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



10103

Contents

[Technical Specifications 4](#_Toc49964482)

[Memory Map & Registers 5](#_Toc49964483)

[Memory map for EP4CE10 FPGA: 5](#_Toc49964484)

[Verilog HDL notes: 6](#_Toc49964485)

[MAGGIEs 7](#_Toc49964486)

[MAGGIE Registers 8](#_Toc49964487)

[BP2RAST\_cmd 9](#_Toc49964488)

[Bits 7-6 – Text\_mode\_master / text\_mode\_slave 9](#_Toc49964489)

[Bit 4 – mode\_565 9](#_Toc49964490)

[Bit 3 – BART enable 9](#_Toc49964491)

[Bit 2 – 16\_bit\_mode 9](#_Toc49964492)

[Bits 1-0 – Video Mode / pixels per byte 9](#_Toc49964493)

[Text Mode 10](#_Toc49964494)

[DEFAULT MAGGIE SETTINGS 12](#_Toc49964495)

[PALETTES 13](#_Toc49964496)

[ARGB4444 Palette 13](#_Toc49964497)

[Host IO ports 14](#_Toc49964498)

[Keyboard Interface 15](#_Toc49964499)

[PS/2 DATA PORT 15](#_Toc49964500)

[PS/2 STATUS REGISTER - READ 15](#_Toc49964501)

[PS/2 STATUS REGISTER – TYPEMATIC SETTINGS 16](#_Toc49964502)

[Hardware-Accelerated Graphics engine 17](#_Toc49964503)

[COMMAND/DATA WRITE ORDER 17](#_Toc49964504)

[HARDWARE COMMANDS 18](#_Toc49964505)

[X & Y registers 18](#_Toc49964506)

[Set destination screen memory pointer (124-127) 19](#_Toc49964507)

[Set source screen memory pointer (120-123) 19](#_Toc49964508)

[Set destination raster bytes-per-horizontal line – X[2] (115) 19](#_Toc49964509)

[Set source raster bytes-per-horizontal line – Y[2] (114) 20](#_Toc49964510)

[Set destination raster bytes-per-horizontal line – X[3] (113) 20](#_Toc49964511)

[Set source raster bytes-per-horizontal line – Y[3] (112) 20](#_Toc49964512)

[Set max width & height of screen – X[0]/Y[0] (95) 21](#_Toc49964513)

[Set max width & height of screen – X[1]/Y[1] (94) 21](#_Toc49964514)

[Set max width & height of screen – X[2]/Y[2] (93) 21](#_Toc49964515)

[Set max width & height of screen – X[3]/Y[3] (92) 22](#_Toc49964516)

[Clear pixel collision counter (91) 22](#_Toc49964517)

[Clear blitter copy pixel collision counter (90) 22](#_Toc49964518)

[HAG Design – To Do: 23](#_Toc49964519)

[Appendix A – Character Codepage 24](#_Toc49964520)

|  |
| --- |
| 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 EP4CE6 FPGA (used in development of the prototype GPU):   * 22KB FPGA graphics RAM, plus 1 KB for palettes * Up to 7 graphics layers * 640x480 resolution at 60 fps (VGA) * Various palettes from 4,096 colours   An Intel EP4CE10 FPGA (used in the first proper video card) provides the following:   * 40KB FPGA graphics RAM, plus 1 KB for palettes * Up to 10 graphics layers * 640x480 resolution in 2 colours at 60 fps (VGA)   **NOTE:** *All 16-bit words in the uCOM GPU are big-endian*. |
|  |

|  |
| --- |
| 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.* 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. |
| MAGGIEs The **M**ultiple **A**ddress **G**enerator and **G**raphic **I**nstruction **E**ngines are the workhorses of the GPU. There are 10 MAGGIEs in the smallest system, numbered 0-9.  All MAGGIEs have their HV resets set to HV\_Trigger 0C & 0D.  MAGGIE top-left window X/Y positions:   * MAGGIE0 at HV\_Trigger 0010-11 & 0012-13 * MAGGIE1 at HV\_Trigger 0014-15 & 0016-17 * MAGGIE2 at HV\_Trigger 0018-19 & 001A-1B * MAGGIE3 at HV\_Trigger 001C-1D & 001E-1F * MAGGIE4 at HV\_Trigger 0020-21 & 0022-23 * MAGGIE5 at HV\_Trigger 0024-25 & 0026-27 * MAGGIE6 at HV\_Trigger 0028-29 & 002A-2B * MAGGIE7 at HV\_Trigger 002C-2D & 002E-2F * MAGGIE8 at HV\_Trigger 0030-31 & 0032-33 * MAGGIE9 at HV\_Trigger 0034-35 & 0036-37   MAGGIE register addresses – 16 bytes for each MAGGIE:   * MAGGIE0 – 0060 * MAGGIE1 – 0070 * MAGGIE2 – 0080 * MAGGIE3 – 0090 * MAGGIE4 – 00A0 * MAGGIE5 – 00B0 * MAGGIE6 – 00C0 * MAGGIE7 – 00D0 * MAGGIE8 – 00E0 * MAGGIE9 – 00F0 |

## MAGGIE Registers

|  |  |  |
| --- | --- | --- |
| Address | Name | Function |
| Base + 00 | BP2RAST\_cmd | 8-bit composite. Video Mode setting. |
| Base + 01 | BP2RAST\_bgc | Background colour. |
| Base + 02 | BP2RAST\_fgc | Foreground colour. |
| Base + 03 | RST\_ADDR\_H | MSB bits of 24-bit base read address. |
| Base + 04 | RST\_ADDR\_M | MID bits of 24-bit base read address. |
| Base + 05 | RST\_ADDR\_L | LSB bits of 24-bit base read address. |
| Base + 06 | YINC\_ADDR\_H | MSB of bytes per line (*varies with video mode*) |
| Base + 07 | YINC\_ADDR\_L | LSB of bytes per line (*varies with video mode*) |
| Base + 08 | X\_SIZE\_H | MSB of display width in pixels. |
| Base + 09 | X\_SIZE\_L | LSB of display width in pixels. |
| Base + 0A | Y\_SIZE\_H | MSB of display height screen lines. |
| Base + 0B | Y\_SIZE\_L | LSB of display height screen lines. |
| Base + 0C | X\_SCALE | Two 4-bit nybbles. Upper 4 bits control X-increment every period, lower 4 is pixel period counter to define number of pixels until upper 4 bits are added to the address pointer. |
| Base + 0D | Y\_SCALE | Two 4-bit nybbles. Upper 4 bits reserved for text tile mode Y-size; lower 4 bits is line period counter to define number of lines until YINC\_ADDR is added to address pointer. |
| Base + 0E | X\_START\_SUB | 8-bit word. Defines an odd pixel start position within the period counter and X-position within a bit plane’s X-coordinate position. |
| Base + 0F | Y\_START\_SUB | 8-bit word. Defines an odd line start within the period counter and Y-coordinates inside a font. |

### BP2RAST\_cmd

The BP2RAST\_cmd register is composed of the following bit settings:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| text\_mode\_master | text\_mode\_slave | not used | mode\_565 | bart\_enable | 16\_bit\_mode | Video Mode | Video Mode |

### Bits 7-6 – Text\_mode\_master / text\_mode\_slave

When using text/font/tile mode, two adjacent MAGGIEs must be used. The first one must have the '*text\_mode\_master*' bit set in the '*BP2RAST\_cmd*' while the second MAGGIE must have the '*text\_mode\_slave*' bit set in the '*BP2RAST\_cmd*'. These bits must be zero for graphics output.

### Bit 4 – mode\_565

### Bit 3 – BART enable

This bit must be HIGH for the linked BART to output pixel data. If the bit is LOW, the MAGGIE/BART pipeline is effectively turned off, unless the MAGGIE is a text\_mode\_master.

### Bit 2 – 16\_bit\_mode

If HIGH, the BART works in 16-bit mode (so all addresses must be even) to use two-byte words when reading the pixel data.

### Bits 1-0 – Video Mode / pixels per byte

Determines the Video Mode, or pixels per byte. The table below shows how bits 2-0 work together to determine the Video Mode:

|  |  |  |  |
| --- | --- | --- | --- |
| Bit 2 | Bit 1 | Bit 0 | Video Mode |
| 0 | 0 | 0 | 1-bit colour |
| 0 | 0 | 1 | 2-bit colour |
| 0 | 1 | 0 | 4-bit colour |
| 0 | 1 | 1 | 8-bit colour |
| 1 | 0 | 0 | 16-bit, 8 pixels-per-word |
| 1 | 0 | 1 | 16-bit, 4 pixels-per-word |
| 1 | 1 | 0 | 16-bit, 2 pixels-per-word |
| 1 | 1 | 1 | 16-bit, true colour |

## Text Mode

Two MAGGIEs are required for text mode – a master and a slave – and they must be adjacent to each other (i.e. MAGGIE0 & MAGGIE1, MAGGIE3 & MAGGIE4 etc).

The master MAGGIE is responsible for the positioning of the window on the screen and pointing to the source data (*screen buffer*), whereas the slave MAGGIE controls the foreground and background colours.

The screen buffer location is specified in the master MAGGIE in the 24-bit word at register addresses +3 to +5.

The Video Mode in both MAGGIEs must match, except in the master where the BP2RAST\_cmd BART\_enable (bit 3) must be disabled; otherwise you will get junk on the screen.

In 16-bit text mode, each character tile is followed by a palette byte. The upper 4-bits control the foreground colour, the lower 4-bits control the background colour.

All other controls are the same as when using it as a bitmap graphics window.

The '*font\_y\_size*', combined upper bits of '{hw\_regs[Y\_START\_SUB][7:6], hw\_regs[Y\_SCALE][7:4]}' in the *text\_mode\_master* which must also be set to the font/tile height -1 (e.g.: 0=1 line, 7=8 lines, 15=16 lines, 31=32 lines) when in text/font/tile master mode.

For the *'text\_mode\_slave*' MAGGIE, the only functioning 4 controls are:

1. reset\_addr which points to the base memory address for font/tile graphics data,
2. period\_x[4] which defines a font as 16 pixels wide when set, or 8 pixels when low, and period\_x[3:0] which mask out the upper bits for fonts/tile addressing for 256, 512, 1024, 2048, 4096 characters (used in 16 bit addressing mode, otherwise set these bits to 0),
3. period\_y[1:0] which defines the font height (0=8, 1=16, 2=32, 3=64 lines tall), and
4. BP2RAST\_cmd which should match the *text\_mode\_master* MAGGIE settings except for the master and slave bit.

All base memory address settings should be on an even byte when using 16-bit mode. Fonts/tile graphics data should always begin on an even byte address.

## DEFAULT MAGGIE SETTINGS

Example default MAGGIE settings for the ‘chip’ image:

Top image: MAGGIE4 - HW triggers @ 20-21 & 22-23

A0 - 1A 10 00 00 33 00 00 60 00 BF 00 77 00 00 00 00

Middle image: MAGGIE5 - HW triggers @ 24-25 & 26-27

B0 - 1A 70 00 00 33 00 00 60 00 BF 00 77 01 01 00 00

Background image: MAGGIE6 - HW triggers @ 28-29 & 2A-2B

C0 - 0A B0 00 00 33 0A 00 60 00 BF 00 77 03 03 00 00

# PALETTES

The palettes are held in memory at $9C00-$9FFF and are 512 entries wide, with the 4444 colour palette occupying the first 256x16 entries while the 565 palette occupies the second 256x16 entries. Both palettes are 16-bit.

## ARGB4444 Palette

This is the default palette for Text Mode, requiring two bytes per palette entry. As it is commonly used with Text Mode, which can specify a maximum of 16 index values, it is generally divided into separate palettes of 16 entries, with the first 16 being the default palette.

Each entry consists of two bytes, as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| 1st BYTE | | 2nd BYTE | |
| 4-bits | 4-bits | 4-bits | 4-bits |
| Transparency | RED | GREEN | BLUE |

Four pages of Bank 0x42 contains the palette, with the first 32 bytes containing the default palette. If Bank 0x42 is mapped to Area 3, the palette memory starts at 0xE000 and ends at 0xE3FF.

# Host IO ports

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 | PS/2 DATA port (*keyboard interface*) | 15 |
| **241 / F1** | Read/Write | PS/2 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 | HAG Low-Byte Register (0-255) | 17 |
| **247 / F7** | Write-only | HAG High-Byte Register (0-255) | 17 |

# Keyboard Interface

The uCOM GPU provides far more than just a video output terminal. The GPU card also provides a PS/2 keyboard terminal to fully release the uCOM from its dependency on a serial link to a host PC.

## PS/2 DATA PORT

This 8-bit data port returns the current MAKE or BREAK code (with bit 7 set). ASCII characters are returned as a result of a key being pressed on the PS/2 keyboard attached to the GPU card. Once read, the PS/2 DATA PORT is set to zero – any further reads will return 0x00 unless another MAKE or BREAK code is ready to be transmitted.

## PS/2 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 PS/2 STATUS register provides information on the current state of the PS/2 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

## PS/2 STATUS REGISTER – TYPEMATIC SETTINGS

Writing to the PS/2 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.

# Hardware-Accelerated Graphics engine

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

1. Pixel plotting
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. …hopefully more to come…

As well as the primary drawing functions, there are a number of 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 HAG has a 16-bit interface, driven by multiple IO writes from the Z80 in little-endian format – the LOW byte is always written first to IO port 246, then the HIGH byte – which causes the GPU to act on the data/command – is written to IO port 247. Data/commands should be written in 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 doesn’t leak garbage values into the code.

## HARDWARE COMMANDS

### X & Y registers

There are four 12-bit X and Y registers, X[0-3], Y[0-3], their contents set using the 16-bit input register. As with all data sent to the HAG, the low byte must be written first to port 246, followed by the high byte to port 247.

NOTE: These commands bytes (sent to the High-Byte Register) may also include data in their lower nybble.

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

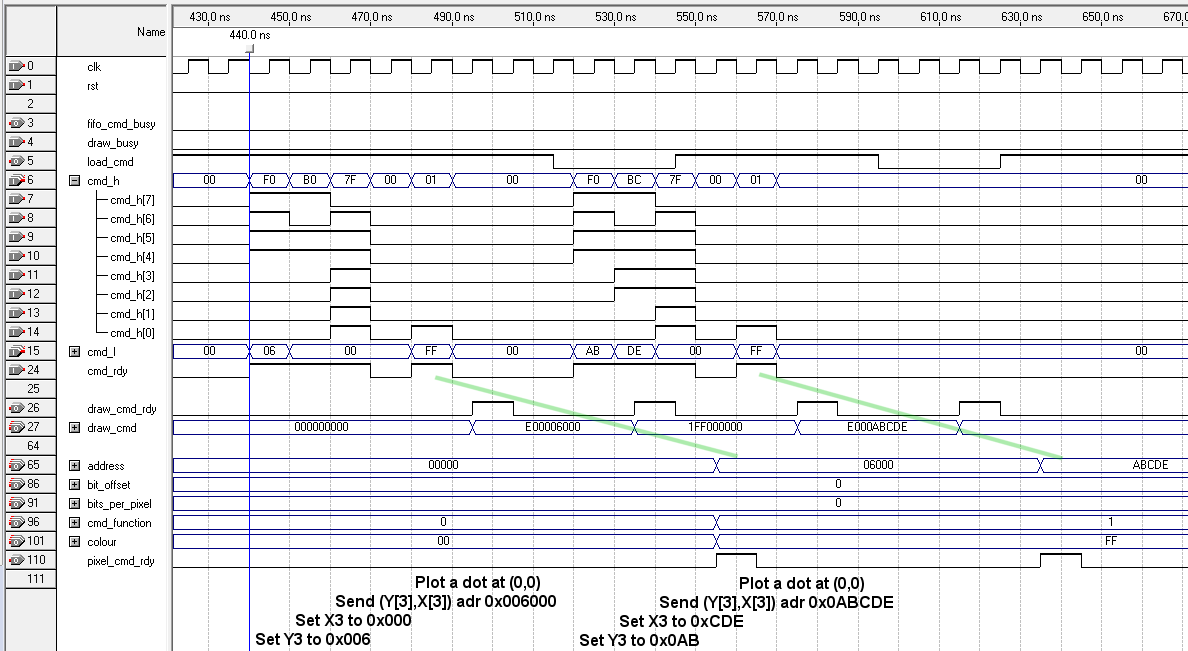
R - high-byte bit 6 (register bit 14)

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

nn – high-byte bits 5 & 4 (register bits 13 & 12)

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

? – data bits to be written to the specified register. Room for three nybbles (or three hex digits). Note that 4 bits in the HIGH BYTE can also be used to include data, like when setting a base memory address here.

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 (124-127)

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 – high-byte bits 1 & 0 (register bits 9 & 8)

These bits 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 (120-123)

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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 – high-byte bits 1 & 0 (register bits 9 & 8)

These bits specify which of the four X/Y registers to use the data from.

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

### Set destination raster bytes-per-horizontal line – X[2] (115)

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 | x | x | x |

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

### Set source raster bytes-per-horizontal line – Y[2] (114)

Sets the source raster’s bytes-per-horizontal line to the data in the Y[2] 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 | 1 | 0 | x | x | x | x | x | x | x | x |

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

### Set destination raster bytes-per-horizontal line – X[3] (113)

Sets the source raster’s bytes-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 | x | x | x |

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

### Set source raster bytes-per-horizontal line – Y[3] (112)

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 | x | x | x |

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

### Set max width & height of screen – X[0]/Y[0] (95)

Sets the bitmap dimensions to the data in the 0-index registers (X[0], Y[0]). 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 | 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] (94)

Sets the bitmap dimensions to the data in the 1-index registers (X[1], Y[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 | 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] (93)

Sets the bitmap dimensions to the data in the 0-index registers (X[0], Y[0]). 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 | 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] (92)

Sets the bitmap dimensions to the data in the 3-index registers (X[3], Y[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 | 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 (91)

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 (90)

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.



# HAG Design – To Do:

Current goal – create triangle feature.

Steps below outline key steps towards achieving this feature:

1. First, get a single line generator working and simulating in place of the original one.
2. Get the linegen to recognize that the beginning and ending coordinates are on the same point bypassing everything else and just drawing the dot.
3. Control the linegens to step/advance a Y-coordinate at a time with pre-sorted coordinates coming from the Z80.
4. Wiring three linegens together and control them with pre-sorted coordinates from the Z80.
5. Enable the fill feature.
6. Sort the coordinates to the 3 triangles in the geometry unit.
7. Make the linegens process all other shapes except ellipses.

# Appendix A – Character Codepage

