GC320 CGPU IP

Composition and 2D Graphics Processing Core:

Technical Reference Manual

for

OMAP4470 – OMAP5430 – OMAP5432

Revision 1.0

20 July 2012

Legal Notices

COPYRIGHT INFORMATION

Vivante Corporation reserves the right to make changes to any products herein at any time without notice. Vivante Corporation does not assume any responsibility or liability arising out of the application or use of any product described herein, except as expressly agreed to in writing by Vivante Corporation; nor does the purchase or use of a product from Vivante Corporation convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual property rights of Vivante Corporation or third parties.

TRADEMARK ACKNOWLEDGMENT

Vivante Corporation and the Vivante Corporation logo design are the trademarks or the registered trademarks of Vivante Corporation. Texas Instruments and the TI logo are the trademarks of Texas Instruments. All other brand and product names may be trademarks of their respective companies.

For our current distributors, sales offices, design resource centers, and product information, visit our web page located at http://www.vivantecorp.com.

For technical support, please contact your local TI representative.

Preface

This document describes the feature set and upper level architecture of the GC320 graphics core.

Audience

Those who would benefit from this technical manual are:

- Engineers and managers who are evaluating GC320 CGPU IP for use in a system
- Engineers who are designing the GC320 CGPU IP into a system.

Organization

This document has the following sections:

- Section 1, **Introduction,** gives a brief description of the CGPU IP.
- Section 2, **2D Operations and Features**, introduces the feature set available with this core.
- Section 3, **Data Formats**, summarizes the data formats availability for input and output.
- Section 4, **Constraints**, describes requirements and constraints.
- Sections 5 – 10, discuss the individual modules which make up the graphics core.
- Section 5, **Host Interface Unit** describes the Host Interface AXI and AHB interface and clocks.
- Section 6, **Power Management**, discusses the power management mechanisms available for low power control.
- Section 7, **Memory Control Units**, provides information on the memory management and control functions in the processor.
- Section 8, **Common Unit**, discusses the common module and its function.
- Section 9, **Front End Unit**, discusses the Fetch Engine and front end module.
- Section 10, **Draw Engine Unit**, discusses the Draw Engine and Pixel Engine module of the core.
- Sections 11 – 15, describes some basic AHB register settings. Please contact us at support@vivantecorp.com or info@vivantecorp.com for the full version.

Conventions Used in This Manual

CGPU – Graphics Processing Unit
SoC – System on Chip
DE – Draw Engine
FE – Graphics Pipeline Front End
HI – Host Interface
MC – Memory Controller
PE – Pixel Engine
GUI – Graphical User Interfaces

The word *assert* means to drive a signal true or active. Signals that are active LOW end in an "n."

Hexadecimal numbers are indicated by the prefix "0x" —for example, 0x32CF.

Binary numbers are indicated by the prefix "0b" —for example, 0b0011.0010.1100.1111

Table of Contents

Legal Notices **2**

Preface **3**

Table of Contents **4**

List of Figures **6**

List of Tables **6**

1 Introduction **7**
1.1 Design Description 7
1.2 Full Featured GC320 Pipeline 8

2 2D OPERATIONS and FEATURES **9**
2.1 Line 9
2.2 Rectangle Fill and Clear 9
2.3 BitBLT 9
2.4 Stretch BLT 10
2.5 Monochrome Expansion and Mask BLT 10
2.6 Filter BLT 11
2.7 Performance of different operations 13
2.8 Other Features 13
2.8.1 ROP Support 13
2.8.2 Rotation 13
2.8.3 Transparency Mode 13
2.8.4 Clipping 14
2.8.5 Data Formats 14
2.8.6 ARGB Data Conversion 14
2.8.7 YUV to RGB Conversion 14
2.8.8 YUV Input and Output Formats 15
2.8.9 Source and Destination Color Key Support 15
2.8.10 Source/Destination Pre-multiply and De-Multiply Support 15
2.8.11 Alpha Blending 16
2.9 CGPU Cache Management 18
2.10 Memory Management Unit 18
2.11 Dither 19
2.12 Multi-Source Blending 19
2.13 One Pass Filter 19
2.14 Tile input and output 19

3 Data Format Summary **20**

4 Constraints **21**
4.1 One pass filter constraints 21
4.2 BitBlit constraints 21
4.3 StretchBlit constraints 21
4.4 Alignment Requirements 22

5 Host Interface Unit **23**
5.1 AXI Interfaces 0 and 1 23

5.2 AHB Interface 23
5.3 Clock Domains 24
5.3.1 Clock Gating 24
5.3.2 Second Level Clock Gating 24
5.3.3 Clock Disabling 24

6 Power Management Unit 25
6.1 AXI Low Power Interface 25

7 Memory Control Units 25
7.1 Memory Controller 25
7.2 MMU Operation 26
7.2.1 MMU Operation 27

8 Common Unit 28

9 Front End Unit 28
9.1 Commands 29

10 Draw Engine 30
10.1 Draw Engine 30
10.2 Pixel Engine 31

11 AHB Accessible Registers 32
11.1 Module Memory Map 32
11.2 AHB Accessible Register Modules 32
11.3 AHB-Accessible Registers 33

12 Host Interface Registers 34

13 Power Management Registers 38

14 Memory Controller Registers 40

15 DMA Unit Registers 43

Document Revision History 44

List of Figures

FIGURE 1. GC320 CGPU IP BLOCK DIAGRAM 8
FIGURE 2. LINE 9
FIGURE 3. BITBLT 10
FIGURE 4. ALPHA BLENDER FOR ONE PIXEL 18
FIGURE 5. HOST INTERFACE BLOCK DIAGRAM 23
FIGURE 6. POWER MANAGEMENT STATE DIAGRAM 25
FIGURE 7. MMU 27
FIGURE 8. FRONT END BLOCK DIAGRAM 28

List of Tables

TABLE 1. BIT BLT FORMATS 10
TABLE 2. FILTER BLT FORMATS 11
TABLE 3. PERFORMANCE OF DIFFERENT OPERATIONS WITHOUT ROTATION 13
TABLE 4. PERFORMANCE OF DIFFERENT OPERATIONS WITH ROTATION 13
TABLE 5. PRE-MULTIPLIED MODES 15
TABLE 6. BLENDING MODES FRACTIONS DESCRIPTION 16
TABLE 7. DATA FORMAT SUMMARY 20
TABLE 8. ALIGNMENT FOR RGB AND YUV SOURCE (METHOD A) 22
TABLE 9. ALIGNMENT FOR RGB AND YUV SOURCE (METHOD B) 22

1 Introduction

Vivante GC320 composition graphics processing unit (CGPU) IP defines a high-performance 2D raster graphics core that accelerates the 2D graphics display on a variety of consumer devices. Addressable screen sizes range from the smallest cell phones to HD 1080p displays. The difference between GC320 and GC320W is that GC320 has two 64 bit AXI bus interfaces whereas GC320W has one 128 bit AXI bus interface and GC320WW has two 128 bit AXI buses.

Vivante CGPU IP is designed for easy integration onto the SoC, providing powerful graphics at low power consumption and the smallest of silicon footprints. The core is delivered as synthesizable RTL. It is technology independent and can be synthesized using any library. Dynamic power consumption is minimized by extensive use of localized clock gating. The design also includes a 32-bit AHB interface, two 64-bit or one or two 128-bit AXI interfaces, and support for virtual memory.

GC320 supports the following graphics APIs:

- BLTsville (Open source)
- DirectFB (on Linux)
- GDI/DirectDraw (on Windows CE)
- Android
- Direct2D (on Windows RT and Windows 8)

1.1 Design Description

The main functional blocks of the GC320 CGPU IP are described here. A block diagram of the complete graphics pipeline is given in the figure below.

Host Interface
Allows the GC320 core to communicate with external memory and the CPU through the AXI or the AHB bus. In this block, data crosses clock domain boundaries.

Memory Controller
Internal memory unit that is the block-to-host interface for memory requests

Graphics Pipeline Front End
Inserts high level primitives and commands into the graphics pipeline.

2D Drawing and Scaling Engine
Draws 2D graphics primitives and rasterizes 2D images.

Pixel Engine
Pixel manipulation and filtering on rendered images. GC320 has four (4) pixel pipes.

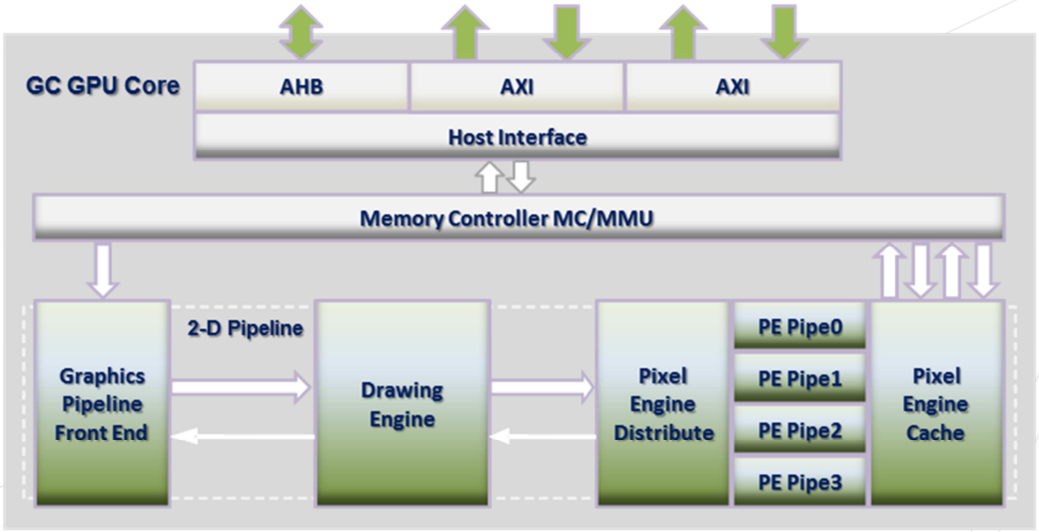

Figure 1. GC320 CGPU IP Block Diagram

1.2 Full Featured GC320 Pipeline

- Bit BLT
- Stretch BLT
- Rectangle fill and clear
- Line drawing
- Filter BLT
- Mono expansion for text rendering
- ROP2, ROP3, and ROP4
- Alpha blending, including Java 2 Porter-Duff compositing blending rules
- 32K x 32K coordinate system
- 90 / 180 / 270 degree rotation
- Transparency by monochrome mask, chroma key, or pattern mask
- Full functional MMU with variable page size support
- Full support for Multi destination support for converting non-planar YUV formats to planar YUV
    - Used in extracting various components from the input color into different destination planes
- Full support for Multi source blending with variable block size to improve BW and reduce SW overhead.
    - Up to 4 sources are supported with v5.2 GC320 v5.3 supports 8 sources.
    - Programmable block size guarantees cache efficiency so each source is read once and each destination is written once.
    - Supports 90, 180, 270 degree rotation with different block size to have high cache efficiency.

2 2D OPERATIONS and FEATURES

2.1 Line

The LINE operation draws a line. Coordinates for two points are given: start point and end point. The end point is not drawn.

Lines are rendered using the Bresenham algorithm.

The Bresenham algorithm has the advantage of using integer arithmetic and has no accumulation of rounding errors.

In the case of line, only ROP2 and ROP4 are supported. It operates on pattern and destination. The pattern should have a transparency mask in order to use ROP4.

Clipping is supported for lines on a per pixel basis.

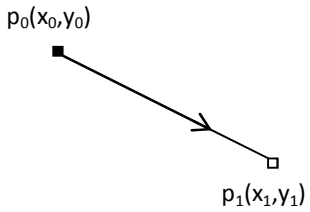

Figure 2. Line

2.2 Rectangle Fill and Clear

Rectangle fill suffuses a rectangle area with a given color. Essentially rectangle fill is a pattern fill, where an 8x8 pattern is initialized with the specified color. It supports ROP2 and ROP4 with the pattern and destination as its inputs. If ROP4 is used, the pattern should have a transparency mask.

Clear is similar to rectangle fill except that it does not does not use a pattern. A 32-bit clear value with 4-bit byte mask is used to fill the entire rectangle area.

Both rectangle fill and clear support clipping, which is performed on a per primitive basis.

2.3 BitBLT

Bit blit transfers data from one area of a memory (source) to another area of the memory (destination). The source and destination can be from the same or from different memory locations. Both source and destination must be described by a rectangular area. The source and destination rectangles can be the same size (most bit blits are of this nature) or they can be different sizes, in which case the operation becomes a stretch or shrink blit.

Bit blit supports ROP2, ROP3, and ROP4 which includes source, destination and pattern, and an optional transparency color.

Clipping can be performed on a primitive basis.

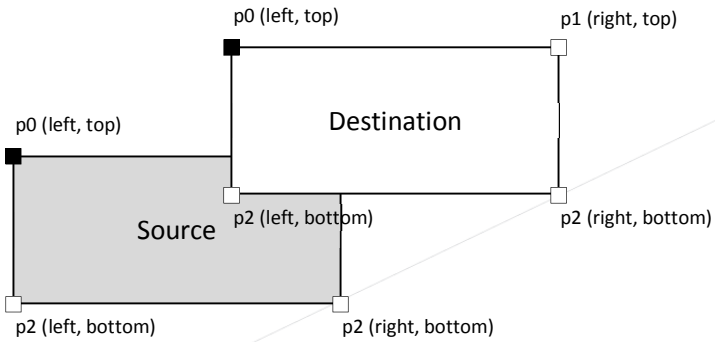

Figure 3. BitBLT

The BIT BLT primitive supports the following 10 source and 7 destination image formats:

Table 1. Bit BLT Formats

Formats	Source Image	Destination Image
A1R5G5B5	Yes	Yes
A4R4G4B4	Yes	Yes
X1R5G5B5	Yes	Yes
X4R4G4B4	Yes	Yes
R5G6B5	Yes	Yes
A8R8G8B8	Yes	Yes
X8R8G8B8	Yes	Yes
A8	Yes	Yes
1-bit monochrome	Yes	No
8-bit color index	Yes	No

2.4 Stretch BLT

The STRETCH BLT primitive performs a BitBlt operation with stretch or shrink. The modified Bresenham algorithm is used to generate corresponding coordinates for fast stretching. The stretch factor is specified in a 15.0 fixed-point format. Stretch blit is not allowed to overlap, that is no part of source and destination can share any piece of memory. Non-stretch blits can overlap. For stretch blit clipping is performed on a per pixel basis.

2.5 Monochrome Expansion and Mask BLT

Monochrome expansion and mask blit are different operations, although both use the bit stream from command buffer. And both can be the source for ROP4 source selection. This means that each output pixel can be a combination of source, pattern, monochrome mask (for masked blits) and destination.

Monochrome expansion

For monochrome expansion, the bit from the stream is used to switch on/off a solid color that is defined in a register. This mechanism enables the use of just one bit per pixel to represent colors. In effect, the MONO EXPANSION primitive increases color representation from one bit per pixel to multiple bits per pixel. A typical application for mono color is font drawing.

Monochrome expansion does not support overlapping of the source and destination. It is the responsibility of the driver to make sure that the command will never be executed on overlapping source and destination.

Mask BLT

For Mask BLT, the bit from the stream is used to toggle on/off a color in the source frame buffer. Mask BLT takes its color source from memory and its monochrome mask from the command stream. Clipping is supported and is performed on a per pixel basis.

2.6 Filter BLT

Filter blit performs high quality scaling, up or down, using an FIR re-sampling filter with up to 9 taps. Sub-pixel coordinates (locations between the pixel grids) are generated by the drawing engine. The filter block in the drawing engine uses the sub-pixel information to select the appropriate filter kernel. GC320 processes 1 pixel every cycle when performing filter blit.

A stretch- or shrink-factor of 15.16 fixed-point format is supported. To generate a single destination pixel requires 9 source pixels. An image is scaled in two passes, one for X-dimension (HOR_FILTER_BLT) and the other for Y-dimension (VER_FILTER_BLT). Software sets up the filter kernel/coefficient table and the kernel size, as well as a temporary buffer for storing intermediate results. After the first pass is completed, intermediate results are sent back to memory, and then the second pass starts to scale the first-pass image. Because of this two-step procedure, the throughput of FILTER BLT is lower than that of STRETCH BLT. Also the Filter Kernel Table may need to be reloaded, and some cycles are consumed in calculating the stepping parameters.

When the stretch or shrink factor is 1, the filterBlit works as a bitBlit copy. It can be used as format converter in that case, for instance, YUV to RGB converter. To use as a format converter, only one pass (HOR_FILTER_BLT or VER_FILTER_BLT) is needed. To optimize the memory bandwidth, when using filterBlit to do YUV to RGB filtering, the temporary target buffer format can be specified as YUY2 to process Y-dimension filtering (VER_FILTER_BLT). This is to avoid converting YUV to A8R8G8B8 in the 1st vertical pass to reduce the memory bandwidth and increase the pixel processing rate. This is the only special case that CGPU may use YUY2 as target format.

The FILTER BLT primitive supports the following 13 source and 7 destination image formats:

Table 2. Filter BLT Formats

Formats		Source Image	Destination Image
A1R5G5B5		Yes	Yes
A4R4G4B4		Yes	Yes
A8R8G8B8		Yes	Yes
R5G6B5		Yes	Yes
X1R5G5B5		Yes	Yes
X4R4G4B4		Yes	Yes
X8R8G8B8		Yes	Yes
YUV	NV12 (4:2:0, 2 planes)	Yes	No
	NV16 (4:2:2, 2 planes)	Yes	No
	UYVY (4:2:2, interleave)	Yes	No
	YUY2 (4:2:2, interleave)	Yes	No
	YV12 (4:2:0, 3 planes)	Yes	No

	8-bit color index	Yes	No

FilterBlit Summary:

- Color space conversion between YUY2 and RGB.
- High quality re-sampling filter with kernel sizes of 1, 3, 5, 7 and 9.
- Stretch factor of format 15.16 fixed-point is supported.
- Programmable filter coefficients.
- FilterBlit support alpha-blending.
- FilterBlit support rotation.
- FilterBlit support BW reduction between vertical and horizontal scaling.
- Clipping is supported and is performed on per pixel basis.

2.7 Performance of different operations

Table 3. Performance of different operations without rotation

Primitive	Peak Performance	Source/destination overlap	Clipping
Line	1 pixel / cycle	N/A	Done on pixel basis
Rectangle	2.4 pixel / cycle	N/A	Done on primitive basis
Clear	2.4 pixel / cycle	N/A	Done on primitive basis
Blit	2.4 pixel / cycle	Overlap is allowed	Done on primitive basis
Stretch blit	1 pixel / cycle	No overlap allowed	Done on pixel basis
Monochrome expansion	1.6 pixel / cycle	No overlap allowed	Done on pixel basis
Filter blit	1 pixel / cycle	N/A	N/A

2.8 Other Features

2.8.1 ROP Support

ROP2 supports 16 ROP types. ROP3 and ROP4 support 256 ROP types.

2.8.2 Rotation

90° / 180° / 270° / X-Flip / Y-Flip / Mirror rotation is supported for all primitives.

Table 4. Performance of different operations with rotation

Primitive	Performance	Source/destination overlap	Clipping
Line	1 pixel / cycle	N/A	Done on pixel basis
Rectangle	2 pixel / cycle	N/A	Done on primitive basis
Clear	2 pixel / cycle	N/A	Done on primitive basis
Blit	2 pixel / cycle	Overlap is allowed	Done on primitive basis
Stretch blit	1 pixel / cycle	No overlap allowed	Done on pixel basis
Monochrome expansion	1 pixel / cycle	No overlap allowed	Done on pixel basis
Filter blit	1 pixel / cycle	N/A	N/A

2.8.3 Transparency Mode

For monochrome expansion:

- Opaque
- Conditional transparency. Transparent if the current pixel matches the specified value.

For blits:

- Opaque
- Masked transparency. Transparent if the mask for the current pixel or pattern is zero.
- Source Conditional transparency. Transparent if the source pixel is within the specified value range.
- Destination Conditional transparency. Transparent if the destination pixel is not within the specified value range.

2.8.4 Clipping

One clipping rectangle is supported for all bitBlit primitives.

2.8.5 Data Formats

The graphics engine supports 14 source data formats. In addition to these 14 source formats, for RGB source formats, CGPU also supports their swizzle formats (ARGB, RGBA, ABGR, BGRA) for RGB formats. For YUV formats, CGPU supports their U/V swap formats.

- A1R5G5B5
- A4R4G4B4
- A8R8G8B8
- R5G6B5
- X1R5G5B5
- X4R4G4B4
- X8R8G8B8
- A8
- NV12
- NV16
- UYVY (4:2:2)
- YUY2 (4:2:2)
- YV12 (4:2:0)
- 8-bit color index

There are 8 destination data formats supported by the graphics engine. In addition to these destination RGB formats, their swizzle formats (ARGB, RGBA, ABGR, BGRA) are also supported.

- A1R5G5B5
- A4R4G4B4
- A8R8G8B8
- R5G6B5
- X1R5G5B5
- X4R4G4B4
- X8R8G8B8
- A8

2.8.6 ARGB Data Conversion

The pixels read from source or destination will be expanded into A8R8G8B8 format to maintain lossless pixel operations. The resulting pixels will be converted into the destination format.

2.8.7 YUV to RGB Conversion

YUV data can be converted into 8-bit per component RGB format at the output of the cache only. Once converted, there is no way back to YUV format. CGPU supports BT.601 and BT.709 YUV to RGB color conversion standards.

In BT.601, the YUV to RGB conversion is done using the following approximation:

$16 \le Y \le 235$
$16 \le U \le 240$

16 ≤ V ≤ 240

A = Y – 16
B = U – 128
C = V – 128

R = clip((298*A + 410*C + 128) >> 8)
G = clip((298*A – 101*B – 209*C + 128) >> 8)
B = clip((298*A + 519*B + 128) >> 8)

The Y, U and V components are clamped prior to the conversion.
Y is clamped between 16 and 235, inclusively.
U and V are clamped between 16 and 240, inclusively.
In BT.709, the R, G, B equations are slightly changed to

R = clip((298*A + 461*C + 128) >> 8)
G = clip((298*A – 55*B – 137*C + 128) >> 8)
B = clip((298*A + 543*B + 128) >> 8)

2.8.8 YUV Input and Output Formats

- Support YUY2 and UYVY output BitBlit and OPF
- Support YUV input format in BitBlit and one pass filter.
  - YUY2 and UYVY package
  - NV12, NV16 and YV12 planer YUV format
- YUV output with alpha blending in BitBlit.

2.8.9 Source and Destination Color Key Support

Both source and destination Color Key are supported.

2.8.10 Source/Destination Pre-multiply and De-Multiply Support

CGPU supports source pre-multiply source alpha or global alpha, or source global color for global colorizing. On destination, CGPU supports destination pre-multiply destination alpha, destination de-multiply alpha.

Table 5. Pre-multiplied modes

Microsoft Alpha Blending	Equation	
S:Fix Alpha	D.C=A*S.C+(1-A)*D.C D.A=A*S.A+(1-A)*D.A	**D:destination** **S:source** **A: alpha** **C: color**
S:Per-Pixel	D.C=S.C+(1-S.A)*D.C D.A=S.A+(1-S.A)*D.A	
S:Fix+Per-Pixel	D.C=A*S.C+(1-S.A)*D.C D.A=A*S.A+(1-S.A)*D.A	
S:Per-Pixel	D.C=S.A*S.C+(1-S.A)*D.C D.A=S.A+(1-S.A)*D.A	

S:Fix+Per-Pixel	D.C=A*S.A*S.C+(1-S.A)*D.C D.A=A*S.A+(1-S.A)*D.A	

2.8.11Alpha Blending

CGPU supports alpha blending together with ROP. Alpha blending function is performed on ROP function result source.

The general alpha blending equations are:

$$Cd = Fs * Cs' + Fd * Cd' \quad (1)$$

$$Ad = Fs * As'' + Fd * Ad'' \quad (2)$$

Where

Cs' is the source color component (adjusted for NPM if necessary)

Cd' is the destination color component (adjusted for NPM if necessary)

As'' is the modified source alpha component

Ad'' is the modified destination alpha component

Fs is fraction of the source that contributes to the final value

Fd is fraction of the destination that contributes to the final value

The blending is done in 5 logical stages (not real implementation stages):

1. Transparent/opaque conversion
   - In this stage, the incoming alpha (source or destination independently) can be inverted if needed to match the internal alpha rule. Internally, an alpha of 0 means transparent, while an alpha of "0xFF" means opaque. External content might follow the opposite rule. The output of the block is either As (Ad for destination) or 1-As (1-Ad for destination).
2. Global value substitution
   - A global alpha value from a register can be used to substitute or scaled the incoming alpha. An incoming alpha As can pass-through, be directly substituted by Ags (global alpha) or scaled by the global alpha value (As * Ags). The source and destination have distinct global alpha values.
3. Blending factor generation
   - At this stages, the blending factors are generated (refer to table below). Each alpha can take the values 0, 1, A or 1-A depending on the blending mode.
4. Final blending
   - This is the final stage which implements the operations described by equations (1) and (2),

The different stages are illustrated in Figure 6.

The fractions take the values described in the following table, depending on the blending mode.

Table 6. Blending Modes Fractions Description

Blending Mode	Fs	Fd
Clear	0	0
SRC	1	0
DST	0	1
SRC_OVER	1	1 – As''

DST_OVER	1 – Ad''	1
SRC_IN	Ad''	0
DST_IN	0	As''
SRC_OUT	1 – Ad''	0
DST_OUT	0	1 – As''
SRC_ATOP	Ad''	1 – As''
DST_ATOP	1 – Ad''	As''
XOR	1 – Ad''	1 – As''

Alpha blending is supported on bit blit and filter blit primitives. To control the blending modes, we have the following register fields:

1. 1 bit for transparent/opaque conversion for source alpha
2. 1 bit for transparent/opaque conversion for destination alpha
3. 2 bits for source alpha modifications, to specify the 3 cases (As, Ags, As*Ags)
4. 2 bits for destination alpha modifications, to specify the 3 cases (Ad, Agd, Ad*Agd)
5. 4 bits to select between the 12 blending modes
6. 8-bits for global source alpha
7. 8-bits for global destination alpha

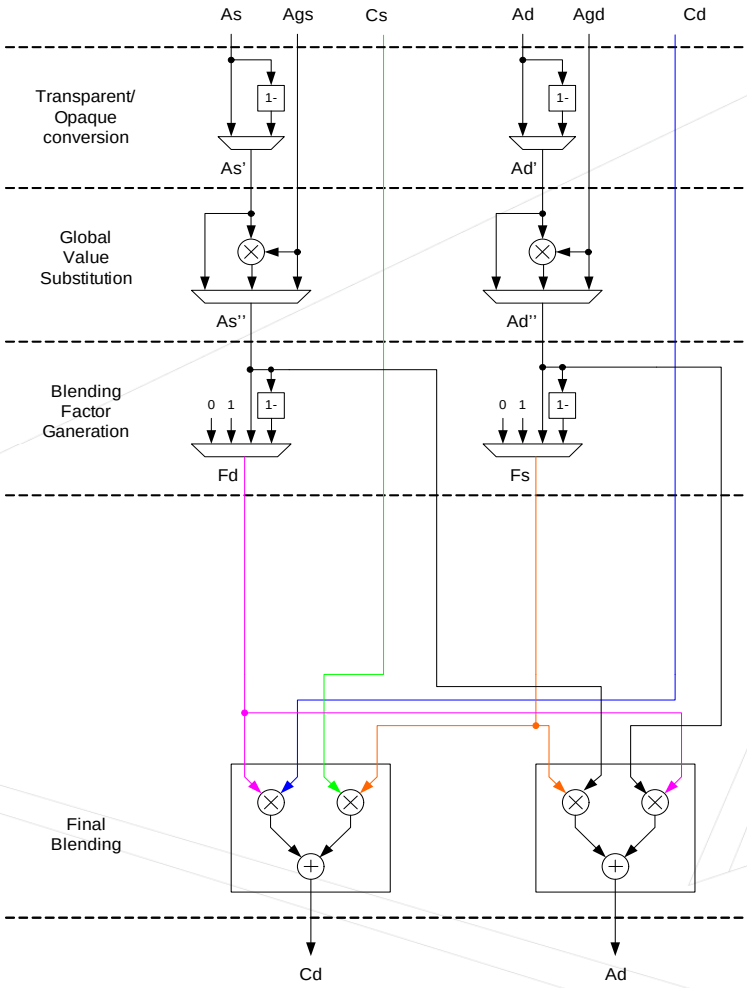

Figure 4. Alpha Blender for One Pixel

2.9 CGPU Cache Management

SW cache flush is supported to flush the CGPU cache to memory. Auto-flush of CGPU cache is also supported. SW sets up the auto-flush interval, and HW will do the cache flush automatically at programmable intervals.

2.10 Memory Management Unit

Full functional MMU with variable page size support.

2.11 Dither

Color intensity at any point of an image can be thought of as a real number between 0.0 and 1.0 at each location. However, in reality only a discrete number of intensity levels can be represented. This discretization leads to a total error in the color intensities of an image equal to the square root of the sum of the squares from each point.

This error results in contouring. In an image where intensities change slowly, this will cause noticeable jumps. Dithering can be used to diffuse the intensity across neighboring pixels.

In dithering mode pixels are scanned in order, and errors in calculating a pixel's intensity are distributed (i.e., *diffused*) to neighboring pixels to keep the overall intensity of the image closer to the input intensity.

Dithering is supported for 16-bit color destination formats. The dither table is programmable by software.

2.12 Multi-Source Blending

Up to 8 sources are supported. Programmable block size guarantees cache efficiency so each source is read once and each destination is written once. It supports 90, 180, 270 degree destination and source rotation and mirror with different block size to have high cache efficiency. It also supports ROP2, ROP3, and ROP4 which includes source, destination and pattern, and an optional transparency color. Alpha blending between every source is supported.

GC320 v5.2 supports 4 sources, GC320 v5.3 supports 8 sources.

2.13 One Pass Filter

- One pass filter support 3x3, 5x5 tap
- SW could control DE walker. There are some config resisters, please look at register define.
- There is no alpha blending and color key once output is YUV format.
- Do not support rotation if output is YUY2 or UYVY format.
- OPF support normal support tile input (4x4 tile), super tile input and multi-tile input.
- OPF support normal support tile output (4x4 tile).

2.14 Tile input and output

Support tile input and output.

- Tile mode support YUV package format and RGB format

Add super tile and multi-tile to input.

- Super tile and multi-tile support YUV package format and RGB format

3 Data Format Summary

The following table gives a summary of the supported formats.

Table 7. Data Format Summary

Format	BitBlit Input	BitBlit Output	FilterBlit Input	FilterBlit Output	OPF Input	OPF Output	Multi-Source Input	Multi-Source Output
A1R5G5B5	Y	Y	Y	Y	Y	Y	Y	Y
A4R4G4B4	Y	Y	Y	Y	Y	Y	Y	Y
A8R8G8B8	Y	Y	Y	Y	Y	Y	Y	Y
X1R5G5B5	Y	Y	Y	Y	Y	Y	Y	Y
X4R4G4B4	Y	Y	Y	Y	Y	Y	Y	Y
X8R8G8B8	Y	Y	Y	Y	Y	Y	Y	Y
R5G6B5	Y	Y	Y	Y	Y	Y	Y	Y
A8	Y	Y	Y	Y	N	N	N	N
RG16	Y	Y	Y	Y	N	N	N	N
YUY2 (package YUV422)	Y	Y	Y	N	Y	Y	Y	Y
UYVY (package YUV422)	Y	Y	Y	N	Y	Y	Y	Y
YV12 (planar YUV420)	Y	N	Y	N	Y	N	Y	N
NV12 (semi-planar YUV420)	Y	N	Y	N	Y	N	Y	N
NV16 (semi-planar YUV422)	Y	N	Y	N	Y	N	Y	N
8-bit color index	Y	N	Y	N	Y	N	Y	N
1-bit monochrome	Y	N	N	N	N	N	N	N

4 Constraints

4.1 One pass filter constraints

- One pass filter can be configured as 5-tap or 3-tap.
- SW could control DE walker. There are some configation resisters, please look at register definitions.
- There's no alpha blending and color_key once output is YUV format.
- We do not support rotation if output is YUY2 or UYVY format.
- SW could control YUV2RGB and YUV bypass in pipeline if both input and output are YUV format.
- OPF support normal support tile input (4x4 tile), super tile input and multi-tile input.
- OPF support normal support tile output (4x4 tile).

4.2 BitBlit constraints

- There's no alpha blending and color_key once output is YUV format.
- We do not support rotation if output is YUY2 or UYVY format.
- SW could control YUV2RGB and YUV bypass in pipeline if both input and output are YUV format.
- BitBlit support normal support tile input (4x4 tile), super tile input and multi-tile input.
- BitBlit support normal support tile output (4x4 tile).

4.3 StretchBlit constraints

- StretchBlit support normal support tile input (4x4 tile), super tile input and multi-tile input.
- StretchBlit support normal support tile output (4x4 tile).

4.4 Alignment Requirements

Vivante CGPU 2D cores GC320 v5_2_2 and before have the following alignment requirements for RGB and YUV source.

- RGB formats include: A1R5G5B5, A4R4G4B4, A8R8G8B8, X1R5G5B5, X4R4G4B4, X8R8G8B8, R5G6B5, A8.
- Packed YUV formats (YUV422) include: YUY2 (packed YUV422) and UYVY (packed YUV422).
- Planar YUV formats (YUV420) include: YV12 (planar YUV420), NV12 (semi-planar YUV420) and NV16 (semi-planar YUV422).

Table 8. Alignment for RGB and YUV Source (Method A)

2D Engine	RGB Source	RGB Destination	Packed YUV422 Source	Packed YUV422 Destination	Planar YUV420 Source	Planar YUV420 Destination
Width	1	1	2	No YUV output	16	No YUV output
Height	1	1	1	No YUV output	2	No YUV output
Buffer address	64 bytes	64 bytes	64 bytes	64 bytes	64 bytes	64 bytes
Stride	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes	16 bytes

Vivante CGPU 2D cores GC320 v5_3_0 and later have the following alignment requirements for RGB and YUV source.

For planar YUV there is an 8 pixel (or byte) alignment requirement for Y, and 4 pixel for U/V, e.g., when the Y stride is 40, both U and V stride is 20. Also, note that the Y stride needs to be greater or equal to 32.

Table 9. Alignment for RGB and YUV Source (Method B)

2D Engine	RGB Source	RGB Destination	Packed YUV422 Source	Packed YUV422 Destination	Planar YUV420 Source	Planar YUV420 Destination
Width	1	1	2	2	16	2
Height	1	1	1	1	2	2
Buffer address	1 pixel	1 pixel	2 pixels	2 pixels	64 bytes	64 bytes
Stride	1 pixel	1 pixel	2 pixels	2 pixels	8 bytes Y stride needs to be ≥ 32	8 bytes

5 Host Interface Unit

Two major interfaces are used to connect the CGPU to the rest of the SoC: AXI and AHB. The AXI master interface included in the CGPU IP is used to fetch data from memory that is attached to the SoC AXI interconnect. The AHB slave interface is used by the SoC processor to access the graphics IP registers for configuration, debug, and test.

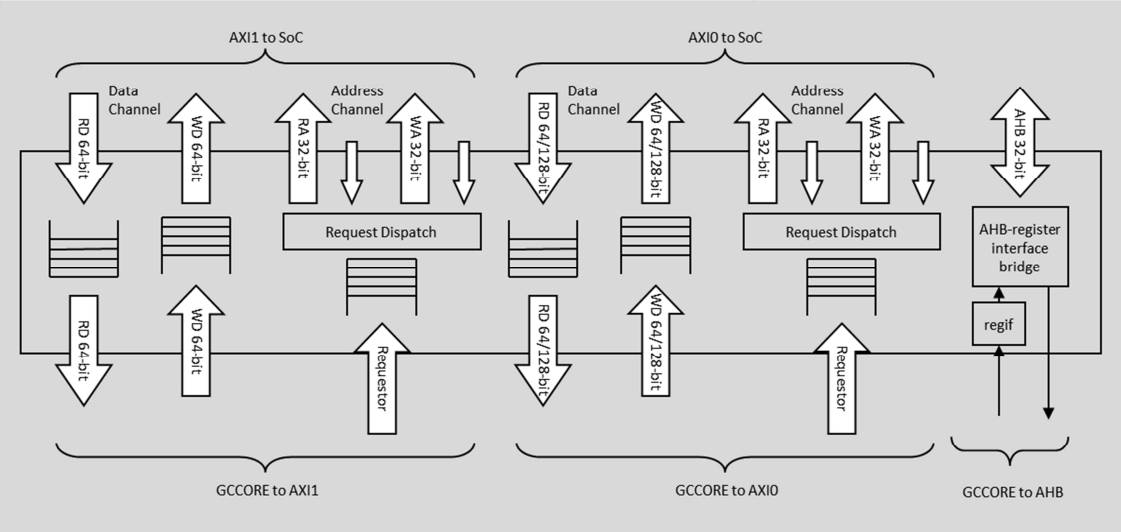

Figure 5. Host Interface Block Diagram

5.1 AXI Interfaces 0 and 1

Main features of each AXI interface:

- Independent read and write data buses
  - Two 64-bit AXI buses in GC320
  - One 128-bit AXI bus in GC320W
  - Two 128-bit AXI buses in GC320WW
- A spreader block can take read requests and send them to different AXIs, and do the same for writes. The read data will be confined to 128-bit throughput.
- Multiple burst length (8 bytes, 16 bytes, 32 bytes, or 64 bytes)
- Supports out-of-order return data for different clients
- Asynchronous interface to the CGPU
- Optimized for size while meeting performance requirements

If the AXI DMA master receives an ERROR response from the AXI fabric, GC320 will log it and generate an INTERRUPT.

5.2 AHB Interface

Main features:

- 256 Kbyte addressable register space
- 32-bit accesses only (no bursts)

- 32-bit data bus
- Handles error response for illegal accesses
- Asynchronous interface to the CGPU
- Interrupt support

The AHB interface uses a separate clock, which is slower than the AXI bus clock and allows more relaxed logic design. Because the core clock is different from the interface clock, incoming data and outgoing data are handled properly across the clock boundary.

GC320W decodes bits HADDR[17:2] for CGPU internal address ; HADDR[31:18] should decode HSEL. GC320W occupies 256 Kbytes (64K 32-bit words) of system address space. However, the range of addresses used is only 4 Kbytes.

An AHB ERROR response will be returned if an illegal access is detected.

Only 32-bit reads and writes are permitted.

5.3 Clock Domains

There are three independent clock domains in the CGPU IP:

Core clock, which is derived from the clk2x pin

AHB clock, which is derived from the HCLK pin, and

AXI clock, which is derived from the ACLK0 pin.

Communication between the different domains occurs mostly by way of asynchronous FIFO's. All three clocks are asynchronous to each other. The core clock is used in CGPU core logic.

There is no communication between AHB and AXI clock domains.

The core clock can be scaled up and down dynamically without reprogramming its source. This scaling is controlled through use of an internal register.

5.3.1 Clock Gating

Automatic localized clock gating is used throughout the design in order to minimize the dynamic power consumption. Over 94% of the registers are gated after synthesis.

5.3.2 Second Level Clock Gating

Block level clock gating is implemented in a majority of the blocks. If a block and the interface to the block are idle, that particular block's clocks will be gated. This is done automatically. This feature can be disabled by software.

5.3.3 Clock Disabling

The core clock enters the frequency scaling block, and a clock with missing pulses is output, buffered, and sent to clock gating logic. The clock gate output will be used by the 2D CGPU blocks. The core clock can be shut down through software by setting the proper register bits.

6 Power Management Unit

6.1 AXI Low Power Interface

The following state diagram illustrates the AXI Low Power Interface. Please note that the shut-down sequence cannot complete unless 2D is completely idle. Unlike other blocks, the FE cannot become idle unless an END command is explicitly sent to disable it.

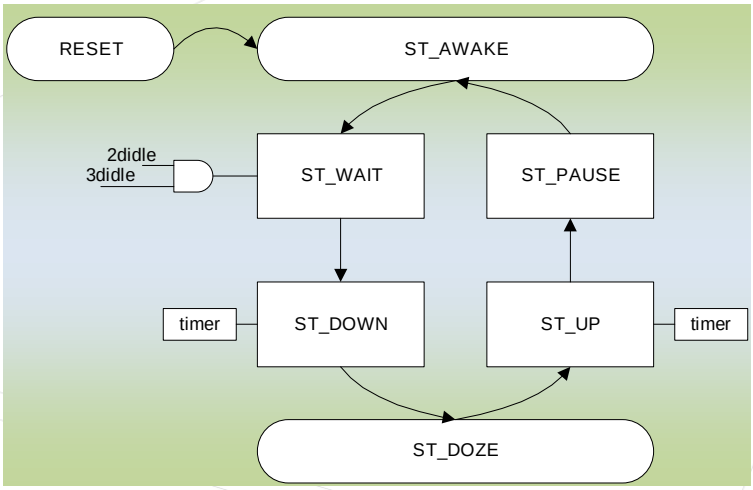

Figure 6. Power Management State Diagram

7 Memory Control Units

Memory management is performed through two modules, the Memory Controller (MC) and the Memory Management Unit (MMU).

7.1 Memory Controller

The memory controller supports read and write requests in the following sizes:

- bytes
- 16 bytes
- 32 bytes
- 64 bytes

The AXI interface takes care of the asynchronous boundary crossing between core clock and the AXI interface. The data buses widths are matched between the AXI and the MC.

Using byte masks, write requestors can have byte granularity in their accesses.

Clients cannot stall the memory controller. Read request return data must be accepted at speed by their respective client. This design decision was made in order to guarantee the memory controller throughput to all clients.

Only one read bus is used to return the read data because there is enough bandwidth to satisfy the clients' bandwidth requirements. Also, there is only one source as far as the MC is concerned (AXI), so there is no opportunity for parallel read return data transfer.

A client can stall under the following circumstances:

- Corresponding request FIFO is full.
- For the read FIFO only, concurrent read request at input of read FIFO.
- Virtual translation miss

7.2 MMU Operation

The MMU supports variable page sizes (4KB, 64KB, 1MB, and 16MB). The pages sizes and address decoding matches the ARM MMU implementation. The MMU supports hierarchical translation look-up. The MMU has a larger cache with prefetch. It has Global translation enable/disable (full use for 32 bit address).

Each address has 3 fields as shown in the figure below.

- Master TLB offset.
- Slave TLB offset.
- Address offset.

The width of these fields is variable based on the current mode of operation. First, the master TLB is found using the master TLB base address and the master TLB offset. From that address, the slave TLB base address is found. Using the slave TLB base address and the slave TLB offset, the address translation is obtained. The address offset is added to this translation to find the final address.

The advantages of hierarchical look up include:

- Smaller page tables.
- Easier to add/remove entries.
- Easier to manage in SW.
- Different surfaces can use different page table sizes.

7.2.1 MMU Operation

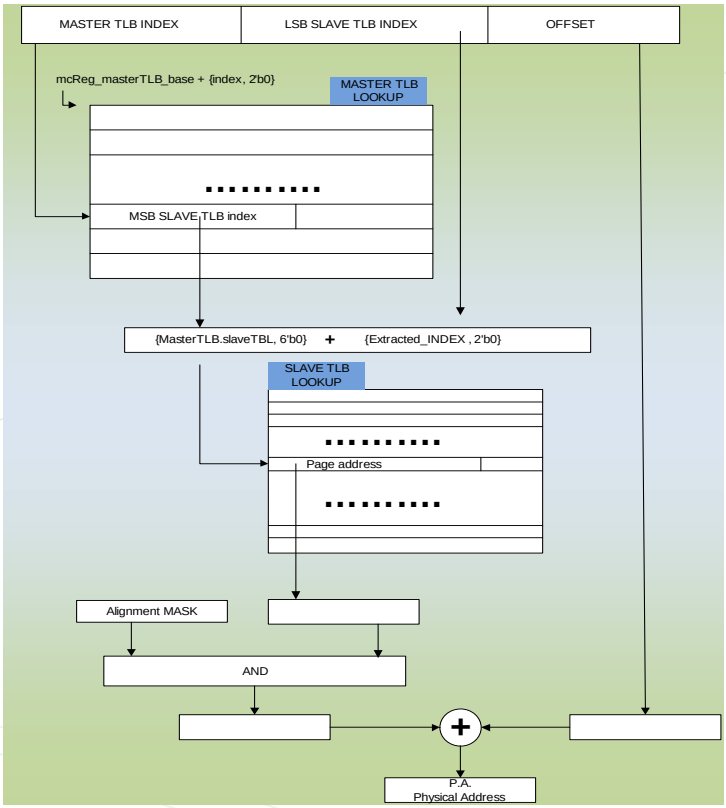

Figure 7. MMU

8 Common Unit

The common unit contains registers which may be shared or used by more than one of the other modules in the core. This unit has no dedicated function.

9 Front End Unit

The Front End (FE) block, sometimes also referred to as the Fetch Engine or DMA module, is the CGPU Front End that controls the input traffic of command streams and DMA data feeding the graphics pipe. It interfaces with the local memory to pull in the various data. Once pulled in, the front end assembles and formats the data.

The front end contains a buffer that acts as a data assembly FIFO. The output of the cache or data buffer goes to a format/conversion unit.

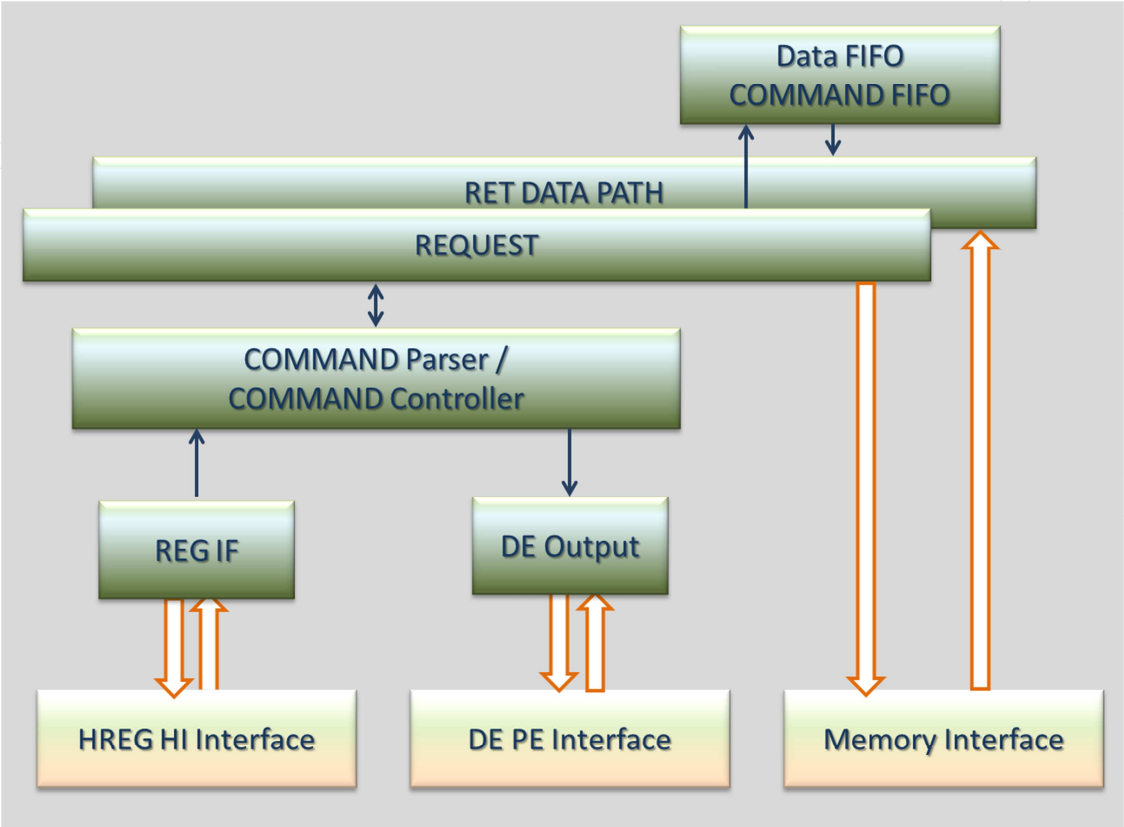

Figure 8. Front End Block Diagram

9.1 Commands

Commands and data are sent to the CGPU through the Front End interface. Commands may be divided into two categories: those that modify the state of the CGPU (e.g., LOAD_STATE) and those that do not (e.g., STALL, WAIT and LINK).

The following Command OPCODES are supported for Vivante 2D Raster Graphics processors:

01 => LOAD_STATE

02 => END

03 => NOP

04 => START_DE

07 => WAIT

08 => LINK

09 => STALL

0A => CALL

0B => RETURN

Refer to the Register Specification for the organization of specific Commands.

10 Draw Engine

The Draw Engine handles Drawing, Scaling, Rotation and pipeline management for 2D raster graphics.

10.1 Draw Engine

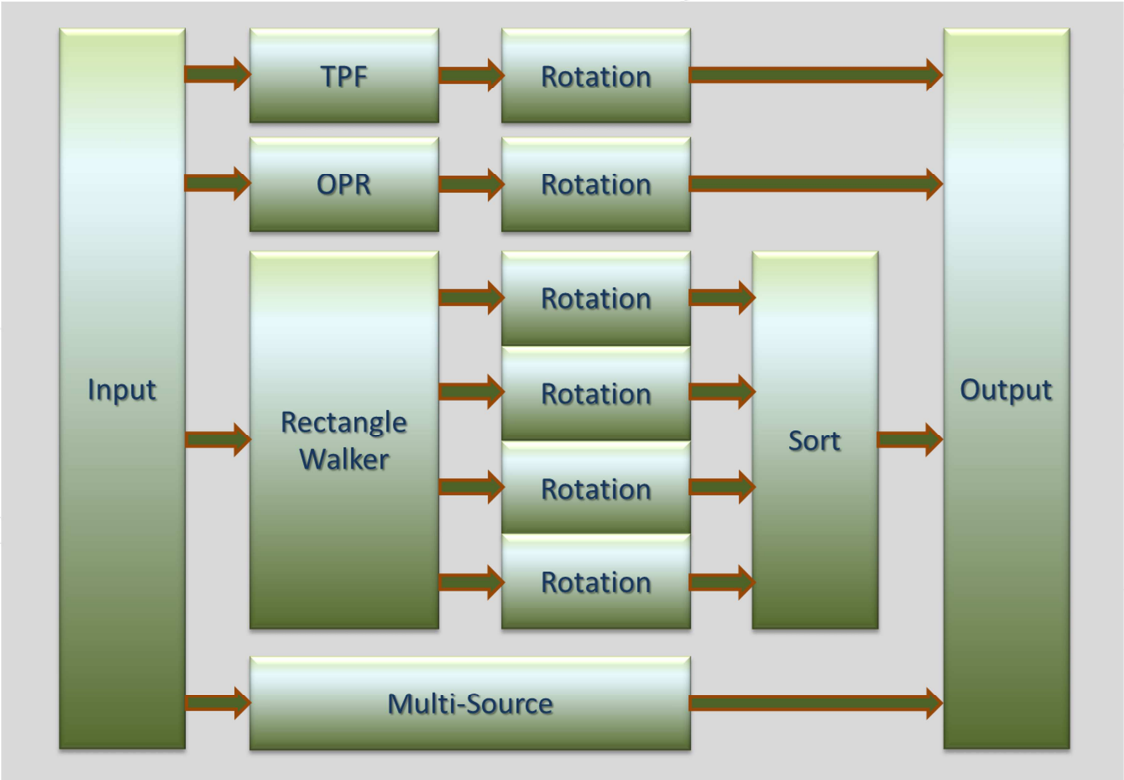

Figure 9. GC320 Drawing Engine Block Diagram

GC320 drawing engine includes a new rectangle walker for BitBlit, mono, line and one pass filtering.

10.2 Pixel Engine

All pixel engine activities are performed through the draw engine block.

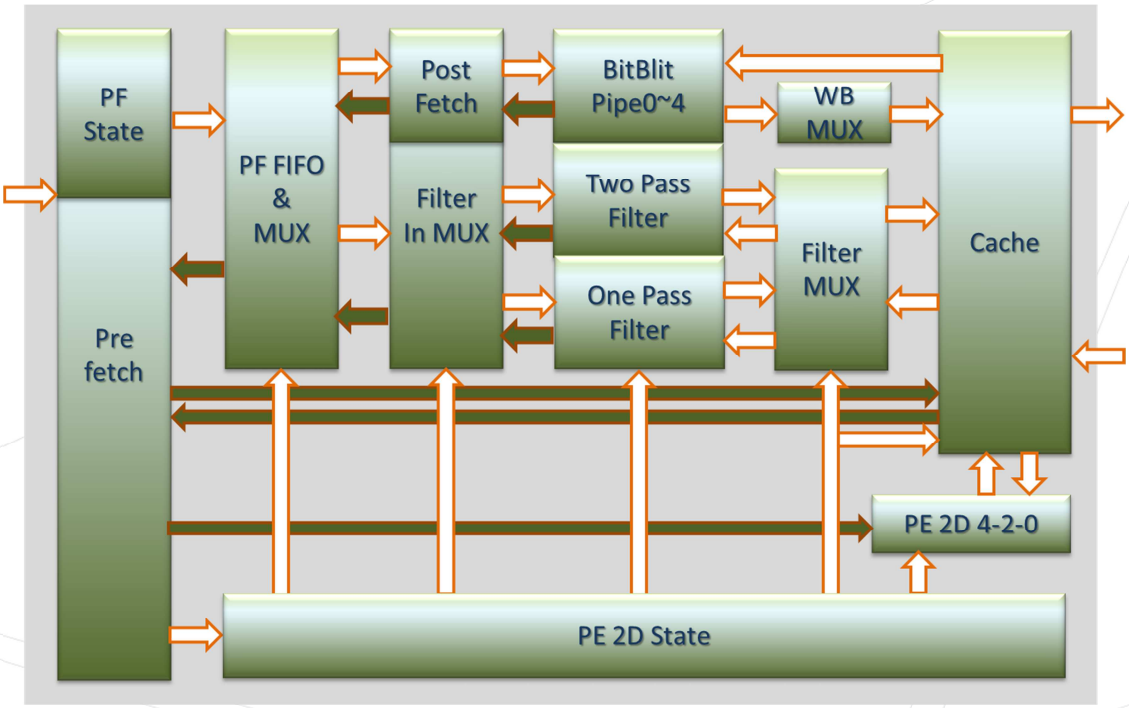

Figure 10. GC320 Pixel Engine Block Diagram

11 AHB Accessible Registers

Note: Sections 11 - 15 contain an abbreviated list of GC320 registers. If you would like the complete version of the register set, please contact us at support@vivantecorp.com or info@vivantecorp.com.

11.1 Module Memory Map

Address	Size	Module
0x0000	0x0040	Host Interface
0x0040	0x0020	Power Management
0x0060	0x0020	MMU
0x0080	0x0080	reserved
0x0100	0x0080	Memory Controller
0x0180	0x0080	DMA Front End

11.2 AHB Accessible Register Modules

The following modules are accessible via the AHB bus:

- Host Interface
- Power Management
- Memory Controller
- DMA Front End

11.3 AHB-Accessible Registers

The following AHB-accessible register fields are accessible to programmers wanting to take advantage of features in the GC320.

AHB Module	Exposed Register
HI	AQAxiConfig
HI	AQAxiStatus
HI	AQHiClockControl
HI	AQHiIdle
HI	AQIdent
HI	AQIntrAcknowledge
HI	AQIntrEnbl
HI	gcAxiControl
HI	GCChipDate
HI	GCChipId
HI	GCChipRev
HI	GCChipTime
HI	gcTotalCycles
HI	gcTotalIdleCycles
PM	gcModulePowerControls
PM	gcModulePowerModuleControl
PM	gcModulePowerModuleStatus
MC	AQMemoryDebug
MC	AQRegisterTimingControl
MC	gcDbgCycleCounter
MC	gcOutstandingReads0
MC	gcOutstandingReads1
MC	gcOutstandingWrites
MC	gcregEndianness0
MC	gcregEndianness1
MC	gcregEndianness2
DMA	AQCmdBufferAdr
DMA	AQCmdBufferCtrl

12 Host Interface Registers

The Host Interface is a bridge between the Memory Controller and the AXI interface. It also acts as a bridge between the core and the AHB bus. The Host Interface's register set contain the control settings for the clocks, AXI interface and interrupts. It also contains the identification registers and some counters that are used for debug. Users can access all of the Host Interface registers via the AHB bus.

AQHiClockControl

Description: Clock control register.

Reset Value = 00070100	Address = 0x0000	Count = 1
Field Mask = 0FFF1FFF	ReadMask = 0FFF1FFF	Write Mask = 0FF81FFF

Sub-Field Name	Bits	Description
CLK3D_DIS	0:0	Disable 3D clock.
CLK2D_DIS	1:1	Disable 2D clock.
FSCALE_VAL	8:2	
FSCALE_CMD_LOAD	9:9	
DISABLE_RAM_CLOCK_GATING	10:10	Disables clock gating for rams.
DISABLE_DEBUG_REGISTERS	11:11	Disable debug registers. If this bit is 1, debug registers are clock gated.
SOFT_RESET	12:12	Soft resets the IP.
IDLE3_D	16:16	3D pipe is idle.
IDLE2_D	17:17	2D pipe is idle.
IDLE_VG	18:18	VG pipe is idle.
ISOLATE_GPU	19:19	Isolate GPU bit
MULTI_PIPE_REG_SELECT	23:20	Determines which HI/MC to use while reading registers.
MULTI_PIPE_USE_SINGLE_AXI	27:24	Force all the transactions to go to one AXI.

AQHiIdle

Description: Idle status register.

Reset Value = 7fffffff	Address = 0x0001	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = 00000000

Sub-Field Name	Bits	Description
IDLE_FE	0:0	FE is idle.
IDLE_DE	1:1	DE is idle.
IDLE_PE	2:2	PE is idle.
IDLE_SH	3:3	SH is idle.
IDLE_PA	4:4	PA is idle.
IDLE_SE	5:5	SE is idle.
IDLE_RA	6:6	RA is idle.
IDLE_TX	7:7	TX is idle.
IDLE_VG	8:8	VG is idle.

IDLE_IM	9:9	IM is idle.
IDLE_FP	10:10	FP is idle.
IDLE_TS	11:11	TS is idle.
UNUSED	30:12	Unused bits reserved for future expansion.
AXI_LP	31:31	AXI is in low power mode.

AQAxiConfig

Description:

Reset Value = 00000000 Address = 0x0002 Count = 1

Field Mask = 0000FFFF ReadMask = 0000FFFF Write Mask = 0000FFFF

Sub-Field Name	*Bits*	*Description*
AWID	3:0	
ARID	7:4	
AWCACHE	11:8	
ARCACHE	15:12	

AQAxiStatus

Description:

Reset Value = 00000000 Address = 0x0003 Count = 1

Field Mask = 000003FF ReadMask = 000003FF Write Mask = 00000000

Sub-Field Name	*Bits*	*Description*
DET_RD_ERR	9:9	
DET_WR_ERR	8:8	
RD_ERR_ID	7:4	
WR_ERR_ID	3:0	

AQIntrAcknowledge

Description: Interrupt acknowledge register. Each bit represents a corresponding event being triggered. Reading frmo this register clears the outstanding interrupt.

Reset Value = 00000000 Address = 0x0004 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = 00000000

Sub-Field Name	*Bits*	*Description*
INTR_VEC	31:0	

AQIntrEnbl

Description: Interrupt enable register. Each bit enables a corresponding event.

Reset Value = 00000000 Address = 0x0005 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = FFFFFFFF

Sub-Field Name	*Bits*	*Description*
INTR_ENBL_VEC	31:0	

AQIdent

Description: Identification register. This register has no set reset value. It varies with the implementation.

Reset Value = 14010000 Address = 0x0006 Count = 1

Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = 00000000
Sub-Field Name	*Bits*	*Description*
FAMILY	31:24	Family value. The field is obsolete since GC500, refer to GCChipId instead. 01 => GC500 02 => GC520 03 => GC530 04 => GC400 05 => GC450 08 => GC600 09 => GC700 0A => GC350 0B => GC380 0C => GC800 10 => GC1000 14 => GC2000
PRODUCT	23:16	Product value.
REVISION	15:12	Revision value.
TECHNOLOGY	11:8	Technology value.
CUSTOMER	7:0	Customer value.

GCChipId

Description: Shows the ID for the chip in BCD. This register has no set reset value. It varies with the implementation.

Reset Value = 00000320	Address = 0x0008	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = 00000000

Sub-Field Name	*Bits*	*Description*
ID	31:0	Id.

GCChipRev

Description: Shows the revision for the chip in BCD. This register has no set reset value. It varies with the implementation.

Reset Value = 00005222	Address = 0x0009	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = 00000000

Sub-Field Name	*Bits*	*Description*
REV	31:0	Revision.

GCChipDate

Description: Shows the release date for the IP. This register has no set reset value. It varies with the implementation.

Reset Value = 20110615	Address = 0x000A	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = 00000000

Sub-Field Name	*Bits*	*Description*
DATE	31:0	Date.

GCChipTime

Description: Shows the release time for the IP. This register has no set reset value. It varies with the implementation.

Reset Value = 22332800	Address = 0x000B	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = 00000000

Sub-Field Name	Bits	Description
TIME	31:0	Time.

gcAxiControl

Description: Special Handling on AXI Bus

Reset Value = 00000000	Address = 0x001C	Count = 1
Field Mask = 00000001	ReadMask = 00000001	Write Mask = 00000001

Sub-Field Name	Bits	Description
WR_FULL_BURST_MODE 0 => NO_BURST_RESET_VALUE 1 => BURST_RESET_VALUE	0:0	

gcTotalCycles

Description: Total cycles. This register is a free running counter. It can be reset by writing 0 to it.

Reset Value = 00001de2	Address = 0x001E	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = FFFFFFFF

Sub-Field Name	Bits	Description
CYCLES	31:0	

gcTotalIdleCycles

Description: Total cycles where the GPU is idle. It is reset when gcTotalCycles register is written to. It looks at all the blocks but FE when determining the IP is idle.

Reset Value = 00001e08	Address = 0x001F	Count = 1
Field Mask = FFFFFFFF	ReadMask = FFFFFFFF	Write Mask = FFFFFFFF

Sub-Field Name	Bits	Description
CYCLES	31:0	

13 Power Management Registers

The Power Management register set has just a few registers for controlling clock gating within the core. The GCCORE allows the user to control the clock gating of each internal module independently of the other modules. Users can access all of these registers via the AHB Bus.

gcModulePowerControls

Description: Control register for module level power controls.

Reset Value = 00140020	Address = 0x0040	Count = 1
Field Mask = FFFF00F7	ReadMask = FFFF00F7	Write Mask = FFFF00F7

Sub-Field Name	Bits	Description
ENABLE_MODULE_CLOCK_GATING	0:0	Enables module level clock gating.
DISABLE_STALL_MODULE_CLOCK_GATING	1:1	Disables module level clock gating for stall condition.
DISABLE_STARVE_MODULE_CLOCK_GATING	2:2	Disables module level clock gating for starve/idle condition.
TURN_ON_COUNTER	7:4	Number of clock cycles to wait after turning on the clock.
TURN_OFF_COUNTER	31:16	Counter value for clock gating the module if the module is idle for this amount of clock cycles.

gcModulePowerModuleControl

Description: Module level control registers.

Reset Value = 00000000	Address = 0x0041	Count = 1
Field Mask = 000000FF	ReadMask = 000000FF	Write Mask = 00001BFF

Sub-Field Name	Bits	Description
DISABLE_MODULE_CLOCK_GATING_FE	0:0	Disables module level clock gating for FE.
DISABLE_MODULE_CLOCK_GATING_DE	1:1	Disables module level clock gating for DE.
DISABLE_MODULE_CLOCK_GATING_PE	2:2	Disables module level clock gating for PE.
DISABLE_MODULE_CLOCK_GATING_SH	3:3	Disables module level clock gating for SH.
DISABLE_MODULE_CLOCK_GATING_PA	4:4	Disables module level clock gating for PA.
DISABLE_MODULE_CLOCK_GATING_SE	5:5	Disables module level clock gating for SE.
DISABLE_MODULE_CLOCK_GATING_RA	6:6	Disables module level clock gating for RA.
DISABLE_MODULE_CLOCK_GATING_TX	7:7	Disables module level clock gating for TX.

gcModulePowerModuleStatus

Description: Module level control status.

Reset Value = 00000000	Address = 0x0042	Count = 1
Field Mask = 000000FF	ReadMask = 000000FF	Write Mask = 00000000

Sub-Field Name	Bits	Description
MODULE_CLOCK_GATED_FE	0:0	Module level clock gating is ON for FE.

MODULE_CLOCK_GATED_DE	1:1	Module level clock gating is ON for DE.
MODULE_CLOCK_GATED_PE	2:2	Module level clock gating is ON for PE.
MODULE_CLOCK_GATED_SH	3:3	Module level clock gating is ON for SH.
MODULE_CLOCK_GATED_PA	4:4	Module level clock gating is ON for PA.
MODULE_CLOCK_GATED_SE	5:5	Module level clock gating is ON for SE.
MODULE_CLOCK_GATED_RA	6:6	Module level clock gating is ON for RA.
MODULE_CLOCK_GATED_TX	7:7	Module level clock gating is ON for TX.

14 Memory Controller Registers

All memory accesses that go to or from the AXI bus pass through the Memory Controller. It connects the Host Interface to the rest of the core. The Memory Controller's register set contains the controls for the memory interface, the settings for the virtual memory page table, and the values of the offset registers. It also contains a variety of debug registers that are set by other blocks within the core. Users can access all of the Memory Controller's registers via the AHB bus.

AQMemoryDebug

Description:

Reset Value = 3c000000 Address = 0x0105 Count = 1

Field Mask = 7FFA60FF ReadMask = 7FFA60FF Write Mask = 7FFA60FF

Sub-Field Name	*Bits*	*Description*
MAX_OUTSTANDING_READS	7:0	Limits the total number of outstanding read requests.
DISABLE_BUBBLE_OPTIMIZATION_IN_DECODER	13:13	
DISABLE_MINI_MMU_CACHE	14:14	
INTERLEAVE_BUFFER_LOW_LATENCY_MODE	17:17	
LIMIT_CONTROL 0 => REQUESTS 1 => DATA	19:19	
DISABLE_FAST_CLEAR	20:20	Reserved.
DISABLE_ZCOMPRESSION	21:21	Reserved.
DISABLE_STALL_READS	22:22	
DISABLE_WRITE_DATA_SPEEDUP	23:23	
ZCOMP_LIMIT	29:24	
DONT_STALL_WRITES_TO_SAME_ADDRESS	30:30	

AQRegisterTimingControl

Description: Bit 22 - Light sleep. Bit 21 - Deep sleep. Bit 20 - Powerdown memory. Bit 19:18 - WTC for fast rams. Bit 17:16 - RTC for fast rams.

Reset Value = 00030000 Address = 0x010B Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = FFFFFFFF

Sub-Field Name	*Bits*	*Description*
RESERVED	31:23	
LIGHT_SLEEP	22:22	
DEEP_SLEEP	21:21	
POWER_DOWN	20:20	
FAST_WTC	19:18	
FAST_RTC	17:16	
FOR_RF2P	15:8	
FOR_RF1P	7:0	

gcDbgCycleCounter

Description: Increments every cycle.

Reset Value = 00001c5e Address = 0x010E Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = FFFFFFFF

Sub-Field Name	Bits	Description
COUNT	31:0	

gcOutstandingReads0

Description: Number of outstanding reads per client in multiples of 8B.

Reset Value = 00000000 Address = 0x010F Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = 00000000

Sub-Field Name	Bits	Description
PEC	7:0	
PEZ	15:8	
FE	23:16	
MMU	31:24	

gcOutstandingReads1

Description: Number of outstanding reads per client in multiples of 8B.

Reset Value = 00000000 Address = 0x0110 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = 00000000

Sub-Field Name	Bits	Description
RA	7:0	
TX	15:8	
FC	23:16	
TOTAL	31:24	This field keeps the value of total read requests or total requested data (in 64bits) depending on the value of LimitControl field in AQMemoryDebug register.

gcOutstandingWrites

Description: Number of outstanding writes per client.

Reset Value = 00000000 Address = 0x0111 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = 00000000

Sub-Field Name	Bits	Description
PEC	7:0	
PEZ	15:8	
FC	23:16	
TOTAL	31:24	

gcregEndianness0

Description:

Reset Value = 00000000 Address = 0x0121 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = FFFFFFFF

Sub-Field Name	Bits	Description
WORD_SWAP	31:0	Flip the words of 32 bit data. 0x12345678 becomes 0x56781234

gcregEndianness1

Description:

Reset Value = 00000000 Address = 0x0122 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = FFFFFFFF

Sub-Field Name	Bits	Description
BYTE_SWAP	31:0	Flip the bytes of 16 bit data. 0x12345678 becomes 0x34127856

gcregEndianness2

Description:

Reset Value = 00000000 Address = 0x0123 Count = 1

Field Mask = FFFFFFFF ReadMask = FFFFFFFF Write Mask = FFFFFFFF

Sub-Field Name	Bits	Description
BIT_SWAP	31:0	Flip the bits of 8 bit data. 0x12345678 becomes 0x84c2a6e1

15 DMA Unit Registers

The DMA Unit (also known as the Fetch Engine) has only a few registers that the user can access via the AHB bus. The most important of these control the location and the fetching of the command stream. The register set also contains several debug registers. The rest of the DMA register set must be accessed by state load commands in software.

AQCmdBufferAddr

Description: Base address for the command buffer. The address must be 64-bit aligned and it is always physical. To use addresses above 0x8000_0000, program AQMemoryFE with the appropriate offset. Also, this register cannot be read. To check the value of the current fetch address use AQFEDebugCurCmdAdr. Since this is a write only register is has no reset value.

Reset Value = 00000000	Address = 0x0195	Count = 1
Field Mask = FFFFFFFF	ReadMask = 00000000	Write Mask = FFFFFFFC

Sub-Field Name	*Bits*	*Description*
TYPE	31:31	Programmers should always write 0 to this bit
0 => SYSTEM		
1 => VIRTUAL_SYSTEM		
ADDRESS	30:0	

AQCmdBufferCtrl

Description: Since this is a write only register is has no reset value.

Reset Value = 00000000	Address = 0x0196	Count = 1
Field Mask = 0031FFFF	ReadMask = 00310000	Write Mask = 0031FFFF

Sub-Field Name	*Bits*	*Description*
PREFETCH	15:0	Number of 64-bit words to fetch from the command buffer.
ENABLE	16:16	Enable the command parser.
0 => DISABLE		
1 => ENABLE		
ENDIAN_CONTROL	21:20	Control endian swapping.
0 => NO_SWAP		
1 => SWAP_WORD		
2 => SWAP_DWORD		

Document Revision History

Doc Revision	Date	Comments
1.0	2012-07-20	TI customer version, including AHB register information
0.2	2012-04-13	Additional detail added for Alignment requirements
0.1	2012-04-01	Preliminary draft