

**ZACH2**  
Programmer's Reference  
Manual

October 16, 2009

# Contents

<b>1</b>	<b>Control Registers</b>	<b>4</b>
1.1	GraphicsConfig . . . . .	4
1.2	GraphicsControl . . . . .	5
1.3	DMAAddress . . . . .	7
1.4	DMAUsed . . . . .	7
1.5	FIFOProcessing . . . . .	8
1.6	EnableInt . . . . .	8
1.7	FlagInt . . . . .	9
1.8	Reboot . . . . .	9
<b>2</b>	<b>Graphics Registers</b>	<b>10</b>
2.1	BlueDeltaX . . . . .	10
2.2	BlueDeltaYDom . . . . .	10
2.3	BlueZero . . . . .	10
2.4	RasterizerConfig . . . . .	11
2.5	SubContinue . . . . .	11
2.6	GreenDeltaX . . . . .	12
2.7	GreenDeltaYDom . . . . .	12
2.8	GreenZero . . . . .	13
2.9	Height . . . . .	13
2.10	RedDeltaX . . . . .	13
2.11	RedDeltaYDom . . . . .	14
2.12	RedZero . . . . .	14
2.13	RenderMode . . . . .	14
2.14	XDomDelta . . . . .	15
2.15	XDomZero . . . . .	15
2.16	XSubDelta . . . . .	16
2.17	XSubZero . . . . .	16
2.18	YDelta . . . . .	16
2.19	YZero . . . . .	17
2.20	ZDeltaXL . . . . .	17
2.21	ZDeltaXU . . . . .	17
2.22	ZDeltaYDomL . . . . .	18
2.23	ZDeltaYDomU . . . . .	18
2.24	ZZeroL . . . . .	18
2.25	ZZeroU . . . . .	19
<b>3</b>	<b>Video Registers</b>	<b>20</b>
3.1	BootAddress . . . . .	20
3.2	ClkCtlVideo . . . . .	20
3.3	MemoryConfig . . . . .	20
3.4	DMAControl . . . . .	21
3.5	FIFOControl . . . . .	22
3.6	MemoryControl . . . . .	23

3.7	LineCount . . . . .	23
3.8	DFIFODiscon . . . . .	23
3.9	FIFODiscon . . . . .	24
3.10	HorzBlankStop . . . . .	24
3.11	HorzOutBand . . . . .	25
3.12	HorzSyncStart . . . . .	25
3.13	HorzSyncStop . . . . .	25
3.14	HorzTotal . . . . .	26
3.15	IntLine . . . . .	26
3.16	Access1Mem . . . . .	26
3.17	Access2Mem . . . . .	27
3.18	ScanlineStride . . . . .	28
3.19	ScreenOrigin . . . . .	29
3.20	ScreenOriginRight . . . . .	29
3.21	VertBlankStop . . . . .	29
3.22	VertSyncStart . . . . .	30
3.23	VertSyncStop . . . . .	30
3.24	VertTotal . . . . .	30
3.25	DataView . . . . .	31
3.26	WriteMaskBypass . . . . .	32
<b>4</b>	<b>Video Unit</b>	<b>33</b>
4.1	Using the Video Unit . . . . .	33
4.2	Example Timing Values for 800x600 32BPP 75Hz . . . . .	34
4.3	Example Control Values for 800x600 32BPP 75Hz . . . . .	34
<b>5</b>	<b>Rendering a Gouraud Shaded Triangle</b>	<b>35</b>
5.1	A Gouraud Shaded Triangle . . . . .	35
5.2	Initialization . . . . .	36
5.3	Dominant and Subordinate Sides of a Triangle . . . . .	36
5.4	Calculating Color Values for Interpolation . . . . .	36
5.5	Register Set-up for Color Interpolation . . . . .	37
5.6	Register Set-up for Depth Testing . . . . .	37
5.7	Calculating the Slopes for each Side . . . . .	38
5.8	Rasterizer Mode . . . . .	38
5.9	Subpixel Correction . . . . .	39
5.10	Rasterization . . . . .	39
<b>6</b>	<b>Programming Interface</b>	<b>41</b>
6.1	Managing FIFO . . . . .	41
6.2	Using DMA . . . . .	42
6.3	DMA Buffers . . . . .	44
6.4	DMA Interrupts . . . . .	45

# 1 Control Registers

## 1.1 GraphicsConfig

**Description:** Chip Configuration

**Offset:** 0000 0008

**Reset Value:** From Configuration Data

**Access:** Read/Write



Bits 0-1: SClk Sel

- 0 – PClk
- 1 – PClk/2
- 2 – MClk
- 3 – MClk/2

Bit 7: Short Reset

- 0 – generate normal reset
- 1 – generate short reset

Bit 8: SBA Capable

- 0 – AGP sideband addressing disable
- 1 – AGP sideband addressing enable

Bit 9: AGP Capable

- 0 – Not AGP Capable
- 1 – AGP Capable

Bit 12: Sub System From ROM

- 0 – Leave subsystem registers at reset state
- 1 – Load subsystem registers from ROM immediately after reset.

Bit 20: Retry Disable

- 0 – enabled PCI Retry using "Disconnect-Without-Data"
- 1 – disabled PCI Retry using "Disconnect-Without-Data"

Bit 29: VGA Fixed

- 0 – disabled SVGA fixed address decoding
- 1 – enabled SVGA fixed address decoding

Bit 30: VGA Enable

- 0 – disabled internal SVGA subsystem
- 1 – enabled internal SVGA subsystem

Bit 31: Base Class Zero

- 0 – use the correct PCI Base Class Code
- 1 – force the PCI Base Class Code to be zero

## 1.2 GraphicsControl

**Description:** Video Control Settings

**Offset:** 0000 0010

**Reset Value:** From Configuration Data

**Access:** Read/Write



Bit 0: Data64Enable

- 0 Data output to RAMDAC as 32 bit units.
- 1 Data output to RAMDAC as 64 bit units.

Bit 1: GPPendingRight

- 0 – ScreenBaseRight value used.
- 1 – New ScreenBaseRight value waiting to be used.  
*Read only bit, set when ScreenBaseRight is loaded through the Graphics Processor.*

Bit 2: BypassPendingRight

- 0 – ScreenBaseRight value used.
- 1 – New ScreenBaseRight value waiting to be used.  
*Read only bit, set when ScreenBaseRight is loaded through the bypass.*

Bit 3: RightFrame

- 0 – Displaying left frame.

1 – Displaying right frame. *Read only bit.*

Bit 4: RightEyeCtl

0 – Active High

1 – Active Low

Bit 5: StereoEnable

0 – Disabled

1 – Enabled. *Not Implemented in this release.*

Bits 6-7: BufferSwapCtl

0 – SyncOnFrameBlank

1 – FreeRunning

2 – LimitToFrameRate

3 – Reserved

Bit 8: GPPending

0 – ScreenBase value used.

1 – New ScreenBase value waiting to be used.

*Read only bit, set when ScreenBase is loaded through the Graphics Processor.*

Bit 9: BypassPending

0 – ScreenBase value used.

1 – New ScreenBase value waiting to be used.

*Read only bit, set when ScreenBase is loaded through the bypass.*

Bits 10-11: VSyncCtl

0 – Forced High

1 – Active High

2 – Forced Low

3 – Active Low

Bits 12-13: HSyncCtl

0 – Forced High

1 – Active High

2 – Forced Low

3 – Active Low

Bit 14: LineDouble

- 0 – Line doubling disabled
- 1 – Line doubling enabled  
*If enabled, each scanline is displayed twice to increase the effective frequency of low resolution screens.*

Bit 15: BlankCtl

- 0 Active High
- 1 Active Low

Bit 16: Enable

- 0 – GP video disabled
- 1 – GP video enabled

Bits 17-31: Reserved

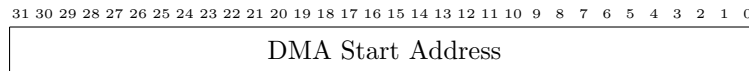
### 1.3 DMAAddress

**Description:** In DMA Start Address

**Offset:** 0000 00F8

**Reset Value:** 0

**Access:** Read/Write



The DMA address should be loaded with the first PCI address for the buffer to be transferred to the GC when using the DMA controller. Writing to the DMA count register loads the address into the DMA counter. Once a DMA has been set off the next DMA start address may be loaded.

### 1.4 DMAUsed

**Description:** In DMA Count

**Offset:** 0000 0100

**Reset Value:** 0

**Access:** Read/Write



The DMA count register should be loaded with the number of words to be transferred in the DMA operation. The action of loading a word count greater than zero sets off the DMA operation. The value read back from this register indicates the current number of words left to be transferred. This register should only be written to if the count is zero. It can be read at any time.

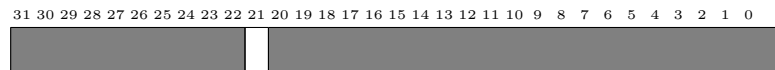
## 1.5 FIFOProcessing

**Description:** FIFO Processing Notification

**Offset:** 0000 0108

**Reset Value:** Read/Write

**Access:**



Bit 21: FIFO Activity

0 – Processing FIFO Commands

1 – No FIFO Activity

## 1.6 EnableInt

**Description:** Interrupt Enable Register

**Offset:** 0000 00E8

**Reset Value:** 0

**Access:** Read/Write



Bit 16: DMA interrupt enable

0 – Disable interrupt

1 – Enable interrupt

The Interrupt Enable Register allows for a DMA flag to generate a PCI interrupt. At reset the interrupt source is disabled.

## 1.7 FlagInt

**Description:** Interrupt Flag Register

**Offset:** 0000 00F0

**Reset Value:** 0

**Access:** Read/Write



Bit 16: DMA Interrupt Flag

0 – No Interrupt

1 – Interrupt Outstanding

The Interrupt Flags Register shows which interrupts are outstanding. Flag bits are reset by writing to this register with the corresponding bit set to a zero.

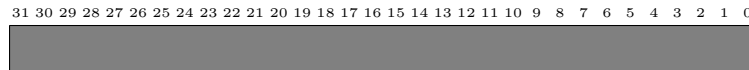
## 1.8 Reboot

**Description:** Reboot Graphics Card

**Offset:** 0000 0000

**Reset Value:** 0

**Access:** Write Only



Bits 0-31: Reserved

Writing to this address instructs the memory controller to reboot the SGRAMs. This involves going through the reset sequence and loading the Boot Address register. A re-boot does not reload the configuration data; registers maintain their contents until a reset. A read from this register returns zero.

## 2 Graphics Registers

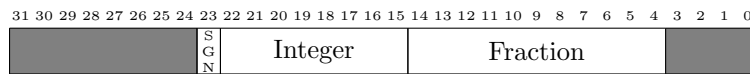
### 2.1 BlueDeltaX

**Description:** X Derivative - Blue

**Offset:** 0000 00D8

**Reset Value:** Undefined

**Access:** Read/Write



This register is used to set the X derivative for the Blue value for the interior of a trapezoid when Gouraud shading. The value is 2's complement 9.11 fixed point format.

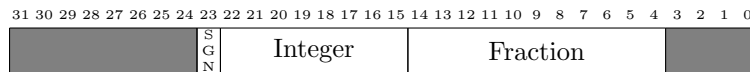
### 2.2 BlueDeltaYDom

**Description:** Y Derivative Dominant - Blue

**Offset:** 0000 0040

**Reset Value:** Undefined

**Access:** Read/Write



This register is used to set the Y derivative dominant for the Blue value along a line, or the dominant edge of a trapezoid when Gouraud shading. The value is 2's complement 9.11 fixed point format.

### 2.3 BlueZero

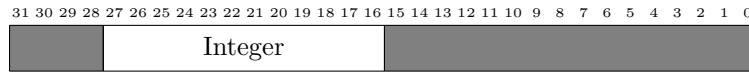
**Description:** Initial Blue Color

**Offset:** 0000 0028

**Reset Value:** Undefined

**Access:** Read/Write





This command causes rasterization to continue with a new subordinate edge. The subordinate DDA is reloaded with the new parameters. The dominant edge is carried on from the previous trapezoid. This is very useful when scan converting triangles with a 'knee' (i.e. two subordinate edges).

The data field holds the number of scanlines to fill. Note this count does not get loaded into the Height register

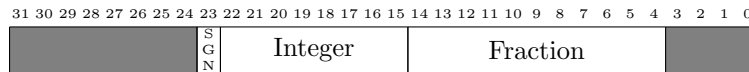
## 2.6 GreenDeltaX

**Description:** X Derivative - Green

**Offset:** 0000 00D0

**Reset Value:** Undefined

**Access:** Read/Write



This register is used to set the X derivative for the Green value for the interior of a trapezoid when Gouraud shading. The value is 2's complement 9.11 fixed point format.

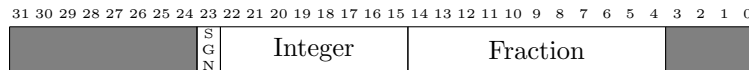
## 2.7 GreenDeltaYDom

**Description:** Y Derivative Dominant - Green

**Offset:** 0000 0038

**Reset Value:** Undefined

**Access:** Read/Write



This register is used to set the Y derivative dominant for the Green value along a line, or the dominant edge of a trapezoid when Gouraud shading. The value is 2's complement 9.11 fixed point format.

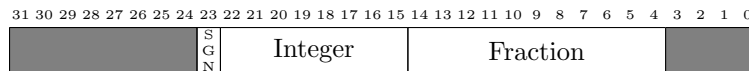
## 2.8 GreenZero

**Description:** Initial Green Color

**Offset:** 0000 0020

**Reset Value:** Undefined

**Access:** Read/Write



The register is used to set the initial value for the Green value for a vertex when in Gouraud shading mode. The value is 2's complement 9.11 fixed point format.

## 2.9 Height

**Description:** Count

**Offset:** 0000 00B0

**Reset Value:** Undefined

**Access:** Read/Write



It specifies the number of pixels in a line, or the number of scanlines in a trapezoid.

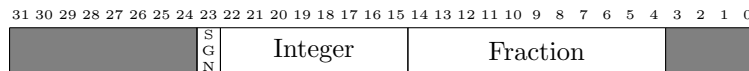
## 2.10 RedDeltaX

**Description:** X Derivative - Red

**Offset:** 0000 00C8

**Reset Value:** Undefined

**Access:** Read/Write



This register is used to set the X derivative for the Red value for the interior of a trapezoid when Gouraud shading. The value is 2's complement 9.11 fixed point format.

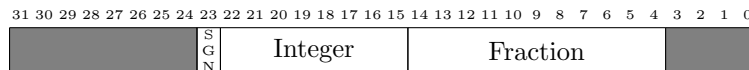
## 2.11 RedDeltaYDom

**Description:** Y Derivative Dominant - Red

**Offset:** 0000 0030

**Reset Value:** Undefined

**Access:** Read/Write



This register is used to set the Y derivative dominant for the Red value along a line, or the dominant edge of a trapezoid when Gouraud shading. The value is 2's complement 9.11 fixed point format.

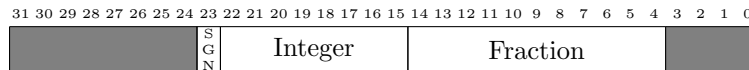
## 2.12 RedZero

**Description:** Initial Red Color

**Offset:** 0000 0018

**Reset Value:** Undefined

**Access:** Read/Write



The register is used to set the initial value for the Red value for a vertex when in Gouraud shading mode. The value is 2's complement 9.11 fixed point format.

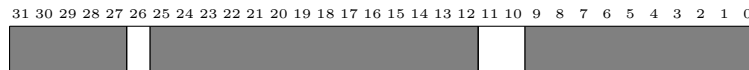
## 2.13 RenderMode

**Description:** Rendering Command

**Offset:** 0000 00B8

**Reset Value:** Undefined

**Access:** Write Only



Command to start the rendering process.

Bit 10-11: PrimitiveType. These bits indicate the type of ZACH2 primitive to be drawn. The primitives supported and the corresponding codes are:

- 0 – Reserved for future use.
- 1 – Screen-aligned trapezoids.
- 2 – Reserved for future use.
- 3 – Reserved for future use.

Bit 26: SubPixelCorrectionEnable. Enables the sub pixel correction of color, depth, fog and texture values at the start of a scanline span.

- 0 – Disable
- 1 – Enable

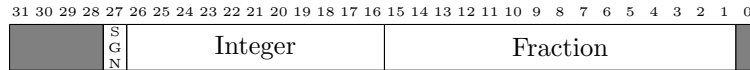
## 2.14 XDomDelta

**Description:** Delta of X Dominant Edge

**Offset:** 0000 0088

**Reset Value:** Undefined

**Access:** Read/Write



Value added when moving from one scanline to the next for the dominant edge in trapezoid filling. The value is in 2's complement 12.15 fixed point format. Also holds the change in X when plotting lines. For Y major lines this will be some fraction ( $dx/dy$ ), otherwise it is normally  $\pm 1.0$ , depending on the required scanning direction.

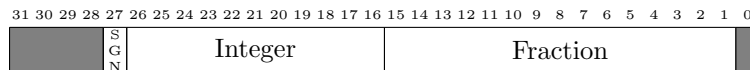
## 2.15 XDomZero

**Description:** Initial X Value of Dominant Edge

**Offset:** 0000 0080

**Reset Value:** Undefined

**Access:** Read/Write



Initial X value for the dominant edge in trapezoid filling, or initial X value in line drawing. The value is in 2's complement 12.15 fixed point format.

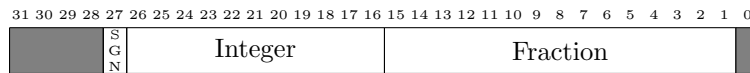
## 2.16 XSubDelta

**Description:** Delta of X Subordinate Edge

**Offset:** 0000 0098

**Reset Value:** Undefined

**Access:** Read/Write



Value added when moving from one scanline to the next for the subordinate edge in trapezoid filling. The value is in 2's complement 12.15 fixed point format.

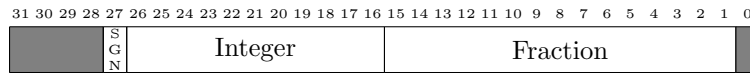
## 2.17 XSubZero

**Description:** Initial X Value of Subordinate Edge

**Offset:** 0000 0090

**Reset Value:** Undefined

**Access:** Read/Write



Initial X value for the subordinate edge in trapezoid filling. The value is in 2's complement 12.15 fixed point format.

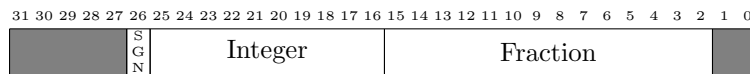
## 2.18 YDelta

**Description:** Y Delta

**Offset:** 0000 00A8

**Reset Value:** Undefined

**Access:** Read/Write



Value added to Y to move from one scanline to the next. For X major lines this will be some fraction ( $dy/dx$ ), otherwise it is normally  $\pm 1.0$ , depending on the required scanning direction. The value is in 2's complement 11.14 fixed point format.

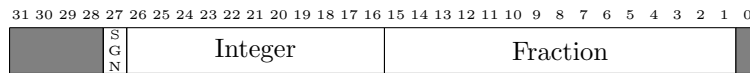
## 2.19 YZero

**Description:** Starting Y Value

**Offset:** 0000 00A0

**Reset Value:** Undefined

**Access:** Read/Write



Initial scanline in trapezoid filling, or initial Y position for line drawing. The value is in 2's complement 12.15 fixed point format.

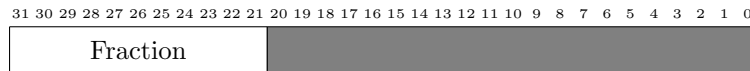
## 2.20 ZDeltaXL

**Description:** Lower part of dZdX

**Offset:** 0000 0070

**Reset Value:** Undefined

**Access:** Read/Write



This register holds part of the depth derivative per unit in  $X$  used in rendering trapezoids. ZDeltaXU holds the most significant bits, and ZDeltaXL the least significant bits. The combined value is in 2's complement 17.11 fixed point format.

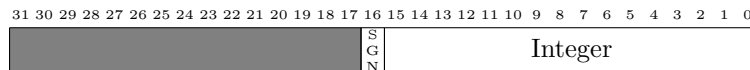
## 2.21 ZDeltaXU

**Description:** Upper part of dZdX

**Offset:** 0000 0068

**Reset Value:** Undefined

**Access:** Read/Write



This register holds part of the depth derivative per unit in  $X$  used in rendering trapezoids. ZDeltaXU holds the most significant bits, and ZDeltaXU the least

significant bits. The combined value is in 2's complement 17.11 fixed point format.

## 2.22 ZDeltaYDomL

**Description:** Depth Derivative Y Dominant - Lower

**Offset:** 0000 0060

**Reset Value:** Undefined

**Access:** Read/Write



This register holds part of the depth derivative per unit in Y used for the dominant edge of a trapezoid, or along a line. ZDeltaYDomU holds the most significant bits, and ZDeltaYDomL the least significant bits. The value is in 2's complement 17.11 fixed point format.

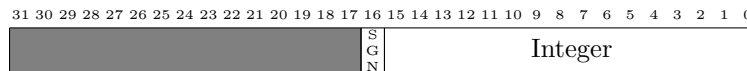
## 2.23 ZDeltaYDomU

**Description:** Depth Derivative Y Dominant - Upper

**Offset:** 0000 0058

**Reset Value:** Undefined

**Access:** Read/Write



This register holds part of the depth derivative per unit in Y used for the dominant edge of a trapezoid, or along a line. ZDeltaYDomU holds the most significant bits, and ZDeltaYDomL the least significant bits. The value is in 2's complement 17.11 fixed point format.

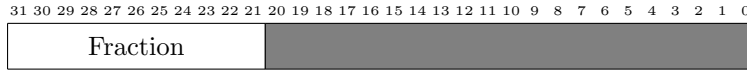
## 2.24 ZZeroL

**Description:** Depth Start Value - Lower

**Offset:** 0000 0050

**Reset Value:** Undefined

**Access:** Read/Write



This register holds part of the start value for depth interpolation. ZZeroU holds the most significant bits, and ZZeroL the least significant bits. The combined value is in 2's complement 17.11 fixed point format.

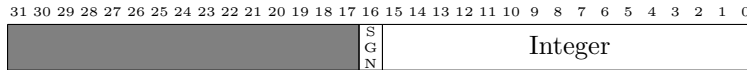
## 2.25 ZZeroU

**Description:** Depth Start Value - Upper

**Offset:** 0000 0048

**Reset Value:** Undefined

**Access:** Read/Write



This register holds part of the start value for depth interpolation. ZZeroU holds the most significant bits, and ZZeroL the least significant bits. The combined value is in 2's complement 17.11 fixed point format.

### 3 Video Registers

#### 3.1 BootAddress

**Description:** Boot Address

**Offset:** 0000 01F0

**Reset Value:** 0000 0020

**Access:** Read/Write



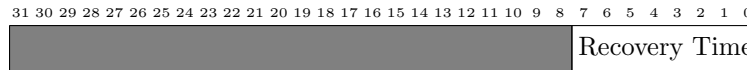
#### 3.2 ClkCtlVideo

**Description:** Clock Controller for Video Unit

**Offset:** 0000 01C8

**Reset Value:** Undefined

**Access:** Read/Write



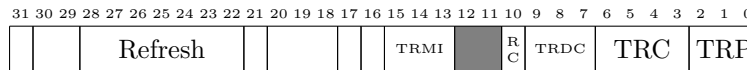
#### 3.3 MemoryConfig

**Description:** Configuration for Memory Options

**Offset:** 0000 01F8

**Reset Value:** E600 2021

**Access:** Read/Write



Bits 0-2: TimeRP

Bits 3-6: TimeRC

Bits 7-9: TimeRCD

Bit 10: RowCharge

Bits 13-15: TimeRASMin

Bit 16: CASLatency  
 Bit 17: DeadCycleEnable  
 Bits 18-20: BankDelay  
 Bit 21: Block Write 1  
 Bits 22-28: RefreshCount  
 Bits 29-30: NumberBanks  
 Bit 31: Burst1Cycle

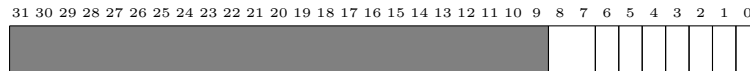
### 3.4 DMAControl

**Description:** DMA Configuration

**Offset:** 0000 01D0

**Reset Value:** Undefined

**Access:** Read/Write



Bit 0: In DMA Using AGP

- 0 – Input DMA uses PCI master
- 1 – Input DMA uses AGP master

Bit 1: In DMA Byte Swap

- 0 – Little endian
- 1 – Big endian

Bit 2: Long Read Disable

- 0 – Long reads allowed
- 1 – Long reads disabled

Bit 3: In DMA Data Throttle. *Applies to AGP transfers to GP input FIFO.*

- 0 – Control data flow using bus protocols
- 1 – Throttle data requests based on input FIFO space

Bit 4: AGP Data Throttle. *Applies to all AGP transfers.*

- 0 – Control data flow using bus protocols

1 – Throttle data requests based on FIFO space

Bit 5: Out DMA Byte Swap Control

0 – Little endian

1 – Big endian

Bit 6: AGP High Priority

0 – Use AGP low priority reads

1 – Use AGP high priority reads

Bits 7-8: Texture Execute Byte Swap

0 – Standard

1 – Read buffer selected by bit 31 of memory contents

2 – Half Word Swapped

3 – Reserved

### 3.5 FIFOControl

**Description:** Control values for FIFO

**Offset:** 0000 01A8

**Reset Value:** Undefined

**Access:** Read/Write



Bits 0-4: LowThreshold

Video data is accessed from memory at a low priority when there are this many or less spaces in the video FIFO.

Bits 11-15: HighThreshold

Video data is accessed from memory at a high priority when there are this many or less spaces in the video FIFO.

### 3.6 MemoryControl

**Description:** Memory Control Register

**Offset:** 0000 01E0

**Reset Value:** 0

**Access:** Read/Write



Bit 6: SDRAM

0 – SGRAM fitted

1 – SDRAM fitted

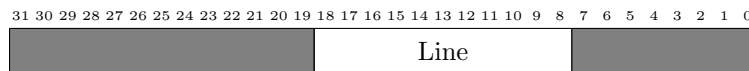
### 3.7 LineCount

**Description:** Current Line Count

**Offset:** 0000 01B0

**Reset Value:** Undefined

**Access:** Read/Write



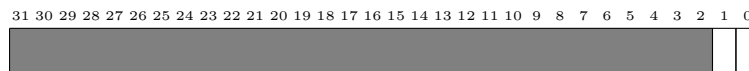
### 3.8 DFIFODiscon

**Description:** Enable/Disable Debug Disconnect FIFO Signals

**Offset:** 0000 0200

**Reset Value:** 0

**Access:** Read/Write



Bit 0: Output Debug FIFO Disconnect Enable

0 – Disabled.

1 – Enabled.

Bit 1: Input Debug FIFO Disconnect Enable

0 – Disabled.

1 – Enabled.

### 3.9 FIFODiscon

**Description:** Enable/Disable Disconnect FIFO Signals

**Offset:** 0000 01D8

**Reset Value:** 0

**Access:** Read/Write



Bit 0: Output FIFO Disconnect Enable

0 – Disabled.

1 – Enabled.

Bit 1: Input FIFO Disconnect Enable

0 – Disabled.

1 – Enabled.

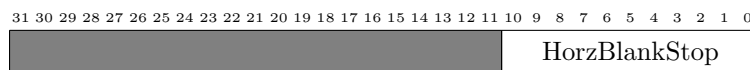
### 3.10 HorzBlankStop

**Description:** Horizontal Blank Stop

**Offset:** 0000 0140

**Reset Value:** Undefined

**Access:** Read/Write



### 3.11 HorzOutBand

**Description:** Horizontal Out of Band Timing

**Offset:** 0000 0138

**Reset Value:** Undefined

**Access:** Read/Write



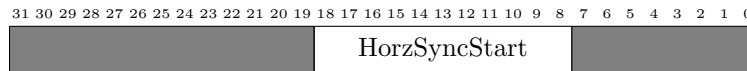
### 3.12 HorzSyncStart

**Description:** Horizontal Sync Start

**Offset:** 0000 0128

**Reset Value:** Undefined

**Access:** Read/Write



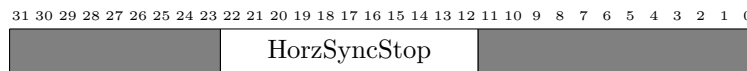
### 3.13 HorzSyncStop

**Description:** Horizontal Sync Stop

**Offset:** 0000 0130

**Reset Value:** Undefined

**Access:** Read/Write



### 3.14 HorzTotal

**Description:** Horizontal Total

**Offset:** 0000 0120

**Reset Value:** Undefined

**Access:** Read/Write



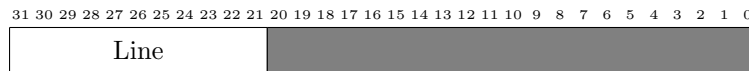
### 3.15 IntLine

**Description:** InterruptLine

**Offset:** 0000 0198

**Reset Value:** Undefined

**Access:** Read/Write



Bits 21-31: Line

Generate interrupt at start of this line.

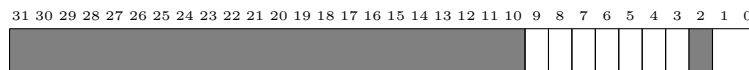
### 3.16 Access1Mem

**Description:** Memory Access One Control Register

**Offset:** 0000 0110

**Reset Value:** Undefined

**Access:** Read/Write



Bits 0-1: Memory Byte Control

0 – Standard.

1 – Byte Swapped.

2 – Half Word Swapped.

3 – Reserved.

Bit 3: Packed 16-bit (1:5:5:5) Memory Enable

- 0 – Disable packed 16-bit mode.
- 1 – Enable packed 16-bit mode.

Bit 4: Packed 16-bit Read Buffer Select

- 0 – Select Buffer A for Read Accesses.
- 1 – Select Buffer B for Read Accesses.

Bit 5: Packed 16-bit Write Buffer Select

- 0 – Select Buffer A for Write Accesses.
- 1 – Select Buffer B for Write Accesses.

Bit 6: Packed 16-bit Write Mode

- 0 – Disable double writes.
- 1 – Enable double writes.

Bit 7: Packed 16-bit Read Mode

- 0 – Read buffer selected by Bit 4 of this register.
- 1 – Read buffer selected by memory contents (bit 31).

Bit 8: SVGA Access

- 0 – Address memory controller directly.
- 1 – Address memory through SVGA subsystem.

Bit 9: ROM Access

- 0 – Use this aperture to access memory (SVGA or direct).
- 1 – Use this aperture to access the Expansion ROM.

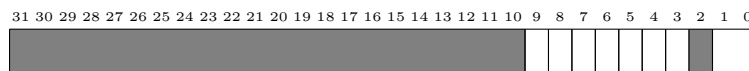
### 3.17 Access2Mem

**Description:** Memory Access Two Control Register

**Offset:** 0000 0118

**Reset Value:** Undefined

**Access:** Read/Write



Bits 0-1: Memory Byte Control

- 0 – Standard.
- 1 – Byte Swapped.
- 2 – Half Word Swapped.
- 3 – Reserved.

Bit 3: Packed 16-bit (1:5:5:5) Memory Enable

- 0 – Disable packed 16-bit mode.
- 1 – Enable packed 16-bit mode.

Bit 4: Packed 16-bit Read Buffer Select

- 0 – Select Buffer A for Read Accesses.
- 1 – Select Buffer B for Read Accesses.

Bit 5: Packed 16-bit Write Buffer Select

- 0 – Select Buffer A for Write Accesses.
- 1 – Select Buffer B for Write Accesses.

Bit 6: Packed 16-bit Write Mode

- 0 – Disable double writes.
- 1 – Enable double writes.

Bit 7: Packed 16-bit Read Mode

- 0 – Read buffer selected by Bit 4 of this register.
- 1 – Read buffer selected by memory contents (bit 31).

Bit 8: SVGA Access

- 0 – Address memory controller directly.
- 1 – Address memory through SVGA subsystem.

Bit 9: ROM Access

- 0 – Use this aperture to access memory (SVGA or direct).
- 1 – Use this aperture to access the Expansion ROM.

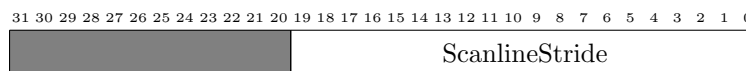
### 3.18 ScanlineStride

**Description:** Stride between scanlines of display

**Offset:** 0000 0178

**Reset Value:** Undefined

**Access:** Read/Write



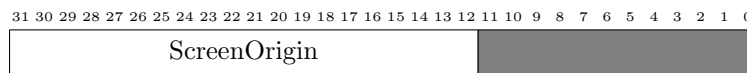
### 3.19 ScreenOrigin

**Description:** Screen Origin

**Offset:** 0000 01C0

**Reset Value:** Undefined

**Access:** Read/Write



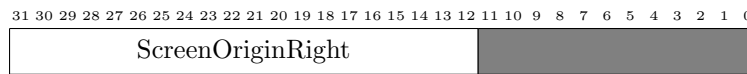
### 3.20 ScreenOriginRight

**Description:** Screen Origin Right

**Offset:** 0000 01B8

**Reset Value:** Undefined

**Access:** Read/Write



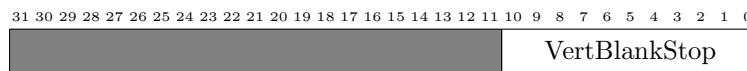
### 3.21 VertBlankStop

**Description:** Vertical Blank Stop

**Offset:** 0000 0160

**Reset Value:** Undefined

**Access:** Read/Write



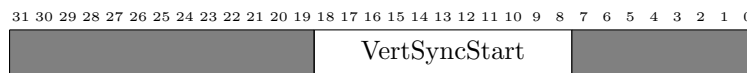
### 3.22 VertSyncStart

**Description:** Vertical Sync Start

**Offset:** 0000 0150

**Reset Value:** Undefined

**Access:** Read/Write



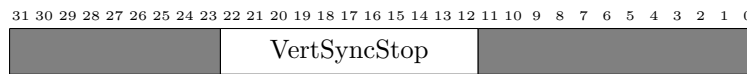
### 3.23 VertSyncStop

**Description:** Vertical Sync Stop

**Offset:** 0000 0158

**Reset Value:** Undefined

**Access:** Read/Write



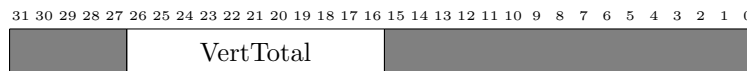
### 3.24 VertTotal

**Description:** Vertical Total

**Offset:** 0000 0148

**Reset Value:** Undefined

**Access:** Read/Write



### 3.25 DataView

**Description:** View Data Registers

**Offset:** 0000 01A0

**Reset Value:** Undefined

**Access:** Read/Write



Bit 0: DataIn

0 – Data is Low.

1 – Data is High.

Bit 1: ClkIn

0 – Clock is Low.

1 – Clock is High.

Bit 4: DataOut

0 – Drive Low.

1 – Drive Tri-state.

Bit 5: ClkOut

0 – Drive Low.

1 – Drive Tri-state.

Bit 8: LatchedData

0 – Data latched at 0.

1 – Data latched at 1.

Bit 12: DataValid

0 – DataIn not valid.

1 – DataIn valid.

Bit 16: Start

0 – DDC bus has not passed through Start state.

1 – DDC bus has passed through Start state.

Bit 20: Stop

- 0 – DDC bus has not passed through Stop state.
- 1 – DDC bus has passed through Stop state.

Bit 24: Wait

- 0 – Do not insert wait states in DDC.
- 1 – Insert wait states.

Bit 28: UseMonitorID

- 0 – Use DDC.
- 1 – Use monitor ID.

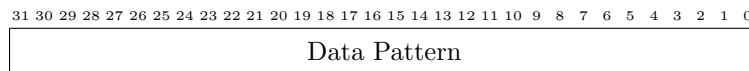
### 3.26 WriteMaskBypass

**Description:** Write protect bits from modification

**Offset:** 0000 01E8

**Reset Value:** Undefined

**Access:** Read/Write



Bits 0-31: Data Pattern

- Bit set to 0 = corresponding bit in memory protected
- Bit set to 1 = corresponding bit in memory writable

## 4 Video Unit

The video unit should be configured to display the framebuffer data with the format, resolution, and refresh frequency required.

### 4.1 Using the Video Unit

The diagram below shows the parameters that are used to control the display of images generated by the graphics processor. Any images generated by the SVGA unit are displayed by the SVGA which should be programmed in accordance with normal SVGA practice

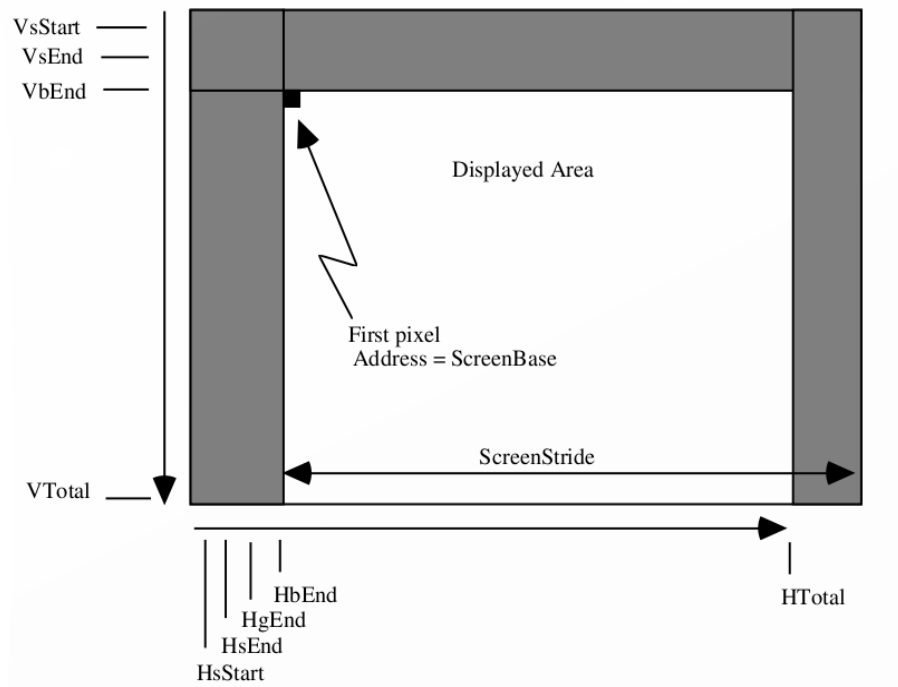


Figure 1: Video Timing Parameters

#### 4.2 Example Timing Values for 800x600 32BPP 75Hz

Register	Hex Value
HorzTotal	041F0000
HorzSyncStart	00001000
HorzSyncStop	00060000
HorzBlankStop	00000100
HorzOutBand	00001000
VertTotal	02700000
VertSyncStart	00000000
VertSyncStop	00003000
VertBlankStop	00000019
ScreenOrigin	00000000
ScreenOriginRight	00000000
GraphicsControl	00011401

#### 4.3 Example Control Values for 800x600 32BPP 75Hz

Register	Hex Value
Access1Mem	00000000
Access2Mem	00000000
IntLine	00000000
FIFOControl	00008010
LineCount	00000000
ClkCtlVideo	00000020
DMAControl	00000000
FIFODiscon	00000010
MemoryControl	00000000
MemoryConfig	E6002021
BootAddress	0C600000
DataView	00000000
WriteMaskBypass	FFFFFFFF

## 5 Rendering a Gouraud Shaded Triangle

In this section we show how to render a typical 3D graphics primitive. The primitive is a Gouraud shaded triangle. For this example, assume the coordinate origin is bottom left of the window and drawing will be from top to bottom.

### 5.1 A Gouraud Shaded Triangle

Consider a triangle with vertices,  $v_1$ ,  $v_2$  and  $v_3$  where each vertex comprises X, Y and Z coordinates, shown below. Each vertex has a different color made up of red, green and blue (R, G and B) components.

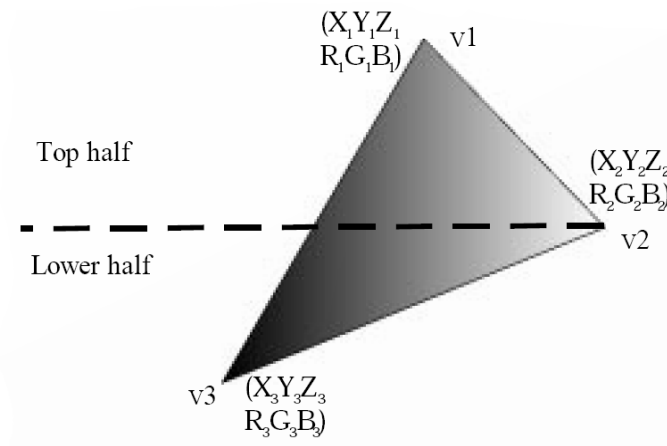


Figure 2: Example Triangle

The diagram makes a distinction between top and bottom halves because ZACH2 is designed to rasterize screen aligned trapezoids and flat topped or bottomed triangles as shown below:



Figure 3: Screen aligned trapezoid and flat topped triangle

## 5.2 Initialization

ZACH2 requires certain registers to be initialized in a particular way, regardless of what is to be drawn; for instance, the screen size and appropriate clipping must be set-up. Normally this only needs to be done once and for clarity this example assumes that all initialization has already been done.

## 5.3 Dominant and Subordinate Sides of a Triangle

The dominant side of a triangle is that with the greatest range of Y values. The choice of dominant side is optional when the triangle is either flat bottomed or flat topped. ZACH2 always draws triangles starting from the dominant edge towards the subordinate edges. This simplifies the calculation of set-up parameters as will be seen below.

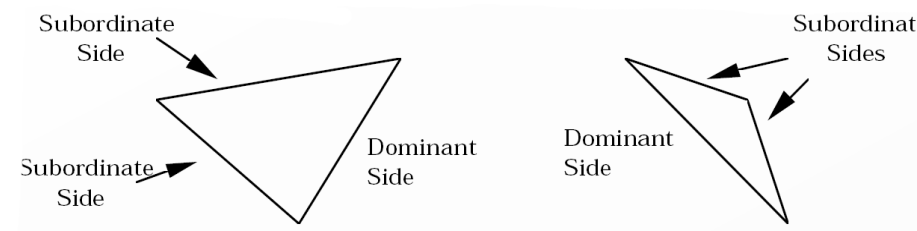


Figure 4: Dominant and Subordinate Sides of a Triangle

## 5.4 Calculating Color Values for Interpolation

To draw from left to right and top to bottom, the color gradients (or deltas) required are:

$$dRdy_{13} = \frac{R_3 - R_1}{Y_3 - Y_1}, \quad dGdy_{13} = \frac{G_3 - G_1}{Y_3 - Y_1}, \quad dBdy_{13} = \frac{B_3 - B_1}{Y_3 - Y_1}$$

And from the plane equation:

$$dRdx = \left( (R_1 - R_3) \times \frac{Y_2 - Y_3}{\alpha} \right) - \left( (R_2 - R_3) \times \frac{Y_1 - Y_3}{\alpha} \right)$$

$$dGdx = \left( (G_1 - G_3) \times \frac{Y_2 - Y_3}{\alpha} \right) - \left( (G_2 - G_3) \times \frac{Y_1 - Y_3}{\alpha} \right)$$

$$dBdx = \left( (B_1 - B_3) \times \frac{Y_2 - Y_3}{\alpha} \right) - \left( (B_2 - B_3) \times \frac{Y_1 - Y_3}{\alpha} \right)$$

where:

$$\alpha = |((X_1 - X_3) \times (Y_2 - Y_3)) - ((X_2 - X_3) \times (Y_1 - Y_3))|$$

These values allow the color of each fragment in the triangle to be determined by linear interpolation. For example, the red component color value of a fragment at  $X_n, Y_m$  could be calculated by:

- adding  $dRdy_{13}$ , for each scanline between  $Y_1$  and  $Y_n$ , to  $R_1$ .
- then adding  $dRdx$  for each fragment along scanline  $Y_n$  from the left edge to  $X_n$ .

The example chosen has the 'knee' i.e. vertex 2, on the right hand side, and drawing is from left to right. If the knee were on the left side (or drawing was from right to left), then the Y deltas for both the subordinate sides would be needed to interpolate the start values for each color component (and the depth value) on each scanline. For this reason the video card always draws triangles starting from the dominant edge and towards the subordinate edges. For the example triangle, this means left to right.

## 5.5 Register Set-up for Color Interpolation

For the example triangle the registers must be set as follows. Details of register formats are given later.

```
// Load the color start and delta values to draw a triangle
RedZero   (R1)
GreenZero (G1)
BlueZero  (B1)
// To walk up the dominant edge
RedDeltaYDom (dRdy13)
GreenDeltaYDom (dGdy13)
BlueDeltaYDom (dBdy13)
// To walk along the scanline
RedDeltaX (dRdx)
GreenDeltaX (dGdx)
BlueDeltaX (dBdx)
```

## 5.6 Register Set-up for Depth Testing

Internally ZACH2 uses fixed point arithmetic. The formats for each register are described later. Each depth value must be converted into a 2's complement fixed point number and then loaded into the appropriate pair of registers. The 'Upper' or 'U' registers store the integer portion, whilst the 'Lower' or 'L' registers store the fractional bits, left justified and zero filled. For the example triangle, depth buffering is disabled and ZACH2 would need its registers set-up as follows:

```
// Load the depth start and delta values
// to draw a triangle
```

```

ZZeroU (0)
ZZeroL (0)
ZDeltaYDomU (0)
ZDeltaYDomL (0)
ZDeltaXU (0)
ZDeltaXL (0)

```

## 5.7 Calculating the Slopes for each Side

ZACH2 draws filled shapes such as triangles as a series of spans with one span per scanline. Therefore it needs to know the start and end X coordinate of each span. These are determined by 'edge walking'. This process involves adding one delta value to the previous span's start X coordinate and another delta value to the previous span's end X coordinate to determine the X coordinates of the new span. These delta values are in effect the slopes of the triangle sides. To draw from left to right and top to bottom, the slopes of the three sides are calculated as:

$$dX_{13} = \frac{X_3 - X_1}{Y_3 - Y_1}, \quad dX_{12} = \frac{X_2 - X_1}{Y_2 - Y_1}, \quad dX_{23} = \frac{X_3 - X_2}{Y_3 - Y_2}$$

This triangle will be drawn in two parts, top down to the 'knee' i.e. vertex 2 and then from there to the bottom. The dominant side is the left side so for the top half:

$$dX_{Dom} = dX_{13}, \quad dX_{Sub} = dX_{12}$$

The start X,Y, the number of scanlines, and the above deltas give ZACH2 enough information to edge walk the top half of the triangle. However, to indicate that this is not a flat topped triangle, the same start position in terms of X must be given twice as XDomZero and XSubZero. To edge walk the lower half of the triangle, selected additional information is required. The slope of the dominant edge remains unchanged, but the subordinate edge slope needs to be set to:

$$dX_{Sub} = dX_{23}$$

Also the number of scanlines to be covered from  $Y_2$  to  $Y_3$  needs to be given. Finally to avoid any rounding errors accumulated in edge walking to  $X_2$  (which can lead to pixel errors), XSubZero must be set to  $X_2$ .

## 5.8 Rasterizer Mode

The ZACH2 Rasterizer has a number of modes which remain effective from the time they are set until they are modified and can thus affect many primitives. In the case of the Gouraud shaded triangle, the default values for these modes are suitable.

```
RasterizerConfig (0) // Default Rasterizer mode
```

## 5.9 Subpixel Correction

ZACH2 can perform subpixel correction of all interpolated values when rendering aliased trapezoids. This correction ensures that any parameter is correctly sampled at the center of a fragment. In general, subpixel correction will always be enabled when rendering any trapezoid which has interpolated parameters. Control of subpixel correction is in the RenderMode command register described in the next section, and is selectable on a per primitive basis. It does not need to be enabled for any primitive that does not use interpolation, including copy operations. If it is disabled and interpolators are used, the values calculated for the primitive may not be exactly correct; enabling sub-pixel correction may reduce the performance of the chip, particularly for small primitives.

## 5.10 Rasterization

ZACH2 is almost ready to draw the triangle. Setting up the registers as described here and sending the RenderMode command will cause the top half of the example triangle to be drawn.

For drawing the example triangle, all the bit fields within the RenderMode command should be set to 0 except the PrimitiveType which should be set to trapezoid and the SubPixelCorrectionEnable bit which should be set to TRUE.

```
// Draw triangle with knee
// Set deltas
XDomZero (X1)
XDomDelta (dX13)
XSubZero (X1)
XSubDelta (dX12)
YZero (Y1)
YDelta (1)
Height (Y1 - Y2)

// Draw the top half of the triangle
RenderMode (render)
```

After the RenderMode command has been issued, the registers can immediately be altered to draw the lower half of the triangle. Note that only two registers need be loaded and the command SubContinue sent. Once ZACH2 has received SubContinue, drawing of this sub-triangle will begin.

```
// Set-up the delta and start for the new edge
XSubZero (X2)
XSubDelta (dX23)

// Draw sub-triangle
SubContinue (Y2 - Y3) // Draw lower half
```



## 6 Programming Interface

### 6.1 Managing FIFO

When a data value is written to a register, this value and the address for that register are combined and put into the FIFO Queue as a new entry. The FIFOProcessing register is not updated until ZACH2 processes this entry. In the case where ZACH2 is busy performing a time consuming operation, and not draining the FIFO very quickly, it is possible for the FIFO to become full. If a write to a register is performed when the FIFO is full no entry is put into the FIFO and that write is effectively lost. If two writes to a register are performed without waiting for the FIFO Queue to drain, it is possible that the data value will be corrupted.

The FIFOProcessing register can be read to determine if the FIFO Queue is actively being processed. A pseudocode example of loading ZACH2 registers and sing the FIFOProcessing register from user space and kernel space is given below.

#### In user space:

```
...
//Send commands to graphics card

//Wait for FIFO Queue to drain

while( *FIFOProcessing != 0x00200000);

//Send more commands
...
```

#### In kernel space:

```
...
//Send commands to graphics card

//Wait for FIFO Queue to drain
while( *FIFOProcessing != 0x00200000) {
    schedule();
}

//Send more commands
...
```

The FIFOProcessing register contains a flag that when 1 means the there is FIFO commands being processed and 0 when there is no processing. Thus, whenever you want to make sure it is safe to issue FIFO commands, you should loop until the flag in the register is 0.

Notice the difference in the while statement between user and kernel space. Since it is possible that it may take a long time to process the commands pending in the FIFO queue. It is recommended that once you determine the queue is not empty, that you schedule yourself off the process if the kernel has other tasks pending.

## 6.2 Using DMA

Loading registers directly via the FIFO is often an inefficient way to download data to ZACH2. Given that the FIFO can accommodate only a small number of entries, ZACH2 has to be frequently interrogated to determine how much space is left. Also, consider the situation where a given API function requires a large amount of data to be sent to ZACH2. If the FIFO is written directly then a return from this function is not possible until almost all the data has been consumed. This may take some time depending on the types of primitives being drawn.

To avoid these problems ZACH2 provides an on-chip DMA controller which can be used to load data from arbitrary sized (< 64K 32-bit words) host buffers into the FIFO. In its simplest form the host software has to prepare a host buffer containing register address tag descriptions and data values. It then writes the base address of this buffer to the DMAAddress register and the count of the number of words to transfer to the DMAUsed register. Writing to the DMAUsed register starts the DMA transfer and the host can now perform other work. In general, if the complete set of rendering commands required by a given call to a driver function can be loaded into a single DMA buffer then the driver function can return. Meanwhile, in parallel, ZACH2 is reading data from the host buffer and loading it into its FIFO. FIFO overflow never occurs since the DMA controller automatically waits until there is room in the FIFO before doing any transfers.

The only restriction on the use of DMA control registers is that before attempting to reload the DMAAddress and DMAUsed register the host software must wait until previous DMA has completed. It is important to call FIFOSYNC (in user code) to flush the FIFO queue before initiating the first DMA transfer. However, calling FIFOSYNC before every DMA transfer negates the benefits of advanced DMA handling (discussed below) and may produce unpredictable behavior on the card. Many display driver functions can be implemented using the following skeleton structure:

```

//get a free DMA buffer and mark as in use
//do any pre-work
//copy render data into DMA buffer

DMAAddress(address of dma_buffer)
DMAUsed(number of words in DMA buffer)

return

```

Using DMA leaves the host free to return to the application, while in parallel, ZACH2 is performing the DMA and drawing. This can increase performance significantly over loading a FIFO directly. In addition, some algorithms require that data be loaded multiple times (e.g. drawing the same object across multiple clipping rectangles). Since the ZACH2 DMA only reads the buffer data, it can be downloaded many times simply by restarting the DMA. This can be very beneficial if composing the buffer data is a time consuming task.

A further optimization is to use a double buffered mechanism with two DMA buffers. This allows the second buffer to be filled before waiting for the previous DMA to complete thus further improving the parallelism between host and ZACH2 processing.

```

//A DMA buffer is already processing...
//do any pre-work
//get a another free DMA buffer and mark as in use
//put render data into this new buffer

while (*DMAUsed != 0)

DMAAddress(address of new buffer)

DMAUsed(number of words in new buffer)

//mark the old buffer as free

return

```

In general the DMA buffer format consists of a 32-bit address tag description word followed by one data word. The DMA buffer consists of one or more sets of this formats.

### 6.3 DMA Buffers

When DMA is performed each 32-bit tag description in the DMA buffer. The tag is the address offset of a register divided by 4. The address offset for each register is given in this manual.

The following pseudo-code shows the previous example of drawing a series of rectangles but this time with filling the buffer with commands. This example uses a single DMA buffer. For an indepth guide on rendering, see Section 5. *Note: This example does not properly format the data values to fit within the registers.*

```
UINT32 *pbuf

while (*DMAUsed != 0)          // wait for DMA to complete

DMAAddress (physical address of dma_buffer)

pbuf = dma_buffer
*pbuff++ = &XDomDelta>>2
*pbuff++ = 0
*pbuff++ = &XSubDelta>>2
*pbuff++ = 0
*pbuff++ = &YDelta>>2
*pbuff++ = 1
for (i = 0; i < nrects; ++i) {
    *pbuff++ = &XDomInit>>2
    *pbuff++ = rect->x1 // Start dominant edge
    *pbuff++ = &XSubInit>>2
    *pbuff++ = rect->x2 // Start of subordinate edge
    *pbuff++ = &Height>>2
    *pbuff++ = rect->y2 - rect->y1
    *pbuff++ = &YInit>>2
    *pbuff++ = rect->y1
    *pbuff++ = &RenderMode>>2
    *pbuff++ = ZACH2_TRAPEZOID_SPC
}

DMAUsed((int)(pbuf - dma_buffer))
return
```

The example assumes that a host buffer has been previously allocated and is pointed at by dma\_buffer.

Host software must generate the correct DMA buffer address for the ZACH2 DMA controller. Normally, this means that the address passed to ZACH2 must

be the physical address of the DMA buffer in host memory. The buffer must also reside at contiguous physical addresses as accessed by ZACH2. On a system which uses virtual memory for the address space of a task, some method of allocating contiguous physical memory, and mapping this into the address space of a task, must be used.

If the virtual memory buffer maps to non-contiguous physical memory then the buffer must be divided into sets of contiguous physical memory pages and each of these sets transferred separately. In such a situation the whole DMA buffer cannot be transferred in one go; the host software must wait for each set to be transferred. Often the best way to handle these fragmented transfers is via an interrupt handler.

## 6.4 DMA Interrupts

ZACH2 provides interrupt support, as an alternative means of determining when a DMA transfer is complete. This can provide considerable speed advantage. If enabled, the interrupt is generated whenever the DMAUsed register changes from having a non-zero to having a zero value. This happens when the last data item is transferred from the DMA buffer.

To enable the DMA interrupt, the DMA Interrupt Enable bit must be set in the EnableInt register. The interrupt handler should check the DMAFlag bit in the IntFlags register to determine that a DMA interrupt has actually occurred. To clear the interrupt a word should be written to the IntFlags register with the DMAFlag bit set to one.

A typical use of DMA interrupts might be as follows:

```
//prepare DMA buffer

DMAUsed(n);          // start a DMA transfer

//prepare next DMA buffer

while (*DMAUsed != 0) {
    //mask interrupts
    //set DMA Interrupt Enable bit in IntEnable register
    //sleep on interrupt handler wake up
    //unmask interrupts
}

DMAUsed(n)          // start the next DMA sequen
```

The interrupt handler could then be

```
if (*FlagsInt & DMA Flag bit) {  
    //reset DMA Flag bit in IntFlags  
    //send wake up to main task  
}
```

Interrupts are complicated and depend on the facilities provided by the host operating system. The above pseudocode only hints at the system details.

This scheme frees the processor for other work while DMA is being completed. Since the overhead of handling an interrupt is often quite high for the host processor, the scheme should be tuned to allow a period of polling before sleeping on the interrupt.