you use the line command '0x02', the blitter will paste a new copy of the source image for every single pixel in the source line coordinates X/Y[0] - X/Y[1], using the source image like a brush.

### Blitter Copy Width & Height – X[2]/Y[2] & X[3]/Y[3] (74-75)

Sets the bitmap dimensions to copy – width is set by the X register contents, height by Y. Bit 0 in the High Byte specifies which register set to use (2/3). Each dimension has 1 added (e.g. X[2] = 0 would give a width of 1). Low byte is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | n | x | x | x | x | x | x | x | x |

* n – specifies which register set (low = X/Y[2], high = X/Y[3])
* x – don’t care – these bits are ignored

### Blitter Source Image Offset – X[2]/Y[2] & X[3]/Y[3] (76-77)

Sets the source image (copy/read) offset. Bit 0 in the High Byte specifies which register set to use (2/3). Low byte is ignored.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | n | x | x | x | x | x | x | x | x |

* n – specifies which register set (low = X/Y[2], high = X/Y[3])
* x – don’t care – these bits are ignored

### Blitter Copy Source Pointer – X/Y[0-3] (78-7B)

Sets the blitter memory source pointer – using the 12-bit Y (MSW) and X (LSW) register contents. Bits 0-1 in the High Byte specifies which register set to use (0-3). Low byte in the command word specifies the source *screen mode*, or bits per pixel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | n | n | Screen\_mode (bpp)  x  x  x  x  x  x  x | | | | | | | |

* nn – specifies which register set (X/Y[0-3])

### Blitter Copy Destination Pointer – X/Y[0-3] (7C-7F)

Sets the blitter memory destination pointer – using the 12-bit Y (MSW) and X (LSW) register contents. Bits 0-1 in the High Byte specifies which register set to use (0-3). Low byte in the command word specifies the destination *screen mode*, or bits per pixel.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | n | n | Screen\_mode (bpp)  x  x  x  x  x  x  x | | | | | | | |

* nn – specifies which register set (X/Y[0-3])

### Source/Destination X/Y Scaling (09)

Sets the blitter’s scaling function, enabling up- and down-scaling of bitmaps during copy/pasting or disabling scaling altogether.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | x | x | x | x | x | x | f | f |

* f – specifies scaling option (see table below)
* x – don’t care – these bits are ignored

Bits 0 & 1 of the Low Byte specify the scaling features as per the table below:

|  |  |  |
| --- | --- | --- |
| **Bit** | **Function** | **Details** |
| 0 | Up-scaling | HIGH – Enables blitter up-scaling of the source image, specified by registers X[0] and Y[0] for width & height accordingly. |
| 1 | Down-scaling | HIGH – Enables blitter down-scaling of the pasted image, specified by registers X[1] and Y[1] for width & height accordingly. |

**How to calculate a zoom**

Valid scale factor settings range from 1-4096. Values outside this range will default the setting back to 4096.

To enlarge the source bitmap, set the down-scaling to 4096 (1:1, or no down-scaling) in the appropriate X/Y[1] registers. Then set the source up-scaling using this ratio: 4096:(set X/Y[0])

For example, if you want your image to be 1:1, then use a down-scale setting of 4096 and an up- scale setting of 4096. If you want your image to be double size, 2:1, then use an up- scale setting of 2048. If you want your image to be 2.32992:1, then use an up- scale setting of 879.

To shrink the destination bitmap, set the up-scale value to 4096. Then set the down- scale value using this ratio: (set X/Y[1]): 4096

If you want your pasted bitmap to be 1:1, then set the down-scale setting to 4096.  
If you want your image to be half size, e.g., 1:2, then use a down-scale setting of 2048.  
If you want your image to be 0.707:1, or, 1:1.41421, then use a down sample setting of 2896 (this value is relevant to 45-degree rotations on blitted bitmaps).

*You may mix different X&Y scale figures including a zoom on one axis and a shrink on the other. Note that the H&V center paste will not work properly with any zoom setting other than 1:1. This is because to obtain the proper center offset based on the source width and height, we need to calculate the reciprocal of the source scale setting to determine the size of the final image which can give us the center offset of that image.  A reciprocal is long division and there is no more space in the current FPGA.*

### Copy Colour Transform (08)

Sets a colour transformation on the blitter copy command. This causes the blitter to XOR read pixels with the Low Byte of this command.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | m | m | m | m | m | m | m | m |

* m – 8-bit colour mask value

|  |  |  |
| --- | --- | --- |
| **INPUT** | | **OUTPUT** |
| **A** | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

*When using the mask feature the transparent colour is '0x00'. XORing the read pixels with a custom number here other than '0x00' will change the read colour which will simultaneously change what colour will become transparent when a paste/write pixel is done with mask enable.*

The chosen drawing colour, in the Low Byte when the drawing command is sent to the GPU, will perform a second XORing of the first XORed pixel copy, then allow you to change the colour once again before the pixel is written.

So, if the source image is monochrome 1-bit color and the destination screen is 4-bit 16 colours, using a number like 0x04 to XOR the read values, will make the output colour 0x04 and 0x05 instead of 0x00 and 0x01.

Be careful not to choose a XOR number larger than the available colours on the destination screen, otherwise garbage will bleed into adjacent pixels.

### Blitter Function – (00)

Enables the blitter and sets its function according to bits 0-2 of the Low Byte (values 0-7):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| HIGH BYTE | | | | | | | | LOW BYTE | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | x | x | f | f | f |

* f – sets required function – see table below
* x – don’t care – these bits are ignored

|  |  |  |
| --- | --- | --- |
| **Bit** | **Function** | **Details** |
| 0 | Blitter | HIGH – Enables blitter, copying source to output coordinates.  LOW – Blitter disabled, normal pixel write commands. |
| 1 | Paste Mask | HIGH – Pastes pixels with transparency mask.  LOW – Paste pixels even if the source has the selected transparent colour. |
| 2 | Centre Paste | HIGH – Offset the paste up and to the left by half of blit\_width/height  LOW – Use the paste coordinates as the top-left. |

# Appendix A – Character Codepage

