ZACH4
Programmer’s Reference
Manual

August 17, 2010
Contents

1 Control Registers 4
  1.1 CfgGraphics ................................................. 4
  1.2 CtrlGraphics .............................................. 5
  1.3 DMABufAddress ............................................. 7
  1.4 DMABufCount ................................................ 7
  1.5 FIFOSpaceFree ............................................. 8
  1.6 EnableInt .................................................. 8
  1.7 FlagsInt .................................................... 9
  1.8 CardReboot ................................................ 9

2 Graphics Registers 10
  2.1 dBdx .......................................................... 10
  2.2 dBdyDom ...................................................... 10
  2.3 B0 ............................................................. 10
  2.4 RasterizerCfg ............................................... 11
  2.5 ExtTriSub .................................................... 11
  2.6 dGdx .......................................................... 12
  2.7 dGdyDom ...................................................... 12
  2.8 G0 ............................................................ 13
  2.9 Count ........................................................ 13
  2.10 dRdx .......................................................... 13
  2.11 dRdyDom ...................................................... 14
  2.12 R0 ............................................................. 14
  2.13 Render ....................................................... 14
  2.14 dXDom ........................................................ 15
  2.15 XDom0 ........................................................ 15
  2.16 dXSub ........................................................ 16
  2.17 XSub0 ........................................................ 16
  2.18 dY ............................................................. 16
  2.19 Y0 ............................................................. 17
  2.20 dZdxL .......................................................... 17
  2.21 dZdxU .......................................................... 17
  2.22 dZdyDomL ...................................................... 18
  2.23 dZdyDomU ...................................................... 18
  2.24 Z0L ........................................................... 18
  2.25 Z0U ........................................................... 19

3 Video Registers 20
  3.1 BootAddress .................................................. 20
  3.2 VClkCtl ....................................................... 20
  3.3 MemoryCfg ................................................... 20
  3.4 DMACtrl ....................................................... 21
  3.5 FIFOCtrl ...................................................... 22
  3.6 MemoryCtrl .................................................. 23
3.7 LineCount ................. 23
3.8 DisconDFIFO ............... 23
3.9 DisconFIFO ................ 24
3.10 HbEnd ..................... 24
3.11 HgEnd ..................... 25
3.12 HsStart .................... 25
3.13 HsEnd ..................... 25
3.14 HTotal ..................... 25
3.15 InterruptLine .............. 26
3.16 Aperture1 .................. 26
3.17 Aperture2 .................. 27
3.18 ScreenStride .............. 28
3.19 ScreenBase .............. 29
3.20 ScreenBaseRight .......... 29
3.21 VbEnd ..................... 29
3.22 VsStart .................... 29
3.23 VsEnd ..................... 30
3.24 VTotal ..................... 30
3.25 DisplayData .............. 30
3.26 WriteMaskBypass .......... 32

4 Video Unit 33
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

5 Rendering a Gouraud Shaded Triangle 35
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 Configuration .... 38
5.9 Subpixel Correction .... 39
5.10 Rasterization .... 39

6 Programming Interface 41
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 CfgGraphics

**Description:** Chip Configuration

**Offset:** 0000 0008

**Reboot Value:** From Configuration Data

**Access:** Read/Write

<table>
<thead>
<tr>
<th>Bits 30-31: SClk Sel</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – PClk</td>
</tr>
<tr>
<td>1 – PClk/2</td>
</tr>
<tr>
<td>2 – MClk</td>
</tr>
<tr>
<td>3 – MClk/2</td>
</tr>
</tbody>
</table>

Bit 20: Short CardReboot

| 0 – generate normal reset     |
| 1 – generate short reset     |

Bit 19: SBA Capable

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

Bit 18: AGP Capable

| 0 – Not AGP Capable       |
| 1 – AGP Capable          |

Bit 17: Sub System From ROM

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

Bit 16: Retry Disable

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

Bit 2: VGA Fixed
0 – disabled SVGA fixed address decoding
1 – enabled SVGA fixed address decoding

Bit 1: VGA Enable
0 – disabled internal SVGA subsystem
1 – enabled internal SVGA subsystem

Bit 0: Base Class Zero
0 – use the correct PCI Base Class Code
1 – force the PCI Base Class Code to be zero

1.2 CtrlGraphics

Description: Video Control Settings
Offset: 0000 0010
Reboot Value: From Configuration Data
Access: Read/Write

Bit 31: Data64Enable
0 Data output to RAMDAC as 32 bit units.
1 Data output to RAMDAC as 64 bit units.

Bit 30: 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 29: 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 28: RightFrame
0 – Displaying left frame.
1 – Displaying right frame. *Read only bit.*

Bit 27: RightEyeCtl
- 0 – Active High
- 1 – Active Low

Bit 26: StereoEnable
- 0 – Disabled
- 1 – Enabled. *Not Implemented in this release.*

Bits 24-25: BufferSwapCtl
- 0 – SyncOnFrameBlank
- 1 – FreeRunning
- 2 – LimitToFrameRate
- 3 – Reserved

Bit 21: 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 20: 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 18-19: VSyncCtl
- 0 – Active High
- 1 – Forced High
- 2 – Active Low
- 3 – Forced Low

Bits 16-17: HSyncCtl
- 0 – Forced High
- 1 – Active High
- 2 – Forced Low
- 3 – Active Low

Bit 15: 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 14: BlankCtl

0 Active High
1 Active Low

Bit 13: Enable

0 – GP video disabled
1 – GP video enabled

### 1.3 DMABufAddress

**Description:** In DMA Start Address

**Offset:** 0000 00F8

**Reboot 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 DMABufCount

**Description:** In DMA Count

**Offset:** 0000 0100

**Reboot 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 FIFO SpaceFree

**Description:** FIFO Processing Notification

**Offset:** 0000 0108

**Reboot Value:** Undefined

**Access:** Read/Write

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Bit 15: FIFO Activity
| 0 – Processing FIFO Commands
| 1 – No FIFO Activity

1.6 EnableInt

**Description:** Interrupt Enable Register

**Offset:** 0000 00E8

**Reboot Value:** 0

**Access:** Read/Write

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Bit 6: 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 FlagsInt

Description: Interrupt Flag Register
Offset: 0000 00F0
Reboot Value: 0
Access: Read/Write

Bit 6: 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 CardReboot

Description: CardReboot Graphics Card
Offset: 0000 0000
Reboot 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 dBdx

Description: X Derivative - Blue
Offset: 0000 00D8
Reboot 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 dBdyDom

Description: Y Derivative Dominant - Blue
Offset: 0000 0040
Reboot 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 B0

Description: Initial Blue Color
Offset: 0000 0028
Reboot Value: Undefined
Access: Read/Write
The register is used to set the initial value for the Blue value for a vertex when in Gouraud shading mode. The value is 2's complement 9.11 fixed point format.

2.4 RasterizerCfg

Description: Rasterizer Configuration

Offset: 0000 0078

Reboot Value: Undefined

Access: Read/Write

Bit 31: MirrorBitMask
0 – use bit mask from least to most significant bit
1 – use bit mask from most to least significant bit

Bit 30: InvertBitMask
0 – test against inverted bitmask
1 – test against bitmask

Bits 28-29: FractionAdjust These bits are for the ExtTriSub command and specify how the fraction bits in the Y and XDom DDAs are adjusted.
0 – No adjustment is done,
1 – Set the fraction bits to zero,
2 – Set the fraction bits to half.
3 – Set the fraction to nearly half, i.e. 0x7FFF

2.5 ExtTriSub

Description: Continue Triangle with new subordinate edge

Offset: 0000 00C0

Reboot Value: Undefined

Access: Read/Write

11
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 Count register.

2.6  \textit{dGdx}

\textbf{Description:} X Derivative - Green

\textbf{Offset:} 0000 00D0

\textbf{Reboot Value:} Undefined

\textbf{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  \textit{dGdyDom}

\textbf{Description:} Y Derivative Dominant - Green

\textbf{Offset:} 0000 0038

\textbf{Reboot Value:} Undefined

\textbf{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  G0

Description: Initial Green Color
Offset: 0000 0020
Reboot 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  Count

Description: Count
Offset: 0000 00B0
Reboot Value: Undefined
Access: Read/Write

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

2.10  dRdx

Description: X Derivative - Red
Offset: 0000 00C8
Reboot 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  dRdyDom

Description:  Y Derivative Dominant - Red
Offset:  0000 0030
Reboot 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  R0

Description:  Initial Red Color
Offset:  0000 0018
Reboot 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  Render

Description:  Rendering Command
Offset:  0000 00B8
Reboot Value:  Undefined
Access:  Write Only

Command to start the rendering process.
Bit 21: SubPixelCorrectionEnable. Enables the sub pixel correction of color, depth, fog and texture values at the start of a scanline span.

0 – Disable
1 – Enable

Bit 10-11: PrimitiveType. These bits indicate the type of ZACH4 primitive to be drawn. The primitives supported and the corresponding codes are:

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

2.14 dXDom

Description: Delta of X Dominant Edge

Offset: 0000 0088

Reboot 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 ±1.0, depending on the required scanning direction.

2.15 XDom0

Description: Initial X Value of Dominant Edge

Offset: 0000 0080

Reboot 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  dXSub

Description: Delta of X Subordinate Edge
Offset: 0000 0098
Reboot 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  XSub0

Description: Initial X Value of Subordinate Edge
Offset: 0000 0090
Reboot 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  dY

Description: Y Delta
Offset: 0000 00A8
Reboot 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 Y0

Description: Starting Y Value
Offset: 0000 00A0
Reboot 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 dZdxL

Description: Lower part of dZdX
Offset: 0000 0070
Reboot Value: Undefined
Access: Read/Write

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

2.21 dZdxU

Description: Upper part of dZdX
Offset: 0000 0068
Reboot Value: Undefined
Access: Read/Write

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

2.22  dZdyDomL

Description:  Depth Derivative Y Dominant - Lower
Offset: 0000 0060
Reboot 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. dZdyDomU holds the most significant bits, and dZdyDomL the least significant bits. The value is in 2’s complement 17.11 fixed point format.

2.23  dZdyDomU

Description:  Depth Derivative Y Dominant - Upper
Offset: 0000 0058
Reboot 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. dZdyDomU holds the most significant bits, and dZdyDomL the least significant bits. The value is in 2’s complement 17.11 fixed point format.

2.24  Z0L

Description:  Depth Start Value - Lower
Offset: 0000 0050
Reboot Value: Undefined
Access: Read/Write
This register holds part of the start value for depth interpolation. Z0U holds the most significant bits, and Z0L the least significant bits. The combined value is in 2’s complement 17.11 fixed point format.

2.25 Z0U

**Description:** Depth Start Value - Upper

**Offset:** 0000 0048

**Reboot Value:** Undefined

**Access:** Read/Write

This register holds part of the start value for depth interpolation. Z0U holds the most significant bits, and Z0L 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
Reboot Value: 0000 0020
Access: Read/Write

3.2 VClkCtl

Description: Clock Controller for Video Unit
Offset: 0000 01C8
Reboot Value: Undefined
Access: Read/Write

3.3 MemoryCfg

Description: Configuration for Memory Options
Offset: 0000 01F8
Reboot 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 DMACtrl

Description: DMA Configuration
Offset: 0000 01D0
Reboot Value: Undefined
Access: Read/Write

Bit 10: In DMA Using AGP
  0 – Input DMA uses PCI master
  1 – Input DMA uses AGP master

Bit 11: In DMA Byte Swap
  0 – Little endian
  1 – Big endian

Bit 12: Long Read Disable
  0 – Long reads allowed
  1 – Long reads disabled

Bit 13: 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 14: AGP Data Throttle. Applies to all AGP transfers.
  0 – Control data flow using bus protocols
1 – Throttle data requests based on FIFO space

Bit 15: Out DMA Byte Swap Control
   0 – Little endian
   1 – Big endian

Bit 16: AGP High Priority
   0 – Use AGP low priority reads
   1 – Use AGP high priority reads

Bits 17-18: Texture Execute Byte Swap
   0 – Standard
   1 – Read buffer selected by bit 31 of memory contents
   2 – Half Word Swapped
   3 – Reserved

### 3.5 FIFO\text{Ctrl}

**Description:** Control values for FIFO

**Offset:** 0000 01A8

**Reboot Value:** Undefined

**Access:** Read/Write

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

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 MemoryCtrl

Description: Memory Control Register

Offset: 0000 01E0

Reboot Value: 0

Access: Read/Write

Bit 6: SDRAM
0 – SGRAM fitted
1 – SDRAM fitted

3.7 LineCount

Description: Current Line Count

Offset: 0000 01B0

Reboot Value: Undefined

Access: Read/Write

3.8 DisconDFIFO

Description: Enable/Disable Debug Disconnect FIFO Signals

Offset: 0000 0200

Reboot 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 DisconFIFO

Description: Enable/Disable Disconnect FIFO Signals
Offset: 0000 01D8
Reboot 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 HbEnd

Description: Horizontal Blank Stop
Offset: 0000 0140
Reboot Value: Undefined
Access: Read/Write
3.11 HgEnd

Description: Horizontal Out of Band Timing
Offset: 0000 0138
Reboot Value: Undefined
Access: Read/Write

3.12 HsStart

Description: Horizontal Sync Start
Offset: 0000 0128
Reboot Value: Undefined
Access: Read/Write

3.13 HsEnd

Description: Horizontal Sync Stop
Offset: 0000 0130
Reboot Value: Undefined
Access: Read/Write

3.14 HTotal

Description: Horizontal Total
Offset: 0000 0120
Reboot Value: Undefined
Access: Read/Write
3.15  InterruptLine

Description:  InterruptLine
Offset:  0000 0198
Reboot Value:  Undefined
Access:  Read/Write

Bits 21-31: Line
  Generate interrupt at start of this line.

3.16  Aperture1

Description:  Memory Access One Control Register
Offset:  0000 0110
Reboot 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 Aperture2

Description: Memory Access Two Control Register

Offset: 0000 0118

Reboot 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 ScreenStride

Description: Stride between scanlines of display

Offset: 0000 0178

Reboot Value: Undefined

Access: Read/Write
3.19 ScreenBase

Description: Screen Origin
Offset: 0000 01C0
Reboot Value: Undefined
Access: Read/Write

3.20 ScreenBaseRight

Description: Screen Origin Right
Offset: 0000 01B8
Reboot Value: Undefined
Access: Read/Write

3.21 VbEnd

Description: Vertical Blank Stop
Offset: 0000 0160
Reboot Value: Undefined
Access: Read/Write

3.22 VsStart

Description: Vertical Sync Start
Offset: 0000 0150
Reboot Value: Undefined
Access: Read/Write
3.23  VsEnd

Description:  Vertical Sync Stop
Offset:  0000 0158
Reboot Value:  Undefined
Access:  Read/Write

3.24  VTotal

Description:  Vertical Total
Offset:  0000 0148
Reboot Value:  Undefined
Access:  Read/Write

3.25  DisplayData

Description:  View Data Registers
Offset:  0000 01A0
Reboot 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

**Reboot Value:** Undefined

**Access:** Read/Write

```
  31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Data Pattern
```

**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.

![Diagram of Video Timing Parameters]

Figure 1: Video Timing Parameters
4.2 Example Timing Values for 800x600 32BPP 75Hz

<table>
<thead>
<tr>
<th>Register</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTotal</td>
<td>041F0000</td>
</tr>
<tr>
<td>HsStart</td>
<td>00001000</td>
</tr>
<tr>
<td>HsEnd</td>
<td>00060000</td>
</tr>
<tr>
<td>HbEnd</td>
<td>00001000</td>
</tr>
<tr>
<td>HgEnd</td>
<td>00010000</td>
</tr>
<tr>
<td>VTotal</td>
<td>02700000</td>
</tr>
<tr>
<td>VsStart</td>
<td>00000000</td>
</tr>
<tr>
<td>VsEnd</td>
<td>00003000</td>
</tr>
<tr>
<td>VbEnd</td>
<td>00000019</td>
</tr>
<tr>
<td>ScreenBase</td>
<td>00000000</td>
</tr>
<tr>
<td>ScreenBaseRight</td>
<td>00000000</td>
</tr>
<tr>
<td>CtrlGraphics</td>
<td>00011401</td>
</tr>
</tbody>
</table>

4.3 Example Control Values for 800x600 32BPP 75Hz

<table>
<thead>
<tr>
<th>Register</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aperture1</td>
<td>00000000</td>
</tr>
<tr>
<td>Aperture2</td>
<td>00000000</td>
</tr>
<tr>
<td>InterruptLine</td>
<td>00000000</td>
</tr>
<tr>
<td>FIFOCtrl</td>
<td>00008010</td>
</tr>
<tr>
<td>LineCount</td>
<td>00000000</td>
</tr>
<tr>
<td>VClkCtl</td>
<td>00000019</td>
</tr>
<tr>
<td>DMACtrl</td>
<td>00000000</td>
</tr>
<tr>
<td>DisconFIFO</td>
<td>0000010</td>
</tr>
<tr>
<td>MemoryCtrl</td>
<td>00000000</td>
</tr>
<tr>
<td>MemoryCfg</td>
<td>E6002021</td>
</tr>
<tr>
<td>BootAddress</td>
<td>0C600000</td>
</tr>
<tr>
<td>DisplayData</td>
<td>00000000</td>
</tr>
<tr>
<td>WriteMaskBypass</td>
<td>FFFFFFFF</td>
</tr>
</tbody>
</table>
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.

![Diagram of a triangle with vertices labeled and color components shown.]

Figure 2: Example Triangle

The diagram makes a distinction between top and bottom halves because ZACH4 is designed to rasterize screen aligned trapezoids and flat topped or bottomed triangles as shown below:

![Screen aligned trapezoid and flat topped triangle images.]

Figure 3: Screen aligned trapezoid and flat topped triangle
5.2 Initialization

ZACH4 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. ZACH4 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.

5.4 Calculating Color Values for Interpolation

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

\[
\begin{align*}
    dRdy_{13} &= \frac{R_3 - R_1}{Y_3 - Y_1}, \quad dGdy_{13} = \frac{G_3 - G_1}{Y_3 - Y_1}, \quad dBdy_{13} = \frac{B_3 - B_1}{Y_3 - Y_1}
\end{align*}
\]

And from the plane equation:

\[
\begin{align*}
    dRdx &= \left(\left( R_1 - R_3 \right) \times \frac{Y_2 - Y_3}{\alpha} \right) - \left(\left( R_2 - R_3 \right) \times \frac{Y_1 - Y_3}{\alpha} \right) \\
    dGdx &= \left(\left( G_1 - G_3 \right) \times \frac{Y_2 - Y_3}{\alpha} \right) - \left(\left( G_2 - G_3 \right) \times \frac{Y_1 - Y_3}{\alpha} \right) \\
    dBdx &= \left(\left( B_1 - B_3 \right) \times \frac{Y_2 - Y_3}{\alpha} \right) - \left(\left( B_2 - B_3 \right) \times \frac{Y_1 - Y_3}{\alpha} \right)
\end{align*}
\]

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.

```markdown
// Load the color start and delta values to draw a triangle
R0 (R1)
G0 (G1)
B0 (B1)
// To walk up the dominant edge
dRdyDom (dRdy_{13})
dGdyDom (dGdy_{13})
dBdyDom (dBdy_{13})
// To walk along the scanline
dRdx (dRdx)
dGdx (dGdx)
 dBdx (dBdx)
```

5.6 Register Set-up for Depth Testing

Internally ZACH4 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 ZACH4 would need its registers set-up as follows:

```markdown
// Load the depth start and delta values
// to draw a triangle
```
5.7 Calculating the Slopes for each Side

ZACH4 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 ZACH4 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 XDom0 and XSub0. 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), XSub0 must be set to \( X_2 \).

5.8 Rasterizer Configuration

The ZACH4 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.

\texttt{RasterizerCfg (0x40000000)} // Default Rasterizer mode
5.9 Subpixel Correction

ZACH4 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 Render 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

ZACH4 is almost ready to draw the triangle. Setting up the registers as described here and sending the Render command will cause the top half of the example triangle to be drawn.

For drawing the example triangle, all the bit fields within the Render 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
XDom0 \( (X_1) \)
dXDom \( (dX_{13}) \)
XSub0 \( (X_1) \)
dXSub \( (dX_{12}) \)
Y0 \( (Y_1) \)
dY \( (1) \)
Count \( (Y_1 - Y_2) \)

// Draw the top half of the triangle
Render \( (render) \)

After the Render 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 ExtTriSub sent. Once ZACH4 has received ExtTriSub, drawing of this sub-triangle will begin.

// Set-up the delta and start for the new edge
XSub0 \( (X_2) \)
dXSub \( (dX_{23}) \)

// Draw sub-triangle
ExtTriSub \( (Y_2 - Y_3) \) // 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 FIFOSpaceFree register is not updated until ZACH4 processes this entry. In the case where ZACH4 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 FIFOSpaceFree register can be read to determine if the FIFO Queue is actively being processed. A pseudocode example of loading ZACH4 registers and sing the FIFOSpaceFree 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( *FIFOSpaceFree != 0x00000400);  
//Send more commands  
...

In kernel space:

...  
//Send commands to graphics card  
//Wait for FIFO Queue to drain  
while( *FIFOSpaceFree != 0x00000400) {  
    schedule();  
}  
//Send more commands  
...
The FIFOSpaceFree 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 ZACH4. Given that the FIFO can accommodate only a small number of entries, ZACH4 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 ZACH4. 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 ZACH4 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 DMABufAddress register and the count of the number of words to transfer to the DMABufCount register. Writing to the DMABufCount 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, ZACH4 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 DMABufAddress and DMABufCount 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

DMABufAddress(address of dma_buffer)
DMABufCount(number of words in DMA buffer)

return

Using DMA leaves the host free to return to the application, while in parallel, ZACH4 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 ZACH4 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 ZACH4 processing.

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

while (*DMABufCount != 0)

DMABufAddress(address of new buffer)
DMABufCount(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.*

```c
UINT32 *pbuf
while (*DMABufCount != 0) // wait for DMA to complete
    DMABufAddress (physical address of dma_buffer)
    pbuf = dma_buffer
    *pbuf++ = &dXDom>>2
    *pbuf++ = 0
    *pbuf++ = &dXSub>>2
    *pbuf++ = 0
    *pbuf++ = &dY>>2
    *pbuf++ = 1
    for (i = 0; i < nrects; ++i) {
        *pbuf++ = &XDomInit>>2
        *pbuf++ = rect->x1 // Start dominant edge
        *pbuf++ = &XSubInit>>2
        *pbuf++ = rect->x2 // Start of subordinate edge
        *pbuf++ = &Count>>2
        *pbuf++ = rect->y2 - rect->y1
        *pbuf++ = &YInit>>2
        *pbuf++ = rect->y1
        *pbuf++ = &Render>>2
        *pbuf++ = ZACH4_TRAPEZOID_SPC
    }
    DMABufCount((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 ZACH4 DMA controller. Normally, this means that the address passed to ZACH4 must
be the physical address of the DMA buffer in host memory. The buffer must also reside at contiguous physical addresses as accessed by ZACH4. 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

ZACH4 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 DMABufCount 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:

```c
// prepare DMA buffer
DMABufCount(n); // start a DMA transfer

// prepare next DMA buffer

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

DMABufCount(n) // start the next DMA sequen...```
The interrupt handler could then be

```c
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.