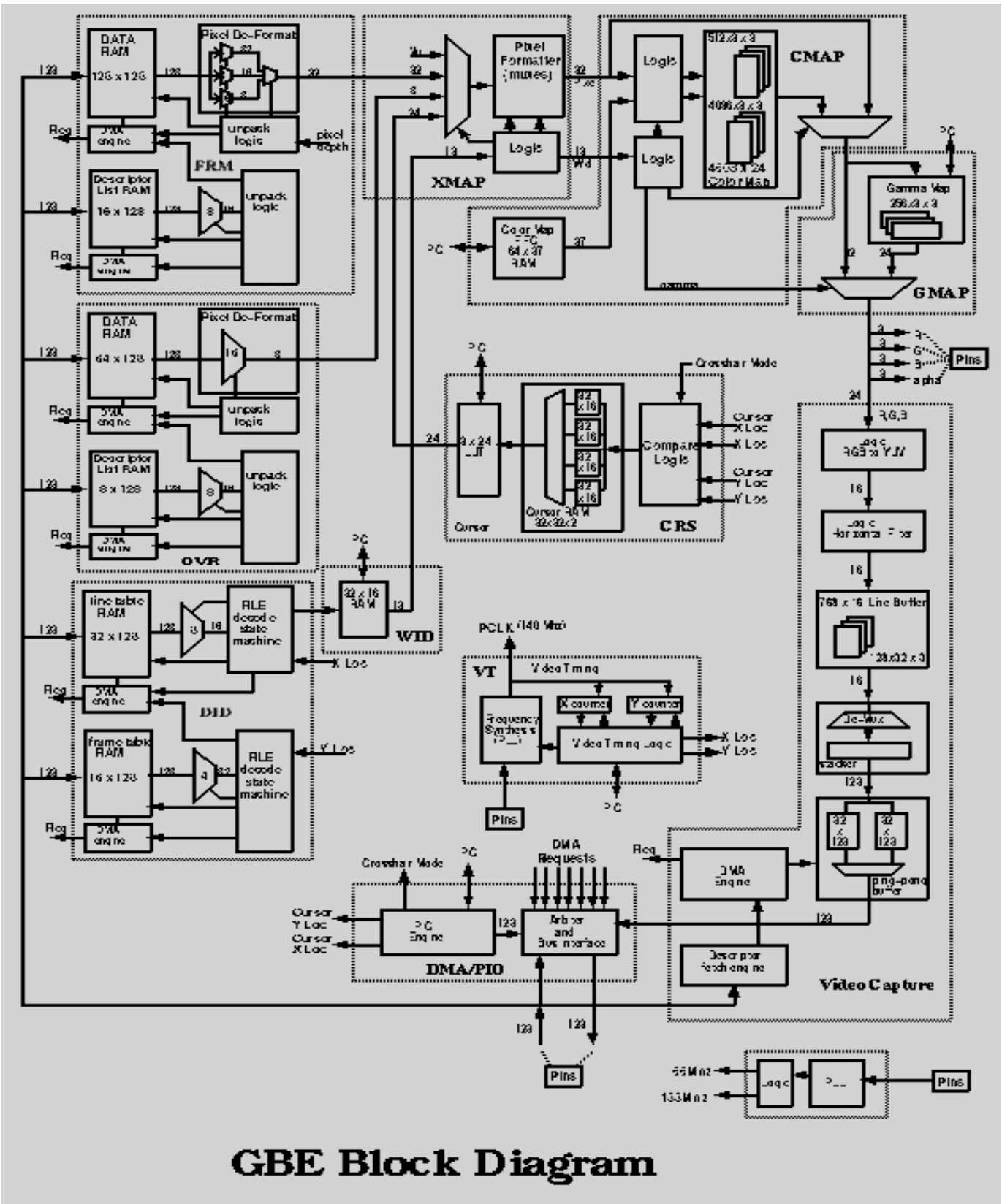
GBE ASIC spec for REV1.1

Rob Liston
Steve Ahlgrim
Mike Nielsen
Kamran Izadi

The Moosehead system architecture is centered around a unified main memory system which is capable of supplying 2.1 GB/s peak bandwidth. All devices in the system share this memory, including the CPU, CRIME, MACE, VICE, and GBE. The GBE chip is responsible for retrieving several data streams from main memory, formatting them according to SGI back end convention, and driving an external DAC at a dot clock of 140 MHz. The GBE chip contains a programmable video timing controller, hardware cursor, 4608 entry color map, and a 256 entry gamma map. A two-dimensional RLE encoded DID mechanism provides pixel by pixel control of the display mode. GBE manages the simultaneous fetching of overlay and normal bitplane streams. The overlay stream is 8 bit color indexed. The normal stream can be 16 or 32 bits deep, and can be split using the DIDs to provide 8+8 and 16+16 double buffering. These streams are mixed, along with the hardware cursor, in the XMAP stage under control of the DIDs.

GBE runs at 66 MHz and 140 Mhz, uses 0.5uM CMOS technology at 3.3V, and is housed in a 592_380 TBGA package. The die size is 573 $mils^2$.

- 16 or 32 bit normal planes
- 8 bit overlay planes
- multimode display
- 4608 entry color map
- 256 entry gamma map
- 32x32, 3 color cursor
- programmable video timing
- stereo goggle support
- flat panel support
- up to 140 MHz dot clock (supports 1280x1024 @ 76 Hz refresh)
- I^2C interfaces for DDC (Display Data Channel) support, flat panel control
- high resolution screen capture with flicker filter



GBE Block Diagram

1.0 Chip Interfaces

This section describes the external chip interfaces of GBE. There are three main ports: the CRIME interface, the DAC/flat panel interface, and the I^2C interface. The GBE pinout is as follows.

Pin name	I/O type	I/O cell	JTAG	Pin #	Description
pad_data[63:0]	3v TTL bidi, 4mA	PT3B02	yes		data bus to CRIME
pad_token_in	3v TTL input	PT3D01	yes		from CRIME
pad_token_out	3v tristate output, 4mA	PT3T02	yes		to CRIME
pad_ref67	3v TTL input	PT3D01	no		66.6 MHz reference clock
pad_ref133	3v TTL input	PT3D01	no		133.3 MHz reference clock
pad_reset_n	3v TTL input	PT3D01	yes		active low reset, internally synchronized
pad_pclk_in_pos	differential +	PG3D00	no		differential pclk in
pad_pclk_in_neg	differential -	PG3D01	no		differential pclk in
pad_pclk_in	3v TTL input	PT3D01	no		dot clock in from external PLL (optional)
pad_half_pclk_out	3v tristate output, 24mA	PT3T07	no		1/2 frequency dot clock to flat panel
pad_pclk_out	3v tristate output, 24mA	PT3T07	no		140 MHz dot clock to DAC
pad_red[7:0]	3v tristate output, 4mA	PT3B02	yes		digital red to DAC/flat panel (also pll test in)
pad_grn[7:0]	3v tristate output, 4mA	PT3B02	yes		digital green to DAC/flat panel (also pll test in)
pad_blu[7:0]	3v tristate output, 4mA	PT3B02	yes		digital blue to DAC/flat panel (also pll test in)
pad_alpha[7:0]	3v tristate output, 4mA	PT3B02	yes		alpha bits for future use
pad_did[1:0]	3v tristate output, 4mA	PT3T02	yes		did bits out for diagnostic purposes
pad_blank_n	3v tristate output, 4mA	PT3T02	yes		$\overline{\text{blank}}$ to DAC
pad_sync_n	3v tristate output, 4mA	PT3T02	yes		$\overline{\text{sync}}$ to DAC
pad_hdrv, pad_vdrv	3v tristate output, 4mA	PT3T02	yes		CRT timing signals
pad_fp_hdrv, pad_fp_vdrv, pad_fp_de	3v tristate output, 4mA	PT3T02	yes		flat panel timing signals
pad_frmlock	3v TTL input	PT3D01	yes		from I/O chip, locks vertical sync
pad_crt_scl_in, pad_crt_sdc_in	3v TTL input, schmitt	PT3D21	yes		crt i^2c clock,data input
pad_crt_scl_out, pad_crt_sdc_out	3v tristate output, 4mA	PT3T02	yes		crt i^2c clock,data output, external 4.7k pullup
pad_fp_scl_in, pad_fp_sdc_in	3v TTL input, schmitt	PT3D21	yes		flat panel i^2c clock,data input

Pin name	I/O type	I/O cell	JTAG	Pin #	Description
pad_fp_scl_out, pad_fp_sdc_out	3v tristate output, 4mA	PT3T02	yes		crt i^2c clock,data output, external 4.7k pullup
pad_f2rf	3v tristate output, 4mA	PT3T02	yes		stereo goggle control
pad_sense_n	3v TTL input	PT3D01	yes		monitor sense input from DAC
pad_aux[9:0]	3v TTL bidi, 4mA	PT3B02	yes		programmable auxiliary pins
pad_xin	xtal pad, low frequency	PT3D01	no		dot clock PLL crystal (20 MHz)
pad_spll_apwr, pad_spll_agnd, pad_spll_vccok	none	none	no		system clock PLL power
pad_spll_fb	3v tristate, 4mA	PT3T02	no		system pll feedback pin
pad_dpll_apwr,pad_dpll_agnd, pad_dpll_dpwr,pad_dpll_dgnd pad_dpll_pplus, pad_div2clr	none	none	no		dot clock PLL support
pad_spll_teste_n					system pll test enable input, active low
pad_spll_iddq					system pll reset, active high
pad_spll_bypass					system clock power down/bypass, active high
pad_dpll_teste					dot clock pll test enable, active high
pad_dpll_orclrn					dot clock reset, active low
pad_dpll_bypass					dot clock power down/bypass, active high
pad_bist_a10, pad_bist_a11			yes		bist address inputs for 4K CMAP RAMs
pad_trst_n	3v TTL input	PT3D01	no		JTAG reset, active low
pad_tdi	3v TTL input	PT3D01	no		JTAG test data in
pad_tms	3v TTL input	PT3D01	no		JTAG test mode select
pad_tck	3v TTL input	PT3D01	no		JTAG test clock
pad_tdo	3v tris. output, 4mA	PT3T02	no		JTAG test data out
pad_io_off_n	3v TTL input	PT3D01	no		tristate all outputs except for tdo, active low

1.1 CRIME interface

The CRIME to GBE interface is a point to point, burst oriented protocol with a peak bandwidth of 1 GB/s. The data path between chips consists of a 64 bit data bus toggling at 133 MHz. The interface is centrally clocked using a 66 MHz clock, with PLL cells in each chip to set the pin to core flip-flop delay to 0.0 ns. Care must be taken in the board level clock routing to ensure 0.0 ns clock skew at the pins of GBE and CRIME. The timing budget assumes minimum 1.0 ns and maximum 5.5 ns clock to out, plus chip and board level clock skew. Input path timing requires 0.0 ns hold time. The data bus input and output flip flops are clocked at 133 MHz.

The CRIME interface is based on the simple notion of a point to point link with one sender and one receiver. A full handshake is used to provide an orderly transition when swapping receiver and sender. There are no explicit flow control signals between the chips. All flow control is performed implicitly through the data lines. The 64 data lines are clocked at 133 MHz and are internally demultiplexed to a 128 bit wide bus running at 66 MHz, synchronous to ref67. Therefore, the

interface is logically 128 bits, and the smallest individual data transfer is 128 bits. Data transfers consist of a 128 bit header, optionally followed by some number of 128 bit data cycles. The header is formatted as follows.

cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

1.1.1 CRIME as sender commands

NOP					
0000	-	-	-	-	-
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

When either CRIME or GBE is the sender, it must drive NOP commands by default. CRIME is the sender on system reset, and must drive NOP commands during and after the reset period. This is to ensure that when GBE comes out of reset and begins receiving, it will be sampling known data.

PIO WRITE					
0001	-	-	-	**DATA**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

CRIME can write to GBE using the PIO write command. All writes are 32 data bits, and addresses are assumed to be 24 bits, and word-aligned (addr[1:0}= “00”). Address bits 31:24 are ignored. There is no flow control for PIO writes, so GBE must be able to accept writes at the full rate of 66 MHz. The only exception is color map writes, which go through a 64 deep fifo and must be throttled by software. The 24 bit address should be based at the start of the GBE address space.

PIO READ REQUEST					
0010	-	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

PIO reads from GBE are executed as a split transaction. To initiate the read, CRIME sends a PIO_READ_REQ command, which includes a 24 bit address field. GBE will respond with a PIO_READ_DATA command. There can only be one outstanding PIO read request. All reads are 32 bits, and the 24 bit read address is 32-bit aligned and based at the start of the GBE address space.

DMA READ DATA					
0011	**COUNT**	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

CRIME uses the DMA_READ_DATA command to send DMA data to GBE. The header contains a 5 bit COUNT field, which specifies the number of 256 bit data words which immediately follow the header. For example, if COUNT= “00002”, then the 128 bit header will be followed by 4 cycles of 128 bit data. DMA transfers are always aligned on 256 bit

boundaries, and are always multiples of 256 bits. The address is returned with the data for debugging purposes; CRIME always returns DMA data in the same order that it was requested by GBE.

DMA WRITE DONE					
0100	**-**	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

CRIME sends a DMA_WR_DONE command to GBE to indicate that a DMA write has been flushed to main memory. CRIME has one DMA write buffer, and can therefore only handle one outstanding DMA write. GBE must wait for a DMA_WR_DONE message from CRIME before issuing a new DMA write.

1.1.2 GBE as sender commands

PIO READ DATA					
0101	**-**	-	-	**DATA**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE sends a PIO_READ_DATA command to complete the PIO read transaction. GBE always returns 32 bits of data in the data field. The read address is returned for debugging purposes.

DMA READ REQUEST					
0110	**COUNT**	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE sends a DMA_READ_REQ command to request a block of main memory. DMA reads are executed as split transactions, and CRIME contains a memory transaction queue which can hold up to 16 requests. As a practical matter, it is essential for GBE to keep the request queue non-empty, so that the memory controller inside CRIME will operate at full bandwidth. The ADDR field is 256 bit aligned, so bits [4:0] must be “00000”. The COUNT field specifies the number of 256 bit words to transfer. CRIME guarantees that it will return DMA_READ_DATA blocks in the same order that they were received.

DMA WRITE DATA					
0111	**COUNT**	-	-	**-**	**ADDR**
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE uses the DMA_WRITE_DATA header, followed by COUNT*2 data cycles, to transfer DMA write data from GBE to one of the two DMA write buffers inside CRIME. The DMA will also be queued in the CRIME memory transac-

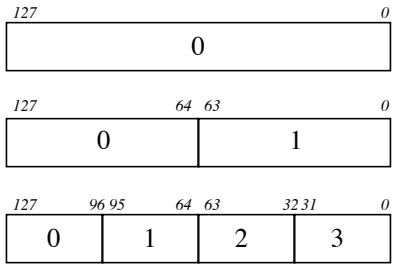
tion queue, to be executed in turn. The COUNT refers to 256 bit words, and the address is 256 bit aligned. GBE must ensure that no more than two DMA writes are pending at any time.

INTERRUPT					
1000	-	-	-	-	MASK
cmd[3:0]	size[4:0]	unused[22:0]	reserved[31:0]	data[31:0]	addr[31:0]
127:124	123:119	118:96	95:64	63:32	31:0

GBE sends an interrupt command to indicate the vertical retrace interrupt. Note that this is a one-shot event, so the interrupt must be latched inside CRIME. The lower 4 bits of the address field (MASK) identify the source of the interrupt within GBE. If a MASK bit is '1', then the corresponding interrupt is set inside CRIME. If the MASK bit is '0', there is no change to the interrupt register in CRIME. CRIME is only capable of clearing the interrupt.

To summarize the important assumptions:

- only 1 outstanding PIO read allowed
- only 1 outstanding DMA write allowed
- only 8 outstanding DMA reads/writes allowed
- no limits on PIO writes
- all PIO is 32 bits data, 24 bits address, 32 bit aligned, GBE start address => 0x000000
- all DMA is 512 byte aligned, multiples of 256 bits
- sender must drive NOP commands by default, also during reset
- data is always sent in big endian order

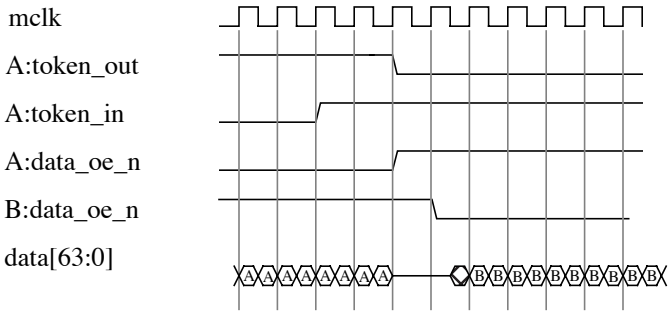


- no 8K DRAM page crossings

1.1.3 Transfer of mastership

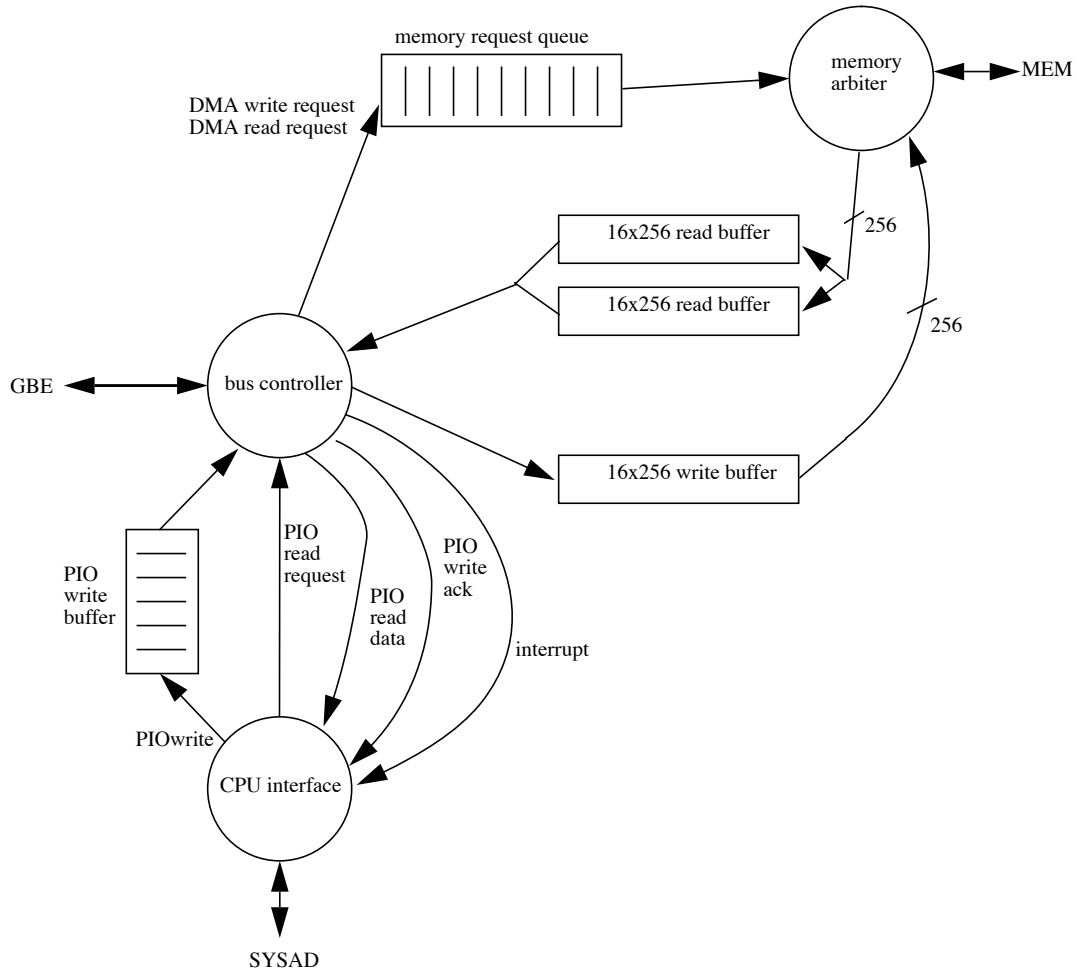
The command/data structure describes the mechanism for transferring data. To complete the protocol we need to know how the two devices swap roles as sender and receiver. This is accomplished using the token_in and token_out signals. On the board, CRIME's token_out should be wired to GBE's token in, and GBE's token_out should be wired to CRIME's token_in. On system reset, CRIME is always the sender, which it indicates by driving token_out high and driving NOP commands. On reset, GBE is idle and will drive its token_out low. This state will continue until, through PIO programming, GBE needs to become the sender. GBE will then drive its token_out high, and wait for its token_in to go low. As soon as GBE detects that token_in is low, it should begin driving the data bus. At this point, GBE is the sender and CRIME is the receiver. The sequence is identical to switch mastership back to CRIME.

The following timing diagram shows the transition from Device A sending to Device B sending.



Note that the current sender should not release the bus until it has completed sending all of its pending commands. This will help to minimize the total number of bus turn arounds.

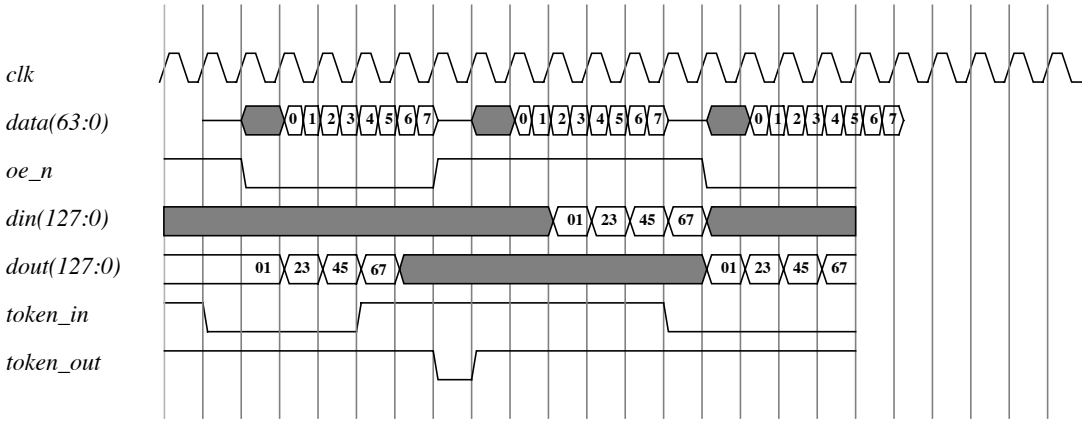
1.1.4 CRIME side of GBE interface, block diagram

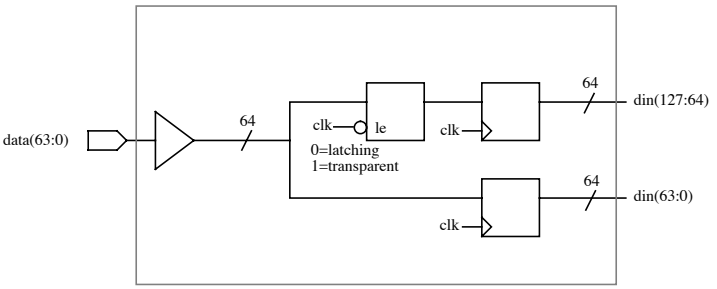


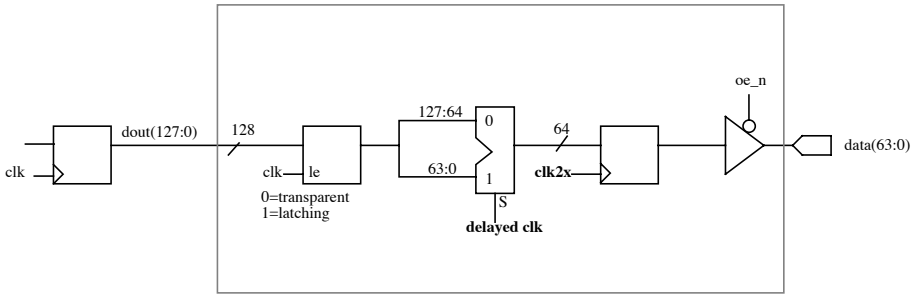
Note that for DMA writes, the memory request is issued after the last data has been written into the 16x256 write buffer. On DMA reads, the data may be transferred as soon as it arrives from the memory arbiter, since the memory data rate is twice the GBE bus interface data rate.

1.1.5 Clocking scheme

The following logical circuits are used to clock data in and out of GBE.







1.2 DAC / flat panel interface

GBE contains a programmable dot clock PLL, which generates a pixel clock up to 140 MHz. This clock drives the pixel pipeline and the video timing controller. The pixel clock is output from GBE on the pclk_out pin. The interface to the DAC consists of pclk_out, red[7:0], grn[7:0], blu[7:0], blank_n and sync_n. In addition, hdrv and vdrv provide sepa-

rate monitor sync signals. An input status bit is provided for the sense_n output of the DAC. This allows software to detect whether the monitor is plugged in. The pixel clock can be driven externally or from the internal pixel PLL

The flat panel display requires a horizontal and vertical sync signal which is active for the entire blanking period. In order to simultaneously drive the CRT and flat panel displays, GBE produces fp_hdrv and fp_vdrv specifically for the flat panel. In addition, an fp_de (flat panel display enable) signal is produced, which functions as a flat panel blanking signal but with slightly different timing than the CRT blank_n.

1.3 I^2C interface

GBE provides an I^2C interface in order to support the DDC (Display Data Channel). DDC is a VESA standard which allows two way communication between the computer and the monitor. This will allow software to read the monitor timing information and automatically program the video timing controller inside GBE. In addition, DDC will allow software access to additional monitor functionality. GBE provides separate input and output pins for the open drain SDA and SCL lines so that external high power drivers can be used.

A separate I^2C interface is also used to control the flat panel display.

1.4 Miscellaneous signals

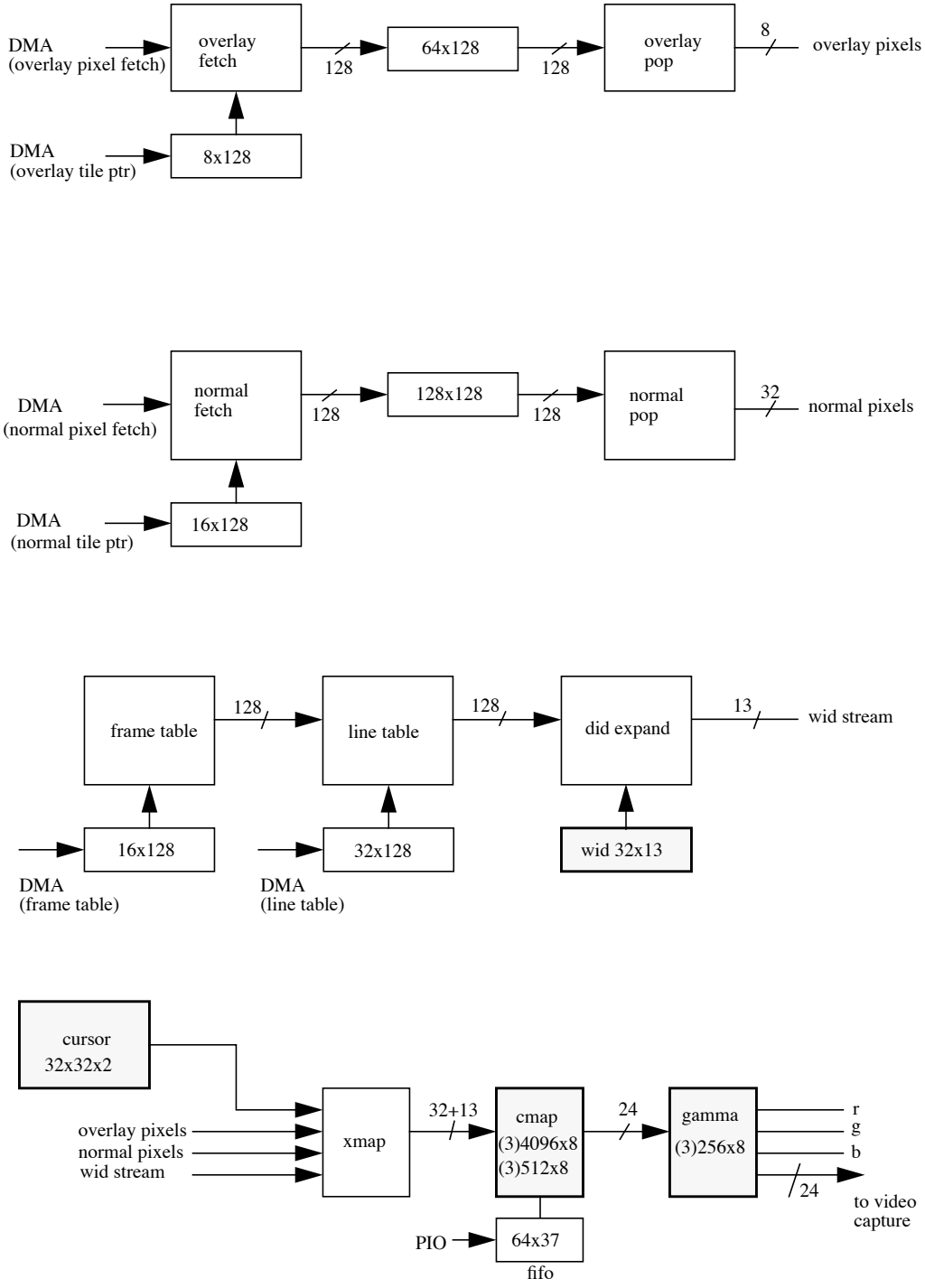
GBE provides 10 general purpose auxiliary pins for unforeseen uses. These pins are programmed using the CTRL-STAT register, and can be individually tristated.

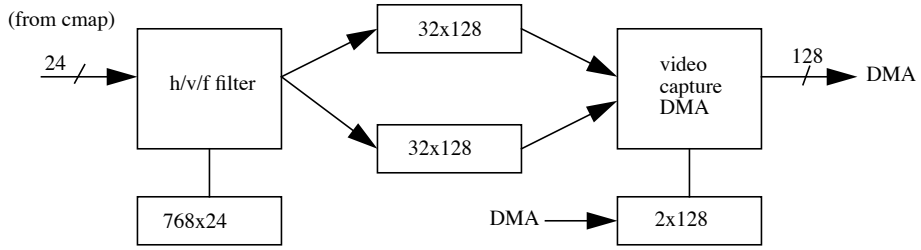
2.0 Functional Description

The primary function of GBE is to fetch several pixel streams from main memory, format and blend them according to the DID stream, and output them to the CRT in real time. The internal block diagram for GBE reflects these functions. There are eight major functional blocks:

- DID pipeline. This logic fetches the DID frame and line tables and converts them into a real time DID stream.
- Normal planes pixel fetch. This block fetches the normal planes pixel stream (16 or 32 bits deep).
- Overlay planes pixel fetch. This block fetches the 8 bit overlay planes.
- Cursor. 32x32, 3 color hardware cursor.
- XMAP-CMAP-GAMMA. Using the DID stream as a control, blend the overlay, normal, and cursor pixels.
- Video timing. Fully programmable, handles stereo goggles.
- DMA/bus control. Handles CRIME interface, internal registers.
- Video capture. Optionally filters the high res output, sends screen pixels back to main memory.

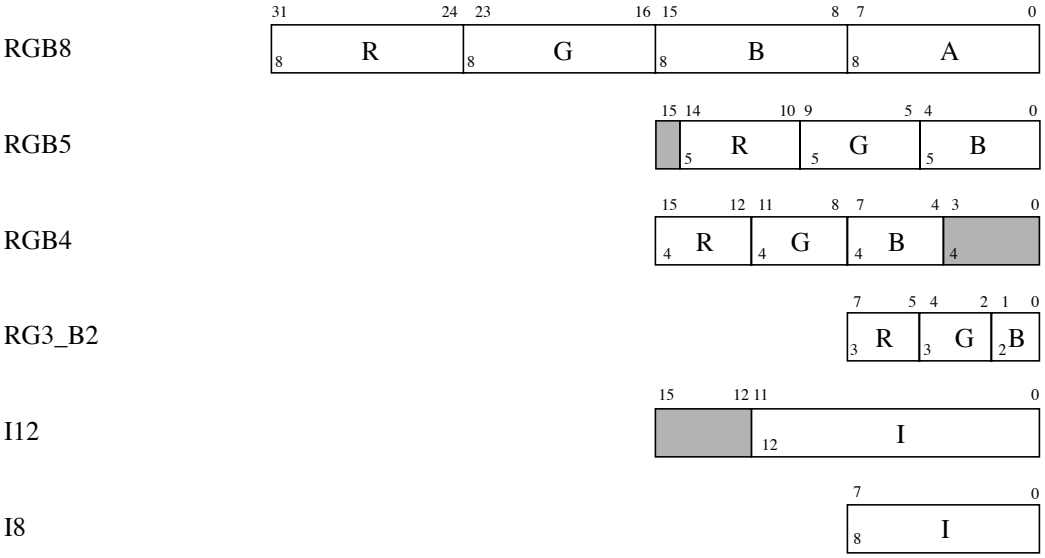
The following block diagram shows the major blocks inside GBE. High speed RAMs are shaded.





2.1 Pixel formats

Moosehead pixels are 8, 16 or 32 bits deep. The following formats are supported in GBE.



2.2 Tile formats

In order to increase the amount of 2D spatial locality for screen rendering, Moosehead stores screen pixels in tiled format. Tiles can be 8, 16, or 32 bits per pixel, and are always 64K bytes in size and are aligned on 64K byte boundaries. The tiles are organized as 128 lines by 512 bytes. The pixel width of a tile is 512 divided by the pixel depth. That is, 8 bit pixels are tiled 512x128, 16 bit pixels are tiled 256x128, and 32 bit pixels are tiled 128x128. The frame buffers are built out of an integral number of tiles, possibly with unused pixels on the right and bottom edges. For example, a

1280x1024x32 frame buffer exactly occupies 10x8 tiles. Pixels appear in big endian format within the 256 bit memory width, where the least significant word corresponds to the leftmost screen pixel.

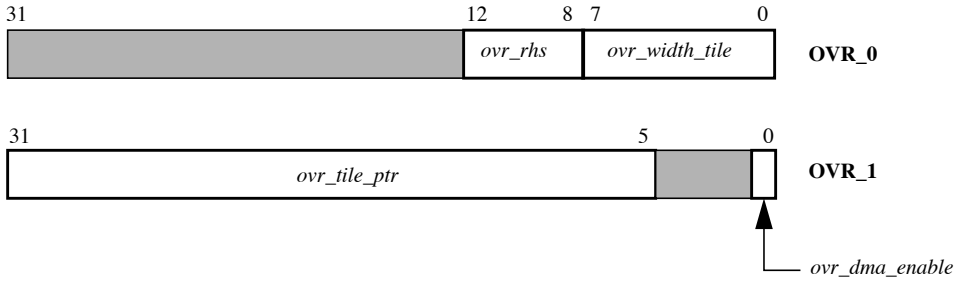
2.3 Overlay planes

The overlay planes are 8 bits per pixel deep and operate in color index mode. An overlay pixel with value 0x00 will be transparent. The overlay pixels always index into the last 256 entry window in the 4608 entry color map.

The overlay frame buffer is composed of 64K byte tiles which are aligned on 64K byte boundaries. The tiles do not have to be contiguous in physical memory. GBE reads a list of tile pointers to determine the addresses of the overlay tiles. The tile pointer list entries are 16 bits each, which correspond to the upper 16 bits of the physical tile address. The tile pointer list must start on a 32 byte boundary, and must not cross an 8K DRAM page boundary. The tile pointers are ordered from top to bottom, left to right. In other words, the first entry in the tile pointer list is the upper-left tile, followed by the rest of the tiles in scan order. If a tile pointer entry is 0x0000, then GBE will not issue any DMA transactions for that tile, producing 0x00 value pixels instead. If possible, software should detect the case of an entirely transparent overlay tile, and set those tile pointers to 0x0000. This will minimize the impact of overlay DMA on the rest of the system.

Note that the overlay and normal planes have the same width and height in pixels, and the same height in tiles. The width in tiles may be different due to the variable pixel width of the normal planes tiles. The height in pixels is specified using the *fb_height_pix* field in the **FRM_1** register. In order to simplify the hardware, the width is specified by the number of integral tiles and the width of the right hand side tile in 32 byte units. These are specified in the *ovr_width_tile* and *ovr_rhs* registers. This means that that the width of the screen in pixels must be evenly divisible by 32. If the screen is tiled into an integral number of pixels, then *ovr_rhs* should be “00000”.

GBE does not support display ID bits for the overlay planes. The 8 bits of overlay color index always point to entries 4352 to 4607 in the color map. Overlay pixels always pass through the gamma map. If the overlay pixel is 0x00, then the overlay is transparent and the normal planes are displayed instead. Overlay can be completely disabled by setting *ovr_dma_enable* to ‘0’, in which case no memory DMA transactions will be issued and zero value pixels will be generated. If possible, software should detect the case when the entire overlay frame buffer is zero, and completely disable overlay DMA. This will help lower the bandwidth consumption of GBE. The following registers control the operation of the overlay planes.

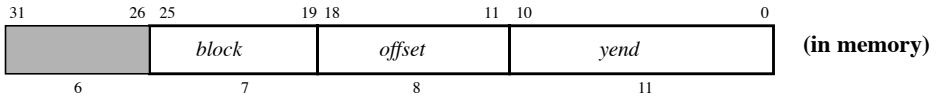


The pixel buffering for the overlay planes consists of a single 64x128 dual port RAM. This RAM is internally partitioned into two 512 byte buffers, which are used in ping pong fashion to store incoming DMA pixels. The resulting pixel stream is then sent to the XMAP stage. GBE reads the entire overlay tile pointer list into a 4x128 RAM at the beginning of vertical blanking.

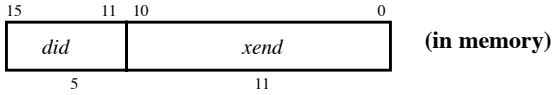
2.4 DID table

SGI graphics hardware traditionally supports a multimode display in which each pixel can have a different display mode. This is useful for multiple double buffered windows, and different color maps for different windows. GBE interprets an RLE encoded DID table, expanding it into a real time stream of DIDs. The DID stream is then used to format the raw pixel stream in the XMAP block. The DID table resides in main memory, and is transferred to GBE buffers as needed using DMA.

The DID table consists of a frame table and a set of line tables referenced by the frame table. Each frame table entry describes a vertical run of lines that have the same line table. Each line table describes a horizontal run of pixels that have the same DID. The frame table is an array of 32 bit entries as shown below:



Each line table is an array of 16 bit entries of the form shown below:

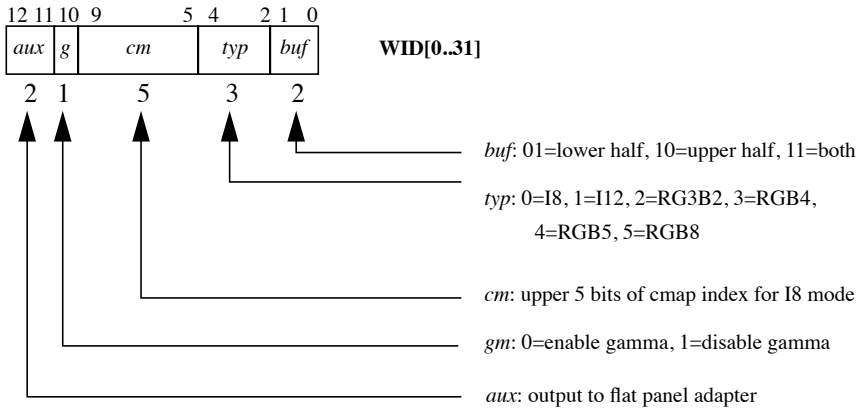


The frame table and line tables must be packed into one 64k byte DID table, which is also aligned on a 64k boundary. The base address of the DID table is specified in the *did_base* register, which specifies the upper 16 bits of the physical address of the DID table. If the *did_dma_enable* bit is set to '0', then GBE will not perform any DID DMA, and will produce "00000" value DIDs instead.

The *yend* field of a frame table entry specifies the last Y coordinate of the vertical line run, where *yend*=0 corresponds to the first active line of a frame. If at the end of the current line table, the *yend* field does not match, the line table is reexecuted; otherwise, the frame table advances to the next frame table entry. The *block* field specifies the 512 byte block number of the display table that contains the line table. The *offset* field specifies the 16 bit offset into the block at which the line table starts. That is, the physical byte address of the line table start *did_base* + 512**block* + 2**offset*. A line table must not cross a 512 byte boundary. This limits a line table to a maximum of 256 entries. As many line tables as possible should be packed into a single 512 byte block, without violating the boundary. The frame table may reference a line table multiple times.

The *xend* field of a line table entry specifies the last X coordinate of the horizontal pixel run, where *xend*=0 corresponds to the first active pixel of a line. When the *xend* field matches, the line table advances to the next entry. The last entry in the line table should have an *xend* value of 2047.

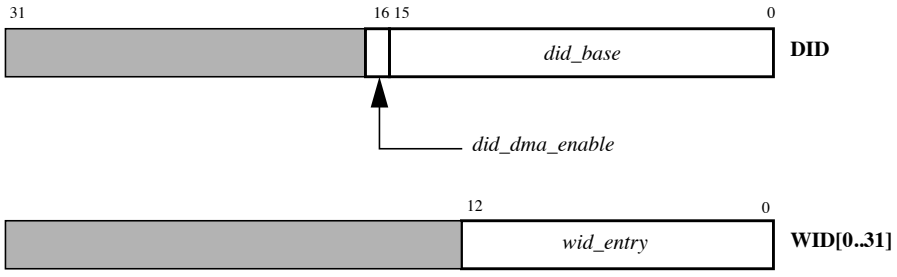
The *did* field of a line table entry indexes a 32 entry window table to obtain the actual display mode bits. This level of indirection allows change of window mode such as *swapbuffers()* without updating the frame and line tables. The window table is a 32x13 RAM inside GBE. The 32 window table entries are organized as follows.



As an implementation detail, the logic which processes the line table entries requires some care. The entire line table is fetched into a 512 byte registered RAM, organized as 16x128. Therefore in each 128 bit location, 8 line table entries are simultaneously available out of the RAM. However, because the RAM is registered there is a two clock pipeline delay from the time the new read address switches to the time the new data switches. The DID logic must prefetch the next 128 bits of line table in advance, so that an uninterrupted stream of real time DIDs is produced.

The frame table RAM is only 256 bytes, organized as 16x128. The size is reduced because of the very small bandwidth requirements of frame table fetch.

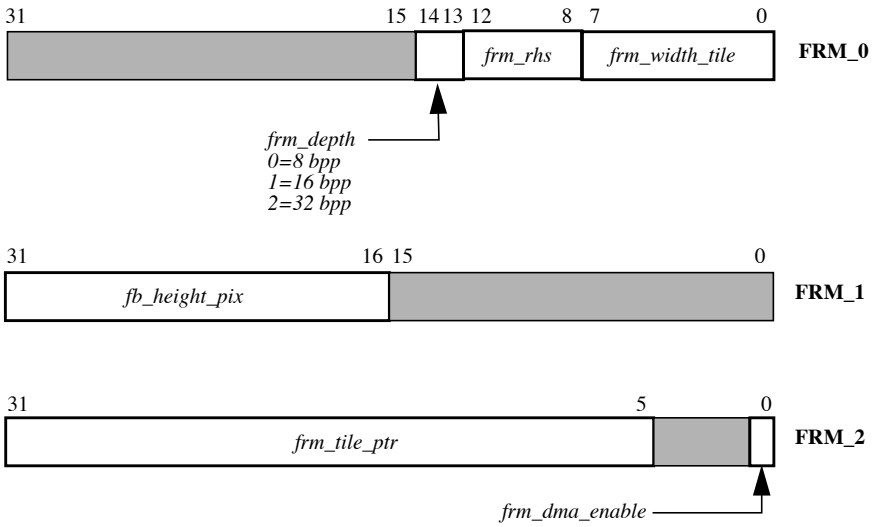
The following registers control the operation of the DID mechanism inside GBE.



2.5 Normal planes

The normal planes fetching logic behaves nearly identically to the overlay fetch. The normal planes are composed of 64K tiles, which are addressed by a tile pointer table to remove the requirement that the tiles be contiguous in physical

memory. The normal planes can be 8, 16, or 32 bits deep, as specified by the *frm_depth* register. Like the overlay planes,



normal plane DMA can be completely disabled by setting *frm_dma_enable*, to '0'. This should be done whenever the entire screen is blanked, e.g. during power saving modes. GBE reads the tile pointer list for the entire frame at the beginning of vertical blanking. Note that the *fb_height_pix* field is also used by the overlay planes logic.

2.6 XMAP

The XMAP block is responsible for selecting between the cursor, overlay, and normal pixels. If the cursor pixel is opaque, then it is selected. Otherwise, if the overlay pixel is non-zero, then it is selected. If the cursor and overlay are both transparent, then the normal planes are selected. The XMAP block also uses the DID stream to demultiplex the front or back buffers from a double buffered 16 or 32 bit pixel.

2.7 Color map

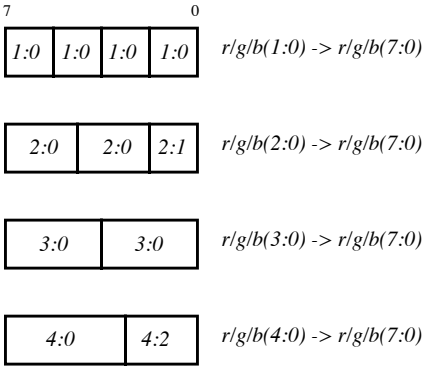
The color map takes the pixel stream from the XMAP block and performs any necessary color lookup to produce a 24 bit RGB8 pixel stream. It is organized as three 4608x8 maps which are indexed together for I8 and I12 modes, or on a component basis for RGB modes to support X direct color visuals.

RG3B2, RGB4, and RGB5 pixels are first expanded to RGB8 by bit replication. Diagram 1 shows how 2, 3, 4, and 5 bit components are expanded to 8 bits.

If the *cm* field of the DID is zero, the resulting RGB8 pixels are output to the gamma stage. If *cm* is nonzero, then the RGB8 pixels are used to index the color map using the following algorithm. The *cm* field is concatenated with each of the R, G, B components to produce 3 13 bit indexes into the color map. The *cm* field provides the upper 5 bits, and the 8 bit color index is the low order bits. The RGB components are independently mapped through the color map and are then output to the gamma stage. Mapping RGB values on a component basis is needed to support the direct color visual in X-windows.

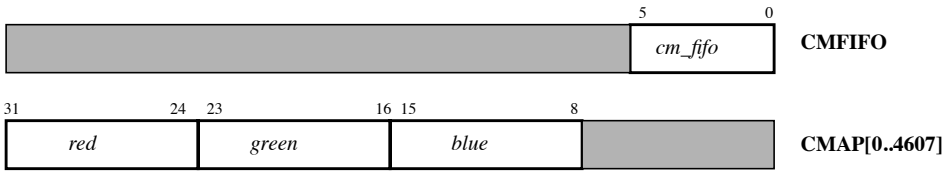
I12 pixels are passed through the color map, using locations 0..4095. I8 pixels are concatenated with the DID *cm* field to produce a 13 bit color index : *cm*(4 downto 0) & i8(7 downto 0).

Diagram 1: RGB color expansion



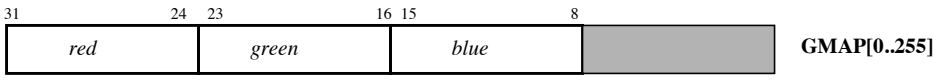
The color map is a single port RAM, and so reading or writing it during the active video time will cause visible artifacts. Reading of the color map is considered a diagnostic operation. Writes to the color map go through a 64 deep FIFO, which is used to maximize the throughput of color map updates. The 64 deep color map FIFO is drained during horizontal and vertical blanking, and is guaranteed not to cause artifacts. Software can read the number of free entries in the FIFO at any time by reading the *cm_fifo* register. Note that the actual number of free entries may be greater than the number returned by reading *cm_fifo*, but will never be less. Software must ensure that no more than 64 color map writes are pending in the FIFO at any time. It can do this by writing 64 values to an empty FIFO, and then polling until a low water mark is detected.

The following registers are used to control the color map.



2.8 Gamma map

The gamma map is organized as three 256x8 maps. Cursor pixels always bypass the gamma map. Overlay pixels are always gamma corrected. Other pixels are gamma corrected if the *gm* field of the DID is ‘0’. The following registers are used to control the gamma map.



2.9 Video timing

The video timing block is responsible for generating all of the pixel synchronous timing signals in GBE. Interlaced video timings are not supported. The video out circuit in the MACE chip should be used to output interlaced video from Moosehead.

All video timing signals are derived from a pair of 12 bit counters, *vt_x* and *vt_y*. *vt_x* counts from zero to *vt_maxx* at the dot clock frequency, and then resets to zero. *vt_y* count counts from zero to *vt_maxy*, incrementing whenever *vt_x==-vt_maxx*, and then resets to zero.

The external video timing signals are hdrv, vdrv, sync_n, and blank_n. These signals are not used internally by GBE and may be programmed arbitrarily. The vertical sync pulse is turned on when *vt_y==vt_vsync_on*, and is turned off when *vt_y==vt_vsync_off*. The horizontal sync is turned on when *vt_x==vt_hsync_on*, and turned off when *vt_x==vt_hsync_off*. The resulting vsync and hsync signals are optionally inverted by setting *vt_vdrv_invert* or *vt_hdrv_invert*, and output as vdrv and hdrv. The hdrv and vdrv signals can be forced high or low by setting *vt_vdrv_low* or *vt_hdrv_low* and optionally inverting (for DPMS signaling).They are also combined into a composite, active low sync_n for use by the DAC. For monitors which cannot handle sync on green and separate syncs simultaneously, the composite sync_n can also be forced high or low by setting the *vt_sync_high* or *vt_sync_low* bits. The internal horizontal blank signal is turned on when *vt_x==vt_hblank_on*, and turned off when *vt_x==vt_hblank_off*. The internal vertical blank signal is turned on when *vt_y==vt_vblank_on*, and turned off when *vt_y==vt_vblank_off*. Horizontal blank and vertical blank are logically OR'ed, and the composite active low blank is output on the blank_n pin. The following VHDL code partially describes the circuit which generates vdrv.

```
process begin
   wait until clk140'event and clk140='1';
   if vt_y=vt_maxy then
      vt_y <= "000000000000";
   else
      vt_y <= vt_y + "000000000001";
   end if;

   if vt_vdrv_low='1' then
      vdrv <= vt_vdrv_invert;
   elsif vt_y=vt_vsync_on then
      vdrv <= not vt_vdrv_invert;
   elsif vt_y=vt_vsync_off then
      vdrv <= vt_vdrv_invert;
   end if;
end process;
```

GBE supports stereo goggles through the f2rf output. Stereo works by running the monitor at 120 Hz refresh and alternating left and right eye images on each frame. The stereo goggles use LCD shutters to alternately opaque each eye. The f2rf output controls the left/right opacity of the goggles. To enable stereo, the video timing should be set to produce a 120 Hz frame rate. Software should create two sets of normal and overlay planes; one for the left eye and one for the right eye. Once per frame, software will reprogram the *ovr_tile_ptr* and *frm_tile_ptr* registers to alternate between left and right eye. Note that GBE reads the entire tile pointer list at the beginning of vertical blanking, so the tile pointers should be reprogrammed during the active period with the intent that they take effect the next frame. The f2rf output toggles when *vt_y==vt_f2rf*, and can be forced high by writing a one to *vt_f2rf_high*.

In order to support external frame lock, the framelock input pin causes the *vt_y* counter to preset to *vt_frmlock*. The framelock pin is driven by the MACE chip, and is derived from an external video source. Note that this is not genlock; only the frame to frame period is locked, not each individual pixel. A specific requirement is that if the GBE frame rate (as

determined by the video timing) and the external frame rate are not exactly the same, the GBE image should not “jump” or lose pixels. This could happen if the framelock input is asserted at the end of the frame when active pixels are being displayed. To prevent this, the *vt_frmlock* register should be set to a line which is advanced slightly in the vertical blanking period. Then, if the frequencies do not match exactly, the high res screen might lose or gain some vertical blanking time, but will maintain a stable image. The framelock input is internally synchronized to the GBE dot clock, and the rising edge of the synchronized signal is used to preset *vt_y* at the next *vt_x==vt_maxx*.

GBE is specifically designed to simultaneously drive a CRT and a flat panel. Because the flat panel requires slightly different sync signals, GBE produces fp_hdrv, fp_vdrv, and fp_de independently of the CRT sync signals. fp_hdrv is produced analogously to the hdrv signal, and is controlled by the *fp_hdrv_on* and *fp_hdrv_off* registers. fp_vdrv is produced analogously to the vdrv, and is controlled by *fp_vdrv_on* and *fp_vdrv_off*. fp_de functions much like the CRT blank_n signal, and is controlled using *fp_de_on* and *fp_de_off*.

Four programmable vertical interrupts are provided. Whenever *vt_intr_0*, *vt_intr_1*, *vt_intr_2* or *vt_intr_3* match *vt_y*, then the corresponding interrupt is generated. The interrupt is latched inside the CRIME chip, and must be cleared by writing to CRIME. The host can also read the value of the *vt_y* register at any time to determine the location of the CRT beam. These interrupts may be reprogrammed in real time to generate an arbitrary number of interrupt points per frame.

2.9.1 GBE internal timing signals

The video timing block is also responsible for generating the various internal timing information for the GBE pipelines. The normal and overlay plane fetch logic requires a “pixel_enable” signal to enable the flow of pixels into the XMAP stage. The horizontal component of this signal is controlled using the *vt_hpixen_on* and *vt_hpixen_off* registers, which are compared against the *vt_x* register. The vertical pixel enable is controlled via the *vt_vpixen_on* and *vt_vpixen_off* registers, which are compared with the *vt_y* register. The horizontal and vertical are logically OR’ed to produce the desired pixel enable signal.

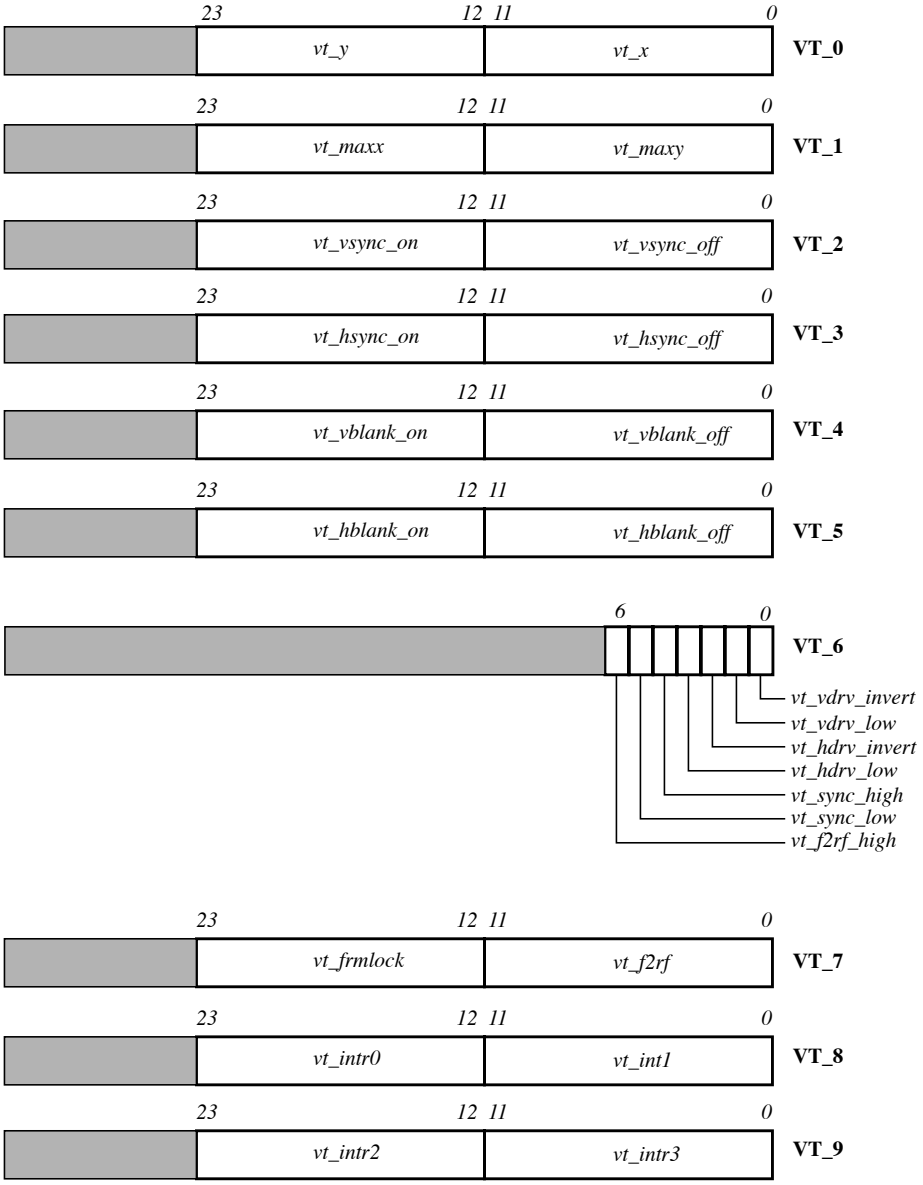
The color map write logic requires a write enable input which is guaranteed to fall within a synchronization boundary of the blanking period. This signal is generated in exactly the same way as pixel_enable. The following registers control the “cmap_wr_enable” signal: *vt_hcmap_on*, *vt_hcmap_off*, *vt_vcmap_on*, *vt_vcmap_off*.

The DID block requires a version of the *vt_x* and *vt_y* counters to compare against the xend and yend fields of the DID tables. These are 12 bit counters and are called did_x and did_y. The did_x counter increments at the dot clock rate, and presets to the value *did_startx* when *vt_x=vt_maxx*. Similarly, the did_y counter increments when *vt_y* increments, and presets to *did_starty* when *vt_y* resets to zero. *did_startx* and *did_starty* should be programmed so that the xend and yend fields of the DID tables correspond tot he active region of the raster, so that xend=0 refers to the left edge and yend=0 refers to the top edge. Note that the preset values can be made negative using two’s complement numbers. The did_x and did_y counters leave the video timing block from a flip flop, and are immediately clocked into another flip flop upon reaching the DID block.

The cursor block requires a similar X and Y counter as the DID block. These counters are preset to the values in the *crs_startx* and *crs_starty* registers. These preset values should be adjusted so that the crs_posx and crs_posy registers are properly referenced so that the lower right pixel of the cursor glyph corresponds to the upper left corner of the active raster.

The video capture block needs an X and Y counter like the DID and cursor blocks. These counters are preset to the values in the *vc_startx* and *vc_starty* fields in the **VC_19** register. These preset values should be set so that the left, right, top and bottom fields of the video capture registers refer to the active region of the raster.

The following registers are used to control the external video timing signals in GBE.



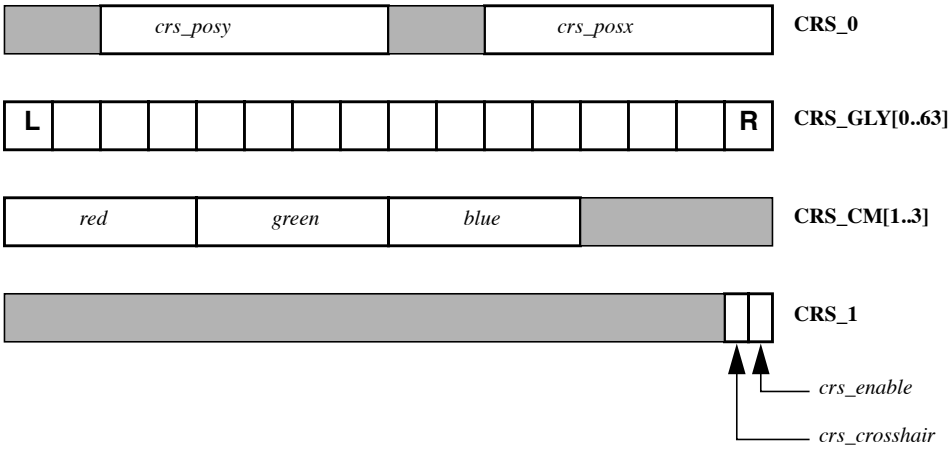
2.10 Cursor

The cursor block generates a 32x32x2 cursor with the upper left pixel screen position specified by the 12 bit *crs_posx* and *crs_posy* registers. The cursor coordinates are offset by (31, 31) to allow a cursor at (0, 0) to only have the lower right pixel visible at the upper left corner of the screen. The cursor glyph is stored in a RAM accessed by the 64 *crs_glyph* registers. Each 32 bit register specifies 16 packed 2-bit cursor pixels of the glyph. The cursor glyph pixels are ordered from

top to bottom, left to right; i.e. crs_glyph[0] and crs_glyph[1] correspond to the left and right 16 pixels of the top row, respectively. The cursor RGB colors are specified via the three *crs_cmap* registers. A cursor glyph pixel with value 0 is transparent.

Alternatively, the cursor may be configured as a crosshair cursor, in which case the *crs_pos* register indicates the position of the crosshair center. The crosshair is set to cursor color index 1 and the cursor glyph is ignored.

The cursor is enabled and the glyph/crosshair mode is set via the *crs_ctl* register. The cursor should be disabled during access to the *crs_glyph* registers. The following registers are used to control the cursor.

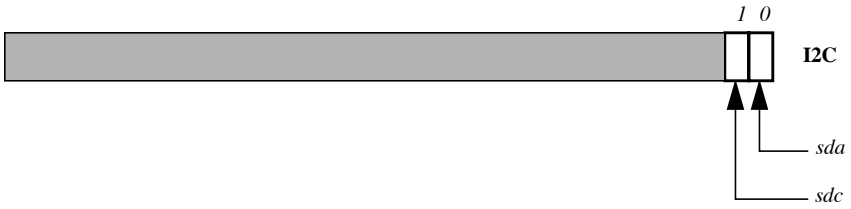


2.11 I²C controller

The I^2C controller is used to read and write the monitor and flat panel display. GBE makes the simplifying assumption that it is the only I^2C master on the bus. This minimizes the number of gates while preserving DDC2 compatibility. The I^2C controller relies on direct software control of the sdc and sda pins in order to send and receive data. The hardware interface consists of two output pins and two input pins. The output pins are connected to external open collector drivers, such as an ‘F06. Software can write the sdc and sda registers at any time. The input pins are Schmitt triggered and should be connected to the I^2C bus. The sdc_in and sda_in inputs are synchronized to the 66 MHz memory clock and can be read at any time by software. Note that writing the sda/sdc registers affects the sda_out and sdc_out pins, while reading the sda/sdc register returns the actual value of the I^2C bus, which in general is not the same as the state of sda_out/sdc_out. This interface is sufficient for very low bandwidth communication such as monitor adjustment and monitor timing information.

The DDC2 specification is a VESA creation which allows a host computer to read and write the monitor via an I^2C interface. DDC2 can be used to query monitor timing information, gamma characteristic, manufacturer information, screen size, power saving modes, and other information.

The following register is used to support the I^2C interface:



2.12 Video capture

GBE has the capability of sending the 140 MHz pixel stream back into main memory, where it can be sent to the video output port in MACE. This feature is referred to as GBE video capture, and consists of a filtering stage and a DMA controller stage.

2.12.1 Video capture filter stage

Because of memory bandwidth limitations, the image data which is written back to main memory must be limited to NTSC/PAL data rates. This means that the 1280x1024 high resolution display must be filtered down to either 768x288 for interlaced square pixel PAL or 640x240 for interlaced square pixel NTSC. Optionally, the user may select no filtering, in which case the capture region must correspond 1:1 with the target television resolution. Note that GBE will always output square pixels; square to non-square conversion is performed in the MACE chip as part of the video out hardware.

The video capture filter can be partitioned into three discrete filtering blocks - the horizontal filter, the vertical filter, and the anti-flicker filter. Each of these filter blocks can be engaged separately.

2.12.2 Horizontal averaging

The horizontal filter has two modes. It can filter 1280 input pixels down to 640 output pixels for NTSC, or filter 1280 input pixels down to 768 output pixels for PAL. In the case of 1:1 capture this filter can be disabled entirely. The NTSC horizontal filter is a three tap transversal filter with weights 1/4, 1/2, 1/4. This filter is shown in the following table.

Horizontal averaging for NTSC

input	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
weight	1/2	1/4		1/4	1/2	1/4		1/4	1/2	1/4
		1/4	1/2	1/4		1/4	1/2	1/4		
output	N1		N2		N3		N4		N5	

The PAL case is somewhat more complicated, requiring a reduction of 3/5 instead of 1/2. The following table describes the relationship between input pixels and output pixels in the PAL case.

Horizontal averaging for PAL

input	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
weight	*2/3*	*1/3*		*1/3*	*2/3*	*2/3*	*1/3*		*1/3*	*2/3*
		1/4	1/2	1/4			1/4	1/2	1/4	
output		*N1*	N2		*N3*		*N4*	N5		*N6*

As in the NTSC case weights are applied to input pixels then accumulated until an output pixel results. The odd weights must be approximated to greatest practical extent using shifts and adds or table lookup.

2.12.3 Vertical and flicker filtering

Vertical filtering is applied after horizontal filtering. Conceptually the vertical filter function can be thought of as an averaging filter followed by an anti- flicker filter. The averaging filter performs the conversion from full screen, 960 lines, down to either PAL (576) or NTSC (480) vertical resolution. Note that GBE outputs 1280x960 in order to maintain a 4:3 aspect ratio. The flicker filter is used to smear adjacent high resolution lines between fields to reduce 30 Hz flicker. If the anti-flicker filter is disabled then every other line is dropped without filtering. In either case an interlaced television image is obtained from a progressively scanned high resolution display. Although the user views these vertical filters as two separate blocks they are collapsed using a single line delay employed as an accumulator with use of different weights applied to effect the following modes:

- High resolution, 960 lines filtered down to 576 lines (PAL) without flicker reduction
- High resolution, 960 lines filtered down to 576 lines (PAL) with flicker reduction
- High resolution, 960 lines filtered down to 480 lines (NTSC) without flicker reduction
- High resolution, 960 lines filtered down to 480 lines (NTSC) with flicker reduction
- PAL video resolution (1:1) 576 lines out, without flicker reduction
- PAL video resolution (1:1) 576 lines out, with flicker reduction
- NTSC video resolution (1:1) 486 lines out, without flicker reduction
- NTSC video resolution (1:1) 486 lines out, with flicker reduction

As in the horizontal case the vertical averaging filter must reduce the full screen image by a factor of 1/2 for NTSC and 3/5 for PAL. The following table shows the relationship of input lines to output lines in the vertical averaging case. Note that in this case as well as the other vertical filtering cases a given input line only contributes to filtered lines at boundaries between output pixels. This eliminates the need for a second vertical delay element.

Vertical averaging for NTSC and PAL

input line	**NTSC**				**PAL**		
	coef	coef	output		coef	coef	output
L0	1/4				*2/3*		
L1	1/2		N1		*1/3*	1/4	*N1*
L2	1/4	1/4				1/2	N3
L3		1/2	N3		*1/3*	1/4	
L4	1/4	1/4			*2/3*		*N5*
L5	1/2		N5		*2/3*		
L6	1/4	1/4			*1/3*	1/4	*N7*
L7		1/2	N7			1/2	N9
L8	1/4	1/4			*1/3*	1/4	
L9	1/2				*2/3*		*N11*

The flicker filter is common to both NTSC and PAL since in both cases it performs 2:1 averaging. The next table shows the relationship between input lines and the output of the flicker filter. Note the spatial relationship between even fields and odd fields. The flicker filter will produce an odd or even field under software control and is determined ultimately by video output timing.

NTSC/PAL flicker filter

input									
	flicker filter on **odd field**			**even field**				**flicker filter off** **odd field**	**even field**
	coef	coef	output	coef	coef	output		output	output
L0	1/4				1/2	E0			E0
L1	1/2		O1	1/4	1/4			O1	
L2	1/4	1/4		1/2		E2			E2
L3		1/2	O3	1/4	1/4			O3	
L4	1/4	1/4			1/2	E4			E4
L5	1/2		O5	1/4	1/4			O5	
L6	1/4	1/4		1/2		E6			E6

Finally, the vertical filter along with the flicker filter are collapsed into a single vertical filter with a single line delay element. This memory element is 768x24. The NTSC and PAL cases are shown in the following figures.

input line	[vertical][flicker]																	
	[1][1]						[1][0]						[0][1]					[0][0]
	odd			even			odd			even			odd			even		odd/even
	coef		out	coef		out	coef		out	coef		out	coef		out	coef		output
L0	1/4				1/4		1/4	1/4		1/4	1/4		1/4			1/2		**E0**	**E0**
L1	1/8	1/8			1/4	**E0**		1/2		1/2		**E0**	1/2		**O1**	1/4	1/4		**O1**
L2		1/4			1/4		1/4	1/4		1/4	1/4		1/4	1/4			1/2	**E2**	**E2**
L3		1/4	**O1**	1/8	1/8		1/2		**O1**		1/2			1/2	**O3**	1/4	1/4		**O3**
L4		1/4		1/4			1/4	1/4		1/4	1/4		1/4	1/4		1/2		**E4**	**E4**
L5	1/8	1/8		1/4		**E2**		1/2		1/2		**E2**	1/2		**O5**	1/4	1/4		**O5**
L6	1/4			1/4			1/4	1/4		1/4	1/4		1/4	1/4			1/2	**E6**	**E6**
L7	1/4		**O3**	1/8	1/8		1/2		**O3**		1/2			1/2	**O7**	1/4	1/4		**O7**
L8	1/4				1/4		1/4	1/4		1/4	1/4		1/4	1/4		1/2		**E8**	**E8**

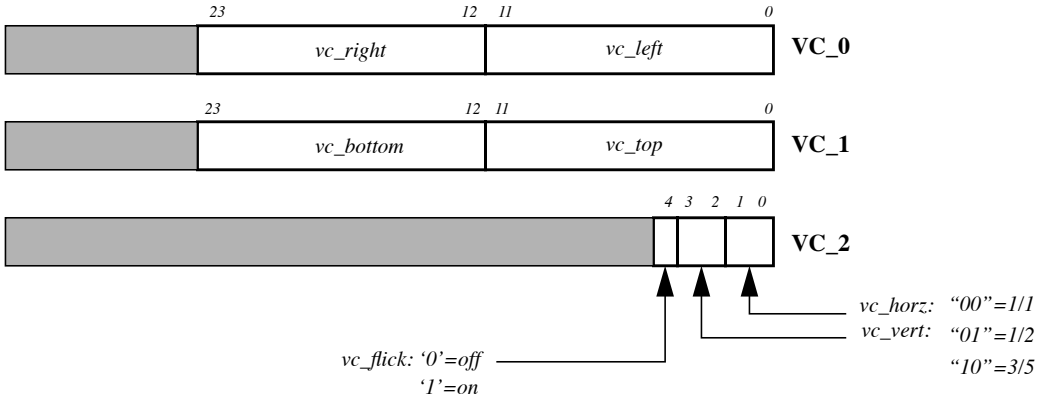
Vertical filter combined with flicker filter, NTSC

input	[vertical][flicker]																	
	[1][1]						[1][0]						[0][1]					[0][0]
	even			odd			even			odd			even			odd		even/odd
	coef		out	coef		out	coef		out	coef		out	coef		out	coef		out	output
L0	1/3			1/3		O1	1/4			1/3	1/4	O1	1/4			1/2		O1	O1
L1	7/24			1/6	1/8		1/2		E2		1/2		1/2		E2	1/4	1/4		E2
L2	1/4		E2		1/4		1/4	1/3		1/3	1/4		1/4	1/4			1/2	O3	O3
L3	1/8	1/6			7/24	O3		2/3		2/3		O3		1/2	E4	1/4	1/4		E4
L4		1/3			1/3		2/3				2/3		1/4	1/4		1/2		O5	O5
L5		1/3	E4	1/3			1/3	1/4	E4	1/4	1/3		1/2		E6	1/4	1/4		E6
L6	1/8	1/6		7/24		O5		1/2		1/2		O5	1/4	1/4			1/2	O7	O7
L7	1/4			1/4			1/3	1/4		1/4	1/3			1/2	E8	1/4	1/4		E8
L8	7/24		E6	1/8	1/6		2/3		E6		2/3		1/4	1/4		1/2		O9	O9
L9	1/3				1/3	O7		2/3		2/3			1/2		E10	1/4	1/4		E10
L10		1/3			1/3		1/4	1/3		1/3	1/4	O7	1/4	1/4			1/2	O11	O11
L11		7/24	E8	1/8	1/6		1/2		E8		1/2			1/2	E12	1/4	1/4		E12

Vertical filter combined with flicker filter, PAL

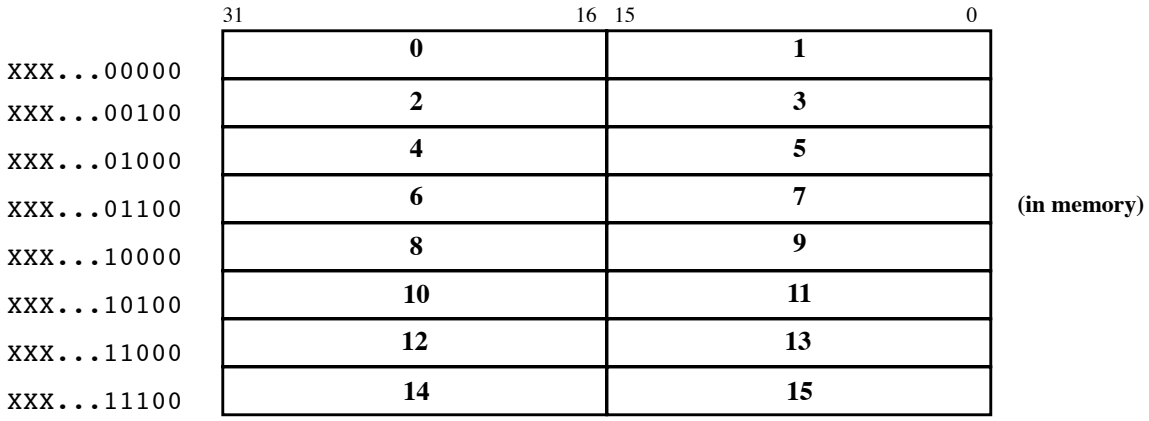
2.12.4 Filter programming interface

Software selects the area of the screen to capture by specifying an inclusive rectangle. The left, right, top, and bottom registers specify this rectangle in terms of the active area of the screen. This area should enclose the pixels to be captured before filtering. The filter mode is selected using the VC_2 register.



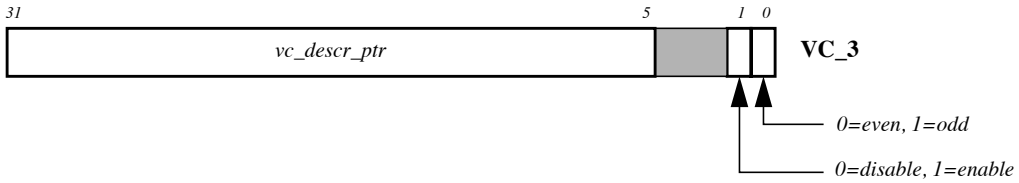
2.12.5 Video capture DMA stage

After filtering, the NTSC/PAL size field is written to main memory using DMA. The field is written as a continuous stream of pixels in 64K byte blocks, as opposed to the tiled format of the frame buffers. The maximum size field is a PAL resolution of 768x288. At 32 bits per pixel, this requires 884736 bytes, or 13.5 64K byte blocks. The video capture DMA controller reads a small descriptor list to determine the addresses of the 64K blocks to write pixels into. The descriptor list is 32 bytes long and must be aligned on a 32 byte boundary in main memory. The descriptor list has the following format:

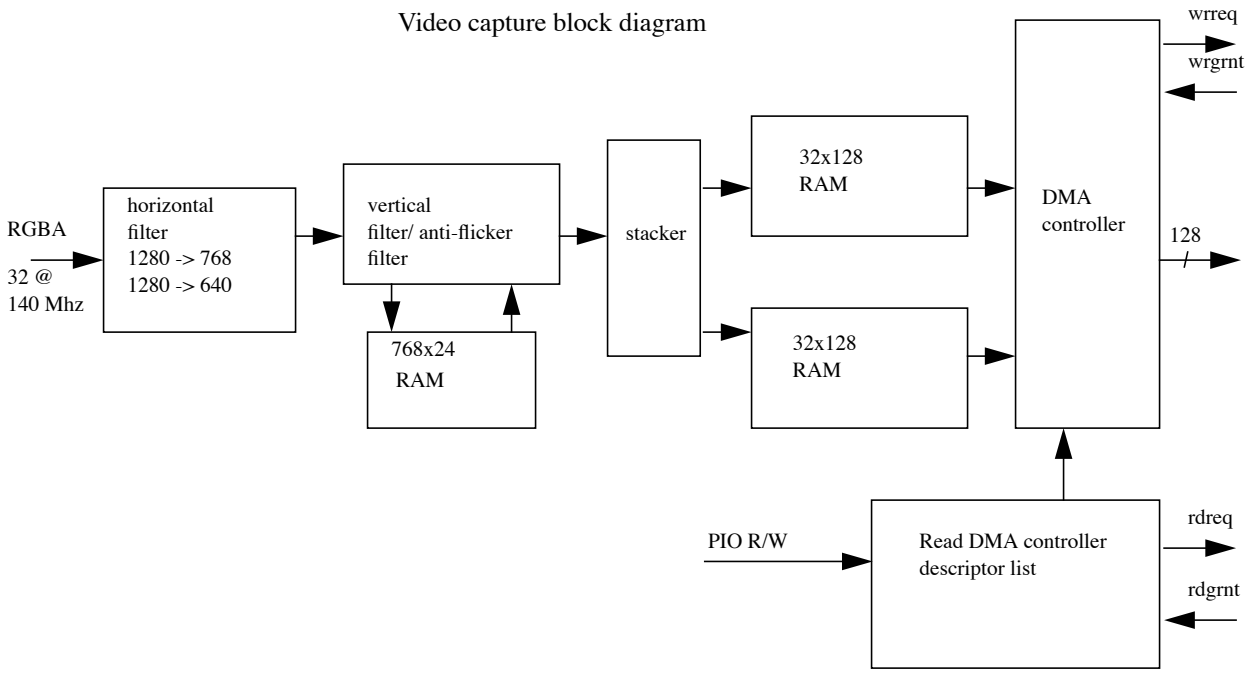


Video capture descriptor list: each 16 bit field points to a 64K byte physical page

The video capture hardware in GBE is capable of transferring one field of video to memory, filling each 64K byte page in the descriptor list in order. Software is responsible for updating the descriptor pointer once per field so that successive fields are written to different main memory buffers. If software does not update the descriptor pointer, GBE will rewrite the same field buffer every vertical CRT scan. The following register controls the video capture DMA controller.

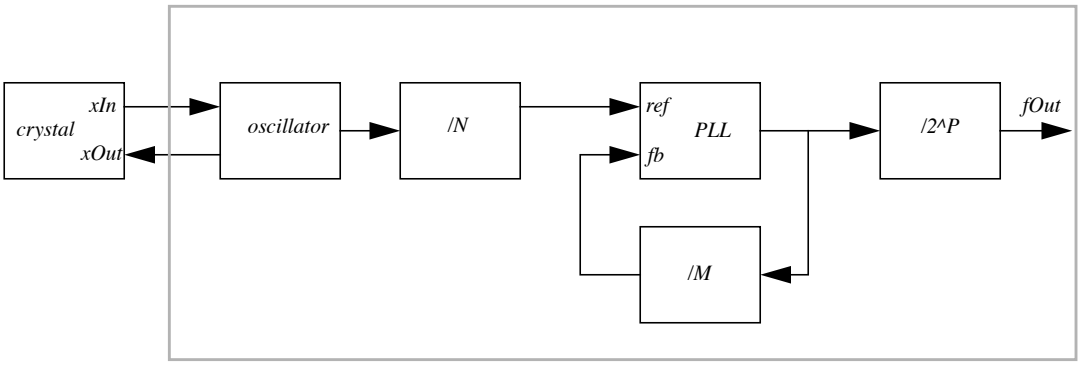


This register may be written at any time. If bit 1=0, then no DMA occurs. Otherwise at the beginning of vertical blanking, GBE will read the 32 byte descriptor list pointed to by the *vc_descr_ptr* field. Note that GBE only examines the *vc_descr_ptr* register at the beginning of the vertical blanking period, so software has a full CRT frame time to update the descriptor list pointer. The filtered pixels from the next active frame are buffered in a pair of 512 byte RAMs and stored into the 64K byte pages pointed to by the descriptor list. Data is always written 512 bytes at a time, so it takes exactly 128 DMA writes to fill a 64K page. After filling a page, GBE will advance to the next descriptor in the list. The total time required to write a 512 byte buffer to main memory must be less than 914 ns in order to avoid overrun. Also note that a video line may cross a 64K boundary in this scheme.



2.13 Dot clock synthesis

GBE contains a PLL-based graphics dot clock synthesis unit. The clock synthesis unit has the following block diagram.



2.13.1 PLL operation

The reference frequency is provided by a 20 MHz crystal connected to the Xin signal of the built in oscillator. The input scaler divides the reference clock by N before entering the PLL phase comparator. The PLL VCO output is fed back to the phase comparator through an M divider and output through a post scaler the divides the frequency by 2^P. Therefore, the output frequency is:

fOut = M * fIn / (N * 2^P)

The M, N, and P parameters have the following precision:

Parameter	Bits	Range	Divider
M	8	0..255	1..256
N	6	0..63	1..64
P	2	0..3	1, 2, 4, 8

The M and N parameters are programmed to the desired value minus one; for example, an M register value of 11 specifies division by 12. The P parameter is directly programmed with the power of two exponent; for example, a P value of 2 specifies division by 4.

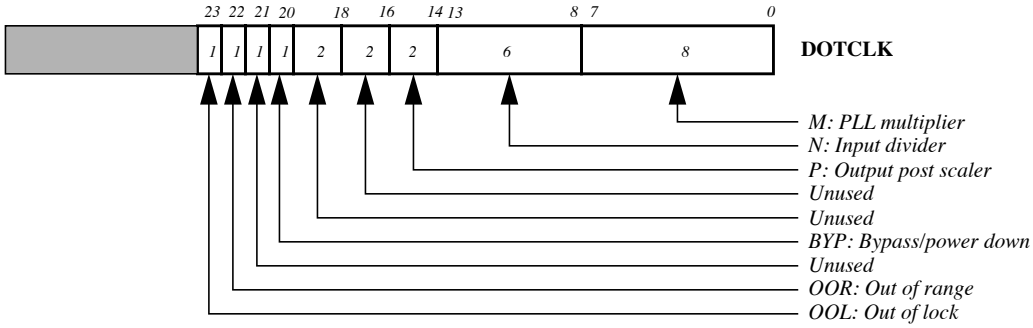
The *oor* and *ool* status bits of the *gbeclk* register allow monitoring of the out-of-range and out-of-lock conditions. The *oor* bit is asserted after TBD reference clock cycles if the VCO has not achieved a stable operating frequency. If the *oor*-

condition occurs, the PLL disables itself and passes the reference clock input through to its output. A restart of the PLL can be attempted by asserting the *clr* bit for 500 us. If *oor* reasserts, the PLL has a hard failure.

When the PLL is in reange, the *ool* bit asserts if after TBD reference clock cycles, the phase error of the PLL is outside of TBD% of the reference clock cycle time. This is not a direct problem for the dot clock since no phase relationship is required, but indicates that the PLL is not operating properly.

OOR	OOL	CLR	PLL condition
1	1	1	PLL in range and phase locked
1	0	1	PLL in range and out of lock
0	0	1	PLL out of range and out of lock
0	X	0	PLL in override mode

There is a *byp* parameter that powers down the PLL and disables the clock to the GBE pixel logic. This allows a low power suspended mode for the majority of the GBE ASIC. When reenabling the PLL, 10 ms must be allowed for the PLL to stabilize. The M, N, P, C, V, *clr*, *oor*, *ool*, and *byp* parameters are accessed via the *gbeclk* register.



The following resolutions and corresponding dot clock frequencies are supported by moosehead:

Active pixels	Active lines	Total width	Total lines	Frames/sec	Frequency (MHz)	Type	Max Err +/- (PPM)
640	480	800	525	59.940	25.175	VGA	6355
800	600	1040	666	60.317	40.000	VESA	5280
800	600	1040	666	72.188	50.000	VESA	4160
1024	768	1320	808	59.940	63.546	SGI	4104
1024	768	1320	808	70.069	75.000	VESA	3541
1024	768	1320	808	75.000	78.750	VESA	3332
1280	1024	1680	1064	48.000	85.882	SGI	3912
1280	1024	1680	1064	50.000	89.544	SGI	3752
1280	1024	1680	1064	59.94	107.245	SGI	3133
1280	1024	1680	1064	72.000	128.701	SGI	2611
1280	1024	1680	1064	75.025	135.000	VESA	2501
1600*	1200*	2112*	1250*	60.000*	162.000*	VESA*	2667*

Active pixels	Active lines	Total width	Total lines	Frames/sec	Frequency (MHz)	Type	Max Err +/- (PPM)
1280	492	?	?	119.880	107.072	SGI	3138
800*	600*	1040*	666*	119.800*	79.448*	SGI*	2658*

* These are not supported with current GBE.

These frequencies would be obtained using the following parameters, assuming a 20 MHz reference clock.

Frequency	M	N	2^P	FVCO	Output	Error
25.175	146	29	4	100.690	25.172	-102.729
40.000	240	30	4	160.000	40.000	0.000
50.000	150	30	2	100.000	50.000	0.000
63.546	197	31	2	127.097	63.548	37.565
75.000	225	30	2	150.000	75.000	0.000
78.750	252	32	2	157.500	78.750	0.000
85.882	146	34	1	85.882	85.882	4.110
89.544	197	44	1	89.545	89.545	16.244
107.245	177	33	1	107.273	107.273	258.541
128.701	251	39	1	128.718	128.718	131.691
135.000	216	32	1	135.000	135.000	0.000
162.000	243	30	1	162.000	162.000	0.000
107.072	182	34	1	79.444	79.444	-44.753

Where the frequency error column results are in PPM and are based on the internal dot clock pll.

2.14 Memory bandwidth analysis

This section discusses the bandwidth usage of GBE in relation to the rest of the system. The Moosehead memory system uses a 256 bit wide SDRAM array running at 66MHz to provide a peak data transfer rate of 2.1 GB/s. The underlying mechanism for accessing memory is still the three step process characteristic of DRAM technology; first destructively read a row into the sense amps, then read/write the sense amps, then write the sense amps back to the row. Because of their relatively slow speed, the row operations effectively reduce the amount of actual bandwidth the memory system is able to deliver. The 16Mbit SDRAMs used in Moosehead have two banks whose row operations can be overlapped. The following discussion assumes the worst case, where there is no overlap due to successive row operations to different banks.

The memory controller inside CRIME is responsible for arbitrating between the various memory masters: CPU, RE, VICE, MACE, and GBE. GBE is the highest priority device. In its maximum configuration, GBE requires a peak bandwidth of 675 MB/s (32+8 bpp). GBE is allowed to burst read 512 bytes at a time from memory, which corresponds to 16 data cycles. Since GBE accesses a different tile on each fetch, it is almost guaranteed to encounter a page miss. The penalty for a page miss varies depending on the type of transition. The possibilities are read-read, read-write, write-read, and

write-write. A read-read miss takes 4 clocks of overhead, while write-read takes 7 clocks, read-write takes 1 clock, and write-write takes 7 clocks. Assuming an average of 6 clocks of overhead, the peak GBE reference pattern is:

6+16 ... 29 ... 6+16 ... 29 ... 6+16 ... 29 ...

Therefore, GBE takes 22 clocks out of 51 during active video, leaving 29 clocks to share between CPU, RE, VICE and MACE. Assuming an average 5 clock penalty and an 8 clock transfer length for the other devices, this leaves about 760 MB/s. The actual left over bandwidth will depend on the average miss penalty and the average burst length, and may be less than 760 MB/s. Overlapping row operations to different banks will reduce the average access time penalty, but this is sensitive to the exact memory address patterns encountered by the memory controller and should not be relied upon if real time access to memory is required.

3.0 Software Interface

All of the registers and RAMs inside GBE are mapped to the physical address range 0x016000000 to 0x017000000. All loads and stores to GBE must be 32 bits and uncached. The following memory map and register descriptions use address offsets from the 0x016000000 base address. All registers in GBE are read/write, and unused bits return undefined values on reads and are ignored on writes. Software is responsible for masking unused bits on reads.

3.1 Memory map

The following memory map is intended for reference.

Name	Address	Description
CTRLSTAT	0x000000	general control
DOTCLK	0x000004	dot clock PLL control
I2C	0x000008	crt I^2C control
SYSCLK	0x00000C	system clock PLL control
I2CFP	0x000010	flat panel I^2C control
DEVICE_ID	0x000014	device id / chip revision
VT_{0..19}	0x010000 - 0x01004C	video timing control
OVR_{0..2}	0x020000 - 0x020008	overlay planes control
FRM_{0..3}	0x030000 - 0x03000C	normal planes control
DID, DID_SHADOW	0x040000 - 0x040004	DID control
WID[0..31]	0x048000 - 0x04807C	WID table RAM
CMAP[0..4607]	0x050000 - 0x0547FC	color map RAM
CMFIFO	0x058000	color map fifo status
GMAP[0..255]	0x060000 - 0x0603FC	gamma map RAM
CRS_0	0x070000	cursor control
CRS_1	0x070004	cursor control
CRS_CM[1..3]	0x070008 - 0x070010	cursor color map
CRS_GLY[0..63]	0x078000 - 0x0780FC	cursor glyph RAM
VC_{0..8}	0x080000 - 0x08000C	video capture control

3.2 Registers

This section summarizes the software-visible state in GBE, and is intended to be a programmer's reference.

3.2.1 Control registers

CTRLSTAT: 0x000000

Bits	Field name	Reset state	Description
3:0	chipid	0001	chip revision number
4	sense_n	-	sense_n input from monitor, read only
5	-	-	-
7:6	io_0	11	bit 6 is a general purpose io pin bit 7 is the active low output enable
9:8	io_1	11	bit 8 is data, bit 9 is oe_n
11:10	io_2	11	bit 10 is data, bit 11 is oe_n
13:12	io_3	11	bit 12 is data, bit 13 is oe_n
15:14	io_4	11	bit 14 is data, bit 15 is oe_n
17:16	io_5	11	bit 16 is data, bit 17 is oe_n
19:18	io_6	11	bit 18 is data, bit 19 is oe_n
21:20	io_7	11	bit 20 is data, bit 21 is oe_n
23:22	io_8	11	bit 22 is data, bit 23 is oe_n
25:24	io_9	11	bit 24 is data, bit 25 is oe_n
26	half_phase	0	sets flat panel 1/2 clock w.r.t the falling edge of blank_n
27	csync_polarity	0	‘1’ makes csync active low
29:28	pclksel	00	“00” = use external TTL pclk input
			“01” = use external differential pclk input
			“11” = use internal dot clock PLL

DOTCLK: 0x000004

Bits	Field name	Reset state	Description
7:0	M	00000000	PLL multiplier
13:8	N	000000	Input divider
15:14	P	00	Output post scaler
17:16	-	-	-
19:18	-	-	-
20	-	-	-
21	-	-	-
22	oor	-	PLL out of range status
23	ool	-	PLL out of lock status
24	tdwni	0	normal operation state of tdwni to dot clock
25	tupi	0	normal operation state of tupi to dot clock

I2C: 0x000008

Bits	Field name	Reset state	Description
0	sda	1	open drain I^2C data
1	sdc	1	open drain I^2C clock

SYSCLK: 0x00000C

Bits	Field name	Reset state	Description
0	tdwni	0	normal operation state of tdwni to system pll
1	tupi	0	normal operation state of tupi to system pll
2	tm(1)	0	normal operation state of tm(1)
3	tm(0)	1	normal operation state of tm(0)
7:4	----	----	----
9:8	vgain	00	normal operation state of vgain
11:10	cgain	00	normal operation state of cgain
14:12	divsel	000	normal operation state of divsel

I2CFP: 0x000010

Bits	Field name	Reset state	Description
0	sda	1	flat panel open drain data
1	sdc	1	flat panel open drain clock

DEVICE_ID: 0x000014

Bits	Field name	Reset state	Description
31:0	device_id	0x00000666	device id (same id that is read from jtag device_id)

3.2.2 Video timing

VT_0: 0x010000

Bits	Field name	Reset state	Description
11:0	vt_x	000000000000	video timing x counter
23:12	vt_y	000000000000	video timing y counter
31	vt_freeze	1	vt_x increments when vt_freeze = ‘0’

VT_1: 0x010004

Bits	Field name	Reset state	Description
11:0	vt_maxx	000000000000	vt_x==vt_maxx causes vt_x to reset
23:12	vt_maxy	000000000000	vt_y==vt_maxy causes vt_y to reset

VT_2: 0x010008

Bits	Field name	Reset state	Description
11:0	vt_vsync_off	000000000000	vsync becomes inactive when vt_y==vt_vsync_off
23:12	vt_vsync_on	000000000000	vsync becomes active when vt_y==vt_vsync_off

VT_3: 0x01000C

Bits	Field name	Reset state	Description
11:0	vt_hsync_off	000000000000	hsync becomes inactive when vt_x==vt_hsync_off
23:12	vt_hsync_on	000000000000	hsync becomes active when vt_x==vt_hsync_on

VT_4: 0x010010

Bits	Field name	Reset state	Description
11:0	vt_vblank_off	000000000000	vblank becomes inactive when vt_y==vt_vblank_off
23:12	vt_vblank_on	000000000000	vblank becomes active when vt_y==vt_vblank_on

VT_5: 0x010014

Bits	Field name	Reset state	Description
11:0	vt_hblank_off	000000000000	hblank becomes inactive when vt_x==vt_hblank_off
23:12	vt_hblank_on	000000000000	hblank becomes active when vt_x==vt_hblank_on

VT_6: 0x010018

Bits	Field name	Reset state	Description
0	vt_vdrv_invert	0	'1' causes inversion of vsync
1	vt_vdrv_low	0	'1' causes vsync='0', or vsync='1' if vt_vdrv_invert='1' used for DPMS monitor power savings
2	vt_hdrv_invert	0	'1' causes inversion of hsync
3	vt_hdrv_low	0	'1' causes hsync='0', or hsync='1' if vt_hdrv_invert='1' used for DPMS monitor power savings
4	vt_sync_high	0	'1' causes sync_n='1'; used to disable sync on green
5	vt_sync_low	0	'1' causes sync_n='0'; used to disable sync on green
6	vt_f2rf_high	0	'1' causes f2rf='1'; used to synchronize left/right

VT_7: 0x01001C

Bits	Field name	Reset state	Description
11:0	vt_f2rf	000000000000	f2rf toggles when vt_y==vt_f2rf
23:12	vt_frmlock	000000000000	vt_y presets to vt_frmlock on rising edge of frmlock WAIT UNTIL vt_x==vt_maxx before resetting vt_y !!!

VT_8: 0x010020

Bits	Field name	Reset state	Description
11:0	vt_intr1	000000000000	GBE issues interrupt #1 when vt_y==vt_intr1
23:12	vt_intr0	000000000000	GBE issues interrupt #0 when vt_y==vt_intr0

VT_9: 0x010024

Bits	Field name	Reset state	Description
11:0	vt_intr3	000000000000	GBE issues interrupt #3 when vt_y==vt_intr3
23:12	vt_intr2	000000000000	GBE issues interrupt #2 when vt_y==vt_intr2

VT_10: 0x010028

Bits	Field name	Reset state	Description
11:0	fp_hdrv_off	000000000000	when vt_x=fp_hdrv_off, clears fp_hdrv
23:12	fp_hdrv_on	000000000000	when vt_x=fp_hdrv_on, sets fp_hdrv

VT_11: 0x01002C

Bits	Field name	Reset state	Description
11:0	fp_vdrv_off	000000000000	when vt_y=fp_vdrv_off, clears fp_vdrv
23:12	fp_vdrv_on	000000000000	when vt_y=fp_vdrv_on, sets fp_vdrv

VT_12: 0x010030

Bits	Field name	Reset value	Description
11:0	fp_de_off	000000000000	when vt_x=fp_de_off, clears fp_de
23:12	fp_de_on	000000000000	when vt_x=fp_de_on, sets fp_de

VT_13: 0x010034

Bits	Field name	Reset value	Description
11:0	vt_hpixen_off	000000000000	when vt_x=vt_hpixen_off, clear internal pixel enable
23:12	vt_hpixen_on	000000000000	when vt_x=vt_hpixen_on, set internal pixel enable

VT_14: 0x010038

Bits	Field name	Reset value	Description
11:0	vt_vpixen_off	000000000000	when vt_y=vt_vpixen_off, clear internal pixel enable
23:12	vt_vpixen_on	000000000000	when vt_y=vt_vpixen_on, set internal pixel enable

VT_15: 0x01003C

Bits	Field name	Reset value	Description
11:0	vt_hcmap_off	000000000000	when vt_x=vt_hcmap_off, clear cmap write enable
23:12	vt_vcmap_on	000000000000	when vt_x=vt_vcmap_on, set cmap write enable

VT_16: 0x010040

Bits	Field name	Reset value	Description
11:0	vt_vcmap_off	000000000000	when vt_y=vt_vcmap_off, clear cmap write enable
23:12	vt_vcmap_on	000000000000	when vt_y=vt_vcmap_on, set cmap write enable

VT_17: 0x010044

Bits	Field name	Reset value	Description
11:0	did_startx	000000000000	when vt_x=did_startx, preset did_x to 000000000000
23:12	did_starty	000000000000	when vt_y=vt_vblank_on, preset did_y to did_starty

VT_18: 0x010048

Bits	Field name	Reset value	Description
11:0	crs_startx	000000000000	when vt_x=crs_startx, preset crs_x to 111111100000
23:12	crs_starty	000000000000	when vt_x-crs_x_offset AND vt_y=vt_vblank_on, preset crs_y to crs_starty. crs_x_offset = vt_hblank_on + 000000100000

VT_19: 0x01004C

Bits	Field name	Reset value	Description
11:0	vc_startx	000000000000	when vt_x=vc_startx, preset vc_x to 000000000000
23:12	vc_starty	000000000000	when vt_y=vt_vblank_on, preset vc_y to vc_starty

3.2.3 Overlay planes

OVR_0: 0x020000

Bits	Field name	Reset state	Description
4:0	ovr_rhs	00000	width of right hand side tile in 32 byte units
12:5	ovr_width_tile	00000000	overlay planes width in tiles (whole tiles)
13	ovr_fifo_reset	0	diagnostic bit, do not modify!

OVR_1: 0x020004

Bits	Field name	Reset state	Description
31:5	ovr_tile_ptr	000...000	pointer to the tile descriptor list for overlay planes
0	ovr_dma_enable	0	‘0’=disable overlay, ‘1’=enable overlay

OVR_2: 0x020008

Bits	Fieldname	Reset state	Description
31:5	ovr_tile_ptr	000...000	copied into OVR_1 ovr_tile_ptr at beginning of blanking
0	ovr_dma_enable	0	copied into OVR_1 ovr_dma_enable at beginning of blanking

3.2.4 Normal planes

FRM_0: 0x030000

Bits	Field name	Reset state	Description
4:0	frm_rhs	00000	width of right hand side tile in 32 byte units
12:5	frm_width_tile	00000000	normal planes width in tiles (whole tiles)
14:13	frm_depth	00	“00” = 8 bit, “01” = 16 bit, “10” = 32 bit
15	frm_fifo_reset	0	diagnostic bit, do not modify!

FRM_1: 0x030004

Bits	Field name	Reset state	Description
31:16	fb_height_pix	0000000000000000	normal/overlay planes height in pixels

FRM_2: 0x030008

Bits	Field name	Reset state	Description
31:5	frm_tile_ptr	000...000	pointer to the tile descriptor list for normal planes
0	frm_dma_enable	‘0’	‘0’=disable normal planes, ‘1’=enable normal planes

FRM_3: 0x03000C

Bits	Fieldname	Reset state	Description
31:5	frm_tile_ptr	000....000	copied to FRM_2 frm_tile_ptr at beginning of blanking.
0	frm_dma_enable	‘0’	copied to FRM_2 frm_dma_enable at beginning of blanking

3.2.5 DID generation

DID: 0x040000

Bits	Fieldname	Resetstate	Description
15:0	did_base	0000000000000000	pointer to 64K DID table block
16	did_dma_enable	'0'	'0'=disable dids, '1'=enable dids

DID_SHADOW: 0x040004

Bits	Field name	Reset state	Description
15:0	did_base	0000000000000000	copied into DID did_base at beginning of blanking
16	did_dma_enable	'0'	copied into DID did_dma_enable at beginning of blanking

WID[0..31]: 0x048000 - 0x04807C

Bits	Field name	Reset state	Description
1:0	buf	-	"01" = lower half, "10" = upper half, "11" = both
4:2	typ	-	"000"=I8, "001"=I12, "010"=RG3B2, "011"=RGB4, "100"=RGB5, "101"=RGB8
9:5	cm	-	upper 5 bits of cmap index for I8 mode, or R,G,B direct visual map for RGB8 pixels if cm != 0
10	gm	-	0=enable gamma, 1=disable gamma
12:11	aux	-	output to digital pixel bus, can be used to externally interpret the 24 bit pixel stream

3.2.6 Color and gamma maps

CMAP[0..4607]: 0x050000 - 0x0547FC

Bits	Field name	Reset state	Description
15:8	blue	-	blue component
23:16	green	-	green component
31:24	red	-	red component

CMFIFO: 0x058000

Bits	Field name	Reset state	Description
5:0	cm_fifo	000000	number of free entries in cmap fifo

GMAP[0..255]: 0x060000 - 0x0603FC

Bits	Field name	Reset state	Description
15:8	blue	-	blue component
23:16	green	-	green component
31:24	red	-	red component

3.2.7 Cursor

CRS_0: 0x070000

Bits	Field name	Reset state	Description
15:0	crs_posx	0000000000000000	cursor x position
31:16	crs_posy	0000000000000000	cursor y position

CRS_1: 0x070004

Bits	Field name	Reset state	Description
0	crs_enable	0	‘1’ means enable cursor
1	crs_crosshair	0	‘1’ means enable crosshair mode

CRS_CM[1..3]: 0x070008 - 0x070010

Bits	Field name	Reset state	Description
15:8	blue	-	blue component
23:16	green	-	green component
31:24	red	-	red component

CRS_GLY[0..63]: 0x0780000 - 0x0780FC

Bits	Field name	Reset state	Description
1:0	crs_pix15	-	rightmost cursor glyph pixel
31:30	crs_pix0	-	leftmost cursor glyph pixel

3.2.8 Video capture

VC_0: 0x080000

Bits	Field name	Reset state	Description
11:0	vc_left	000000000000	left edge of video capture region (inclusive)
23:12	vc_right	000000000000	right edge of video capture region (inclusive)

VC_1: 0x080004

Bits	Field name	Reset state	Description
11:0	vc_top	000000000000	top edge of video capture region (inclusive)
23:12	vc_bottom	000000000000	bottom edge of video capture region (inclusive)

VC_2: 0x080008

Bits	Field name	Reset state	Description
0	vc_dmavideo	0	‘1’ enables video capture dma
1	vc_flick	1	flicker filter enable: ‘0’=disable, ‘1’=enable
2	vc_autofield	1	‘1’- odd/even field sequence automatically advances at oef. ‘0’ field type must be set by PIO
3	vc_fullscreen	0	‘1’ enables fullscreen out mode
4	vc_cscbypass	0	‘1’ enables RGB555 mode that bypasses color space conversion. Intended for diag only

VC_3: 0x08000C

Bits	Field name	Reset state	Description
0	vc_field	0	specifies which field to capture: 0=even, 1=odd
1	vc_discdmaenb	0 (NOTE: This feature not functional on REV1.1. Descriptor list must be PIO’d manually.)	‘0’=disable descriptor list DMA, ‘1’=enable d.l. DMA
2	vc_fieldcorrupt	0	‘1’ - current field corrupt, must be reset by software
3	vc_eof	0	‘1’ - last field successfully DMA’ed, reset by software DMA FIFO fully flushed
4	vc_oddeven	1	READ ONLY, field type currently being DMA’ed
31:5	vc_descr_ptr	000...000	pointer to 32 byte descriptor list

VC_4: 0x080010

Bits	Field name	Reset state	Description
31:16	vc_Tileaddr#1	0x0000	first tile address in descriptor list
15:0	vc_Tileaddr#2	0x0000	second tile address in descriptor list

VC_5: 0x080014

Bits	Field name	Reset state	Description
31:16	vc_Tileaddr#3	0x0000	third tile address in descriptor list
15:0	vc_Tileaddr#4	0x0000	forth tile address in descriptor list

VC_6: 0x080018

Bits	Field name	Reset state	Description
31:16	vc_Tileaddr#5	0x0000	fifth tile address in descriptor list
15:0	vc_Tileaddr#6	0x0000	sixth tile address in descriptor list

VC_7: 0x08001C

Bits	Field name	Reset state	Description
31:16	vc_Tileaddr#7	0x0000	seventh tile address in descriptor list
15:0	vc_Tileaddr#8	0x0000	eigth tile address in descriptor list

VC_8: 0x080020 (WRITE ONLY)

Bits	Field name	Reset state	Description
31:16	vc_Tileaddr#9	0x0000	ninth tile address in descriptor list
15:0	vc_Tileaddr#10	0x0000	tenth tile address in descriptor list

4.0 PLL TEST SUPPORT

GBE contains two PLL blocks. Test support is provided per the recommended usage guidelines from VTI. The following pins are muxed as PLL test pins when spll_teste or dpll_test =’1’. io_off_n should be asserted to ‘0’ during PLL testing. If teste=’1’, then aux(9:0) will drive regardless of the state of io_off_n.

Dot clock PLL test pins

PLL port	Input pin	Output pin	Description
cgain(1:0)	red(7:6)		
multsel(7:0)	red(5:0) & grn(7:6)		
ndivsel(5:0)	grn(5:0)		
pdivsel(1:0)	blu(7:6)		
tclke	blu(1)		
tclki	blu(0)		
tdwni	blu(5)		
tupi	blu(4)		
tm(2:0)	alpha(7:5)		
oor		aux(0)	

Dot clock PLL test pins

PLL port	Input pin	Output pin	Description
ool		aux(1)	
tdwno		aux(2)	
tupo		aux(3)	

System clock PLL test pins

PLL port	Input pin	Output pin	Description
oor		aux(9)	
ool		aux(8)	
div2o		aux(7)	
clkob		aux(6)	
tdwno		aux(5)	
tupo		aux(4)	