# Hardware Realtime Multi-Window Video Graphics Adapter

***Version 1.6, January 4, 2022.***

Supports multiple windows/layers.

With SDI-Sequentially interleave display layers to save on FPGA resources at the expense of dividing the output pixel clock and PDI-Parallel stacked display layers which allow full pixel speed clocks, but uses multiple M9K blocks and multipliers for layer mixing.

Written by Brian Guralnick.

For public use.

See: https:|www.eevblog.com/forum/fpga/brianhg\_ddr3\_controller-open-source-ddr3-controller/

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Supports 32/16a/16b/8/4/2/1 bpp windows.
* Supports accelerated Fonts/Tiles stored in dedicated M9K block RAM with resolutions of 4/8/16/32 X 4/8/16/32 pixels.
* Supports up to 16k characters with 32/16a/16b/8/4/2/1 bpp, with mirror and flip.
* Each window has a base address, X&Y screen position & H&V sizes up to 65kx65k pixels.
* Independent bpp depth for each window.
* Optional independent or shared 32bit colour palettes for each window.
* In tile mode, each tile/character's output with 8 bpp and below can be individually assigned to different portions of the palette.
* Multilayer 8-bit alpha stencil translucency between layers with programmable global override.
* Hardware individual integer X&Y scaling where each window output can be scaled 1x through 16x.

## Source files include:

* **BrianHG\_GFX\_Video\_Window\_System.sv** - Complete system wired together with direct Window Control Access port. Generates a VGA/HDMI compatible output with up to 64 window layers (DDR3 read speed permitting).
* **BrianHG\_GFX\_VGA\_Window\_System\_DDR3\_REGS.sv** - Same as the complete Window System, but all window controls can be accessed through written to DDR3 ram on the BrianHG\_DDR3 COM\_xxx multi-ports.
* **BrianHG\_GFX\_Sync\_Gen.sv and \_tb** - Generates a programmable video syncs and active picture region.
* **BrianHG\_GFX\_Video\_Line\_Buffer.sv and \_tb** - A dual-port video display line buffer which contains the Tile/Font/Palette memory with a line buffer which converts the source DDR3 reads on the CMD\_CLK to the output VID\_CLK domain. Supports up to 8 sequential interleaved layers.
* **BrianHG\_GFX\_Window\_DDR3\_Reader.sv and \_tb** - Used to send DDR3 read commands to fill the video display line buffer to construct the display.
* **BrianHG\_GFX\_Window\_Layer\_Mixer.sv** - Used to superimpose the windows on top of each other using the Alpha blend to mix the layers. Supports both 8 sequential and 8 parallel layers.
* **BrianHG\_GFX\_Window\_Collision\_Detector.sv** - Used in simple 2D gaming to detect active pixel collision between window layers on the display.

## Understanding DDR3 Bandwidth Limitations

*Note: This window system has no protection against over-flooding the available DDR3 bandwidth.*

Exceeding the available DDR3 bandwidth will generate horizontal zipper garbage on the screen.

A) Calculate available bandwidth:

B) Calculate required display bandwidth for each full screen window:

C) Calculate required display bandwidth for each full screen window:

Then determine percentage used of the available bandwidth:

**Example 1**: Deca running its 16-bit DDR3 at 400MHz:

A = 400MHz \* 16 \* 2 = 12800 mbps

Running a 480p display (27MHz pixel clock) with 2x 32bit windows, 1x 8bit window, and a 16-bit text window with the standard 8x16 VGA font:

Window 1 27MHz \* 32 = 864mbps

Window 2 27MHz \* 32 = 864mbps

Window 3 27MHz \* 8 = 216mbps

Window 4 27MHz \* 16 / 8 = 54mbps

----------------------------------------------------------------

Required bandwidth = 1998mbps

= ~16% of the DDR3 bandwidth

It is a good idea to keep this below 70%.

**Example 2:** Same setup for 720p, at 75Mhz pixel. i.e.: 720p@60hz and 1080p@30Hz:

Window 1 75MHz \* 32 = 2400mbps

Window 2 75MHz \* 32 = 2400mbps

Window 3 75MHz \* 8 = 600mbps

Window 4 75MHz \* 16 / 8 = 150mbps

----------------------------------------------------------------

Required bandwidth = 5550mbps

= ~44% of the DDR3 bandwidth.

## Preset Video modes and Window SDI\_LAYERs

Note that the current demo has a reference pixel clock of 148.5MHz.

A table of 8 possible video modes (MN#), optimized for 2 frequency groups, (27/54/108/216) and (74.25/148.5/297) to achieve all the main 16:9 standards except for 1280x1024. All modes target multiples of the standard 59.94Hz.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MN# | Mode | xCLK\_DIVIDER(-1) | Required VID\_CLK frequency | Byte |
| 0 | 480p | 1 | 27.00 MHz=60p or 54.00 MHz=120p or 108.00 MHz=240p |  |
| 0 | 480p | 2 | 54.00 MHz=60p or 108.00 MHz=120p or 216.00 MHz=240p |  |
| 0 | 480p | 4 | 108.00 MHz=60p or 216.00 MHz=120p |  |
| 0 | 480p | 8 | 216.00 MHz=60p |  |
| 1 | 720p | 1 | 74.25 MHz=60p or 148.50 MHz=120p or 297.00 MHz=240p |  |
| 1 | 720p | 2 | 148.50 MHz=60p or 297.00 MHz=120p | 0x11 |
| 1 | 720p | 4 | 297.00 MHz=60p |  |
| 2 | 1440x960 | 1 | 108.00 MHz=60p or 216.00 MHz=120p |  |
| 2 | 1440x960 | 2 | 216.00 MHz=60p |  |
| 3 | 1280x1024 | 1 | 108.00 MHz=60p or 216.00 MHz=120p |  |
| 3 | 1280x1024 | 2 | 216.00 MHz=60p |  |
| 4 | 1080p | 1 | 148.50 MHz=60p or 74.25 MHz=30p | 0x40 |
| 4 | 1080p | 2 | 297.00 MHz=60p or 148.50 MHz=30p |  |
| 4 | 1080p | 4 | too fast MHz=60p or 297.00 MHz=30p |  |
| 5 | … | … | Special mode/spare slot |  |
| 6 | … | … | Special mode/spare slot |  |
| 7 | 480p\* | 5 | 148.50 MHz=60p, x4=75Hz, x6=50hz | 0x74 |

*\* Special non-standard 480p operating on the 148.5MHz clock. If you want the 'OFFICIAL' standard 480p, then use mode #0 with a source clock of 27/54/108/216 MHz & properly set divider.*

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |
| localparam bit | [HC\_BITS-1:0] | VID\_h\_total | [0:7] = '{ | 858, | 1650, | 1716, | 1688, | 2200, | 858, | 858, | 940 | } ; |
| localparam bit | [HC\_BITS-1:0] | VID\_h\_res | [0:7] = '{ | 720, | 1280, | 1440, | 1280, | 1920, | 720, | 720, | 720 | } ; |
| localparam bit | [HC\_BITS-1:0] | VID\_hs\_front\_porch | [0:7] = '{ | 16, | 110, | 32, | 48, | 88, | 16, | 16, | 18 | } ; |
| localparam bit | [HC\_BITS-1:0] | VID\_hs\_size | [0:7] = '{ | 62, | 40, | 124, | 112, | 44, | 62, | 62, | 68 | } ; |
| localparam bit |  | VID\_hs\_polarity | [0:7] = '{ | 1, | 0, | 1, | 0, | 0, | 1, | 1, | 1 | } ; |
| localparam bit | [VC\_BITS-1:0] | VID\_v\_total | [0:7] = '{ | 525, | 750, | 1050, | 1067, | 1125, | 525, | 525, | 527 | } ; |
| localparam bit | [VC\_BITS-1:0] | VID\_v\_res | [0:7] = '{ | 480, | 720, | 960, | 1024, | 1080, | 480, | 480, | 480 | } ; |
| localparam bit | [VC\_BITS-1:0] | VID\_vs\_front\_porch | [0:7] = '{ | 6, | 5, | 6, | 2, | 4, | 6, | 6, | 6 | } ; |
| localparam bit | [VC\_BITS-1:0] | VID\_vs\_size | [0:7] = '{ | 6, | 5, | 6, | 3, | 5, | 6, | 6, | 6 | } ; |
| localparam bit |  | VID\_vs\_polarity | [0:7] = '{ | 1, | 0, | 1, | 0, | 0, | 1, | 1, | 1 | } ; |

Modes 0 through 7 can be selected in real time. You can also select the output clock divider in real time, a value from 0 through 7 which will divide the clock from 1x through 8x.

The current demo system runs on a fixed 148.5MHz clock. Use the table above to see what modes are possible. Byte values are provided which can be POKE’d to HW\_REG address offset +04.

When compiling the project with the parameter SDI\_LAYERS set to 4, this means that with the clock divider set to 4'h3, a divide value of 4, all of the 4 SDI\_LAYERS will function but, if you set the divider to 4'h1, divide clock by 2, only the first 2 SDI layers will function, but you now have double the pixel clock rate.

Same for a setting of 4'h0, divide clock by 1, only 1 SDI\_LAYER window will work. This however will not affect the total available PDI\_LAYERS and your total available windows will always be the functional SDI\_LAYERS multiplied by the PDI\_LAYERS.

Examples:

* Mode 3'h7 & CLK\_DIVIDER 4'h4 should give 480p at 60Hz with a maximum of 5 SDI\_LAYERS.
* Mode 3'h4 & CLK\_DIVIDER 4'h0 should give 1080p at 60Hz with a maximum of 1 SDI\_LAYERS.
* Mode 3'h4 & CLK\_DIVIDER 4'h1 should give 1080p at 30Hz with a maximum of 2 SDI\_LAYERS.
* Mode 3'h1 & CLK\_DIVIDER 4'h1 should give 720p at 60Hz with a maximum of 2 SDI\_LAYERS.

## List of Parameters

*Note that this HDL code was designed so that any disabled features and hard-wired window controls will prune unused logic and vastly cut required FPGA resources.*

HWREG\_BASE\_ADDRESS = 32'h00000100, | The first address where the HW REG controls are located for window layer 0

HWREG\_BASE\_ADDR\_LSWAP = 32'h000000F0, | The first address where the 16 byte control to swap the SDI & PDI layer order.

ENDIAN = "Little", | Enter "Little" or "Big". Used for selecting the Endian in tile mode when addressing 16k tiles.

OPTIMIZE\_TW\_FMAX = 1, | Adds a D-Latch buffer for writing to the tile memory when dealing with huge TILE mem sizes.

OPTIMIZE\_PW\_FMAX = 1, | Adds a D-Latch buffer for writing to the tile memory when dealing with huge palette mem sizes.

PDI\_LAYERS = 1, | No. parallel layered 'BrianHG\_GFX\_Video\_Line\_Buffer' modules in the system. 1-8 is allowed.

SDI\_LAYERS = 1, | Use 1/2/4/8 sequential display layers in each 'BrianHG\_GFX\_Video\_Line\_Buffer' module in the system.

ENABLE\_alpha\_adj = 1, | Use 0 to bypass the CMD\_win\_alpha\_override logic.

ENABLE\_SDI\_layer\_swap = 1, | Use 0 to bypass the serial layer swapping logic

ENABLE\_PDI\_layer\_swap = 1, | Use 0 to bypass the parallel layer swapping logic

LBUF\_BITS | The bit width of the CMD\_line\_buf\_wdata

LBUF\_WORDS | The total number of 'CMD\_line\_buf\_wdata' words of memory.

| Anything less than 256 will still use the same minimum M9K/M10K blocks.

| Only use factors of 2), IE: 256/512/1024...

ENABLE\_TILE\_MODE | Enable font/tile memory mode. This is for all SDI\_LAYERS.

SKIP\_TILE\_DELAY | If 1 and font/tile is disabled, the pipeline delay of the 'tile' engine will be skipped saving logic cells.

| However, if using multiple Video\_Line\_Buffer modules in parallel, some with and some without 'tiles' enabled,

| the video outputs of each Video\_Line\_Buffer will not be pixel accurate superimposed on top of each other.

TILE\_BASE\_ADDR | Tile memory base address.

TILE\_BITS | The bit width of the tile memory. 128bit X 256words (256-character 8x16 font), 1-bit colour. IE: 4kb.

TILE\_WORDS | The total number of tile memory words at 'TILE\_BITS' width.

| Anything less than 256 will still use the same minimum M9K/M10K blocks.

| Use a minimum of 256), maximum can go as far as the available FPGA memory.

| Note that screen memory is 32 bits per tile.

| Real-time software tile controls:

| Each tile can be 4/8/16/32 pixels wide and tall.

| Tile depth can be set to 1/2/4/8/16/32 bits per pixel.

| Each 32bit screen character mem =

| {8bit offset color), 8bit multiply color), 1bit v-flip), 1bit h-mirror), 14bit tile address}

| ---- For 2 thru 8 bit tiles/fonts ---- (multiply), then add rounding to 8 bits)

| Special for 1 bit tiles), the first byte is background and the next byte is foreground color.

| 16/32 bit tile modes are true-color.

| Palette is bypassed when operating in true-color modes.

ENABLE\_PALETTE | Enable a palette for 8/4/2/1 bit depth. Heavily recommended when using 'TILE\_MODE'.

SKIP\_PALETTE\_DELAY | When set to 1 and palette is disabled, the resulting delay timing will be the same as the

| 'SKIP\_TILE\_DELAY' parameter except for when with multiple ideo\_Line\_Buffer modules,

| some have the palette feature enabled and others have it disabled.

PAL\_BITS | Palette width.

PAL\_BASE\_ADDR | Palette base address.

PAL\_WORDS | The total number of palette memory words at 'PAL\_BITS' width.

| Having extra palette width allows for multiple palettes), each dedicated

| to their own SDI\_LAYER. Otherwise), all the SDI\_LAYERS will share

| the same palette.

PAL\_ADR\_SHIFT | Use 0 for off. If PAL\_BITS is made 32 and PORT\_CACHE\_BITS is truly 128bits, then use 2.

| \*\*\* Optionally make each 32 bit palette entry skip a x^2 number of bytes

| so that we can use a minimal single M9K block for a 32bit palette.

| Use 0 if you just want to write 32 bit data to a direct address from 0 to 255.

| \*\*\* This is a saving measure for those who want to use a single M9K block of ram

| for the palette), yet still interface with the BrianHG\_DDR3 'TAP\_xxx' port which

| may be 128 or 256 bits wide. The goal is to make the minimal single 256x32 M9K blockram

| and spread each write address to every 4th or 8th chunk of 128/256 bit 'TAP\_xxx' address space.

\*\*\* DDR3 controller related parameters:

PORT\_ADDR\_SIZE | Must match PORT\_ADDR\_SIZE.

PORT\_VECTOR\_SIZE | Must match PORT\_VECTOR\_SIZE and be at least large enough for the video line pointer + line buffer module ID.

PORT\_CACHE\_BITS | Must match PORT\_R/W\_DATA\_WIDTH and be PORT\_CACHE\_BITS wide for optimum speed.

## List of Window Controls.

CMD\_win\_enable [0:LAYERS-1], | Enables window layer.

CMD\_win\_bpp [0:LAYERS-1], | Bits per pixel. For 1,2,4,8,16a,32,16b bpp, use 0,1,2,3,4,5,6. \*16a bpp = 4444 RGBA, 16b bpp = 565 RGB.

CMD\_win\_base\_addr [0:LAYERS-1], | The beginning memory address for the window.

CMD\_win\_bitmap\_width [0:LAYERS-1], | The full width of the bitmap stored in memory. If tile mode is enabled, then the number of tiles wide.

CMD\_win\_bitmap\_x\_pos [0:LAYERS-1], | The beginning X pixel position inside the bitmap in memory.

CMD\_win\_bitmap\_y\_pos [0:LAYERS-1], | The beginning Y line position inside the bitmap in memory.

CMD\_win\_x\_offset [0:LAYERS-1], | The onscreen X position of the window.

CMD\_win\_y\_offset [0:LAYERS-1], | The onscreen Y position of the window.

CMD\_win\_x\_size [0:LAYERS-1], | The onscreen display width of the window. \*\*\* Using 0 will disable the window.

CMD\_win\_y\_size [0:LAYERS-1], | The onscreen display height of the window. \*\*\* Using 0 will disable the window.

CMD\_win\_scale\_width [0:LAYERS-1], | Pixel horizontal zoom width. For 1x,2x,3x thru 16x, use 0,1,2 thru 15.

CMD\_win\_scale\_height [0:LAYERS-1], | Pixel vertical zoom height. For 1x,2x,3x thru 16x, use 0,1,2 thru 15.

CMD\_win\_scale\_h\_begin [0:LAYERS-1], | Begin display part-way into a zoomed pixel for sub-pixel accurate scrolling.

CMD\_win\_scale\_v\_begin [0:LAYERS-1], | Begin display part-way into a zoomed pixel for sub-pixel accurate scrolling.

CMD\_win\_tile\_enable [0:LAYERS-1], | Enable Tile mode enable. \*\*\* Display will be corrupt if the BrianHG\_GFX\_Video\_Line\_Buffer

| module's ENABLE\_TILE\_MODE parameter isn't turned on.

CMD\_win\_tile\_bpp [0:LAYERS-1], | Defines the tile bits per pixel. For 1,2,4,8,16a,32,16b bpp, use 0,1,2,3,4,5,6. \*16a bpp = 4444 RGBA, 16b bpp = 565 RGB.

CMD\_win\_tile\_base [0:LAYERS-1], | Defines the beginning tile 16 bit base address (multiplied by) X 16 bytes for a maximum of 1 megabytes addressable tile set.

| \*\*\* This is the address inside the line buffer tile/font blockram which always begins at 0, NOT the DDR3 TAP\_xxx port write address.

CMD\_win\_tile\_width [0:LAYERS-1], | Defines the width of the tile. 0,1,2,3 = 4,8,16,32

CMD\_win\_tile\_height [0:LAYERS-1], | Defines the height of the tile. 0,1,2,3 = 4,8,16,32

CMD\_BGC\_RGB , | Bottom background color when every layer's pixel happens to be transparent.

CMD\_win\_alpha\_adj [0:LAYERS-1], | When 0, the layer translucency will be determined by the graphic data.

| Any figure from +1 to +127 will progressive force all the graphics opaque.

| Any figure from -1 to -128 will progressive force all the graphics transparent.

CMD\_SDI\_layer\_swap [0:PDI\_LAYERS-1], | Re-position the SDI layer order of each PDI layer line buffer's output stream. (A Horizontal SDI PHASE layer swap / PDI layer)

CMD\_PDI\_layer\_swap [0:SDI\_LAYERS-1], | Re-position the PDI layer order of each SDI layer. (A PDI Vertical swap/SDI Layer PHASE)

CMD\_win\_\*\*\*\* connection to HW\_REGS in the BrianHG\_GFX\_VGA\_Window\_System\_DDR3\_REGS.sv source file.

localparam int win\_len = 8'h20; | Length of bytes between each new window layer.

always\_comb begin | Also, don't forget everything is offset by the HWREG\_BASE\_ADDRESS parameter.

for (int x=0; x<LAYERS; x++) begin

CMD\_win\_base\_addr[x] = hw\_reg32[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h00]; The beginning DDR3 memory address for the window. Align to every 32 bytes for best DDR3 performance.

CMD\_win\_enable[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h04][7];| Enable window layer.

CMD\_win\_bpp[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h04][2:0];| Bits per pixel. Use (0,1,2,3,4,5,6) for (1,2,4,8,16a,32,16b) bpp, \*16a bpp=4444 RGBA, 16b bpp=565 RGB.

CMD\_win\_alpha\_adj[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h05];| 0=translucency will be determined by the graphic data, 127=100% opaque, -128=100% transparent.

CMD\_win\_bitmap\_width[x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h06]; | The full width of the bitmap stored in memory. If tile mode is enabled, then the number of tiles wide.

CMD\_win\_bitmap\_x\_pos[x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h08]; | The beginning X pixel position inside the bitmap in memory. IE: Scroll left on a huge bitmap.

CMD\_win\_bitmap\_y\_pos [x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h0A] | The beginning Y line position inside the bitmap in memory. IE: Scroll down on a huge bitmap.

CMD\_win\_x\_offset [x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h0C] | The onscreen X position of the window.

CMD\_win\_y\_offset [x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h0E] | The onscreen Y position of the window.

CMD\_win\_x\_size [x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h10] | The onscreen display width of the window.

\*\*\* Using 0 will disable the window.

CMD\_win\_y\_size [x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h12] | The onscreen display height of the window.

\*\*\* Using 0 will disable the window.

CMD\_win\_scale\_width[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h14] | Pixel horizontal zoom width. Use 0-15 for 1x through 16x size.

CMD\_win\_scale\_h\_begin[x]= hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h14] | Begin display part-way into zoomed pixel for sub-pixel accurate scrolling.

CMD\_win\_scale\_height[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h15] | Pixel vertical zoom height. Use 0-15 for 1x through 16x size.

CMD\_win\_scale\_v\_begin[x]= hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h15] | Begin display part-way into zoomed pixel for sub-pixel accurate scrolling.

CMD\_win\_tile\_base [x] = hw\_reg16[HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h16] | Base address x 16 bytes of where the windows font begins.

\*\*\*NOT counting the TILE\_BASE\_ADDR when writing into the DDR3.

CMD\_win\_tile\_enable[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h18][7]| Enable Tile mode enable.

\*\*\* Display will be corrupt if the BrianHG\_GFX\_Video\_Line\_Buffer's ENABLE\_TILE\_MODE=0

CMD\_win\_tile\_bpp [x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h18][2:0] | Defines the tile bits per pixel. For 1,2,4,8,16a,32,16b bpp, use

0,1,2,3,4,5,6. \*16a bpp = 4444 RGBA, 16b bpp = 565 RGB.

CMD\_win\_tile\_width [x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h19][5:4] | Defines the width of the tile. 0,1,2,3 = 4,8,16,32 pixels.

CMD\_win\_tile\_height[x] = hw\_reg8 [HWREG\_BASE\_ADDRESS+(x\*win\_len)+8'h19][1:0] | Defines the height of the tile. 0,1,2,3 = 4,8,16,32 pixels.

CMD\_BGC\_RGB[23:16] = hw\_reg8[HWREG\_BASE\_ADDRESS+16'h001A] | Global system 24-bit background colour for where no active window exists, or any

CMD\_BGC\_RGB[15: 8] = hw\_reg8[HWREG\_BASE\_ADDRESS+16'h001B] | pixels where all the layers are transparent all the way through the bottom layer.

CMD\_BGC\_RGB[ 7: 0] = hw\_reg8[HWREG\_BASE\_ADDRESS+16'h001C]

VIDEO\_MODE = hw\_reg8[HWREG\_BASE\_ADDRESS+16'h001F][6:4]; | 1 special address for changing the global VIDEO\_MODE.

CLK\_DIVIDER = hw\_reg8[HWREG\_BASE\_ADDRESS+16'h001F][2:0]; | 1 special address for changing the pixel CLK\_DIVIDER.

| \*\*\* Yes, the SDI & PDI swap positions are intentionally reversed as this is a grand crossbar 'X' swapper.

for (int x=0;x<PDI\_LAYERS;x++) CMD\_SDI\_layer\_swap[x] = hw\_reg8[HWREG\_BASE\_ADDR\_LSWAP+x+0][2:0];

for (int x=0;x<SDI\_LAYERS;x++) CMD\_PDI\_layer\_swap[x] = hw\_reg8[HWREG\_BASE\_ADDR\_LSWAP+x+8][2:0];

end | \_comb

|  |  |  |
| --- | --- | --- |
| GPU MEMORY MAP | | |
| ADDRESS | **FUNCTION** | **NOTES** |
| 00F0h | HWREG LSWAP Values | 16 bytes |
| 0100h | HWREG Base Address |  |
| 1000h | Layer 0 **P**alette **B**ase **A**ddress |  |
| 1400h | Layer 1 PBA |  |
| 1800h | Layer 2 PBA |  |
| … |  |  |
| 4000h | Tile Base Address | Start of font table |
| 5000h | Start of screen (free) RAM |  |

## HW\_REGS MAP

|  |  |  |
| --- | --- | --- |
| **HWREG\_BASE\_ADDRESS +** | | |
| **Off** | **Bits** | **Value** |
| **00** | 32 | **Base address** for screen memory (inc. Tile Mode). |
| **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 |

## Understanding the TILE/FONT enabled PDI layer.

The font/tile layer utilized 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.

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

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 colour, 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.

## 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.

IE, if the 'CLK\_DIVIDER' = 1, (needed for 720p@60hz, or 1080p@30hz) then only SDI layers 0 & 1 will function meaning only layers 0, 1, 4, 5, 8, 9, 12 & 13 are functional. That's 8 functional window layers.

IE, if the 'CLK\_DIVIDER' = 0, (needed for 1080p@60hz) then only SDI layer 0 will function meaning only layers 0, 4, 8, 12 are functional. That's 4 functional window layers.

Disable any unused layers as they will still eat DDR3 bandwidth, especially in 1080p@60hz as that mode eats a ton of DDR3 bandwidth. Even if the SDI layers aren’t functional, the controller will still try to do the DDR3 reading for those unseen SDI layers, but at full speed wasting bandwidth and potentially corrupting the display's line buffer.

The layer swapping controls are at address 0x00F0 through 0x00FF (**HWREG\_BASE\_ADDR\_LSWAP**). These controls allow you to quick swap which layers sit on top of each other allowing you to move which window appears in front of one another. This means you can also move a tile enabled layer in the 0, 1, 2 & 3 to a different position in the window stack.

Note that by default, layer 15 is the bottom window layer, layer 0 is the top window layer. Only the BGC colour setting is below layer 15 and it cannot be swapped anywhere above like all other windows 0 through 15.