Voodoo Banshee®

UNIVERSAL ACCESS 2D DATABOOK

Revision 1.0
June 11, 1998

3Dfx Interactive, Inc.
4435 Fortran Drive
San Jose, CA 95134
www.3Dfx.com

Copyright Notice:

[English translations from legalese in brackets]

No Warranty

THESE SPECIFICATIONS ARE PROVIDED BY 3DFX "AS IS" WITHOUT ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS, OR ARISING FROM THE COURSE OF DEALING BETWEEN THE PARTIES OR USAGE OF TRADE. IN NO EVENT SHALL 3DFX BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DIRECT OR INDIRECT DAMAGES, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SPECIFICATIONS, EVEN IF 3DFX HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

[You're getting it for free. We believe the information provided to be accurate. Beyond that, you're on your own.]

1. Introduction

Banshee Graphics Engine is the second generation 3D graphics engine based on the original SST1 architecture. Banshee incorporates all of the original SST1 features such as true-perspective texture mapping with advanced mipmapping and lighting, texture anti-aliasing, sub-pixel correction, gouraud shading, depth-buffering, alpha blending and dithering. In addition to the SST1 features, Banshee will include a VGA core, 2D graphics acceleration, and support for Intel's AGP bus.

Features

- SST1 baseline features
- SST1 software compatible
- AGP / PCI bus compliant
- Native VGA core
- 2D acceleration
  Binary/Ternary operand raster ops
  Screen to Screen, Screen to Texture space, and Texture space to Screen Blits.
  Color space conversion YUV to RGB.
  1:N monochrome expansion
  Rendering support of 2048x2048
- Integrated DAC and PLLs.
- Bilinear video scaling
- Video in via feature connector
- Supports SGRAM memories

Video-In:

- decimation
- support for interlaced video data
- support VMI, SAA7110 video connectors
- tripple buffers for video-in data

Video-Out:

- Bilinear scaling zoom-in (from 1 to 10x magnification in increments of 0.25x)
- decimation for zoom-out (0.25x, 0.5x, 0.75x)
- chroma-keying for video underlying and overlaying
- support for stereoscopic display
- hardware cursor
- double buffer frame buffers for video refresh
- DDC support for monitor communication
- DPMS mode support
- overlay windows (for 3D and motion video)

Resolutions

VGA

MODE #	Mode Type	# of Colors	Native Resolution	Alpha Format
0,1	Alpha	16/256K	320x200	40x25
0,1	Alpha	16/256K	320x350	40x25
0,1	Alpha	16/256K	360x400	40x25
2,3	Alpha	16/256K	640x200	80x25
2,3	Alpha	16/256K	720x400	80x25
2,3	Alpha	16/256K	320x200	80x25
4,5	Graphics	4/256K	640x200	40x25
6	Graphics	2/256K	120x350	80x25
7	Alpha	mono	320x200	80x25
D	Graphics	16/256K	640x350	40x25
E	Graphics	16/256K	640x350	40x25
F	Graphics	mono	640x350	80x25
10	Graphics	16/256K	640x350	80x25
11	Graphics	2/256K	640x480	80x30
12	Graphics	16/256K	640x480	80x30
13	Graphics	256/256K	320x200	40x25

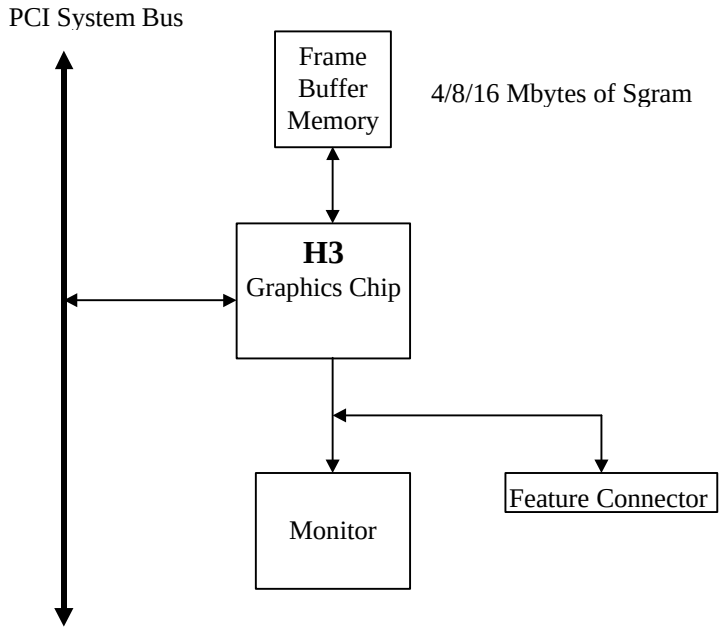
VESA

MODE #	Mode Type	# of Colors	Native Resolution	Alpha Format
100	Graphics	256/256K	640x400	80x25
101	Graphics	256/256K	640x480	80x30

2. Functional Overview

System Level Diagram

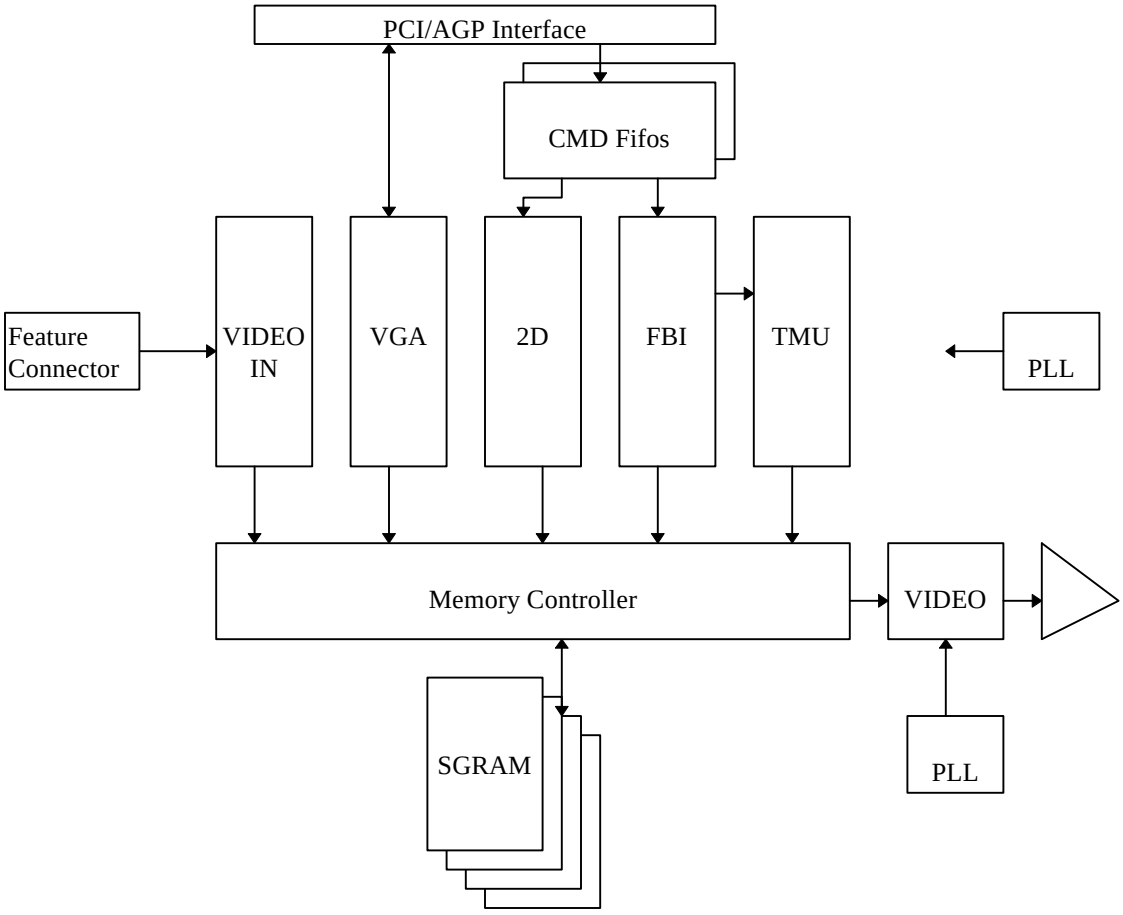
In its entry configuration, a Banshee graphics solution consists of a single ASIC + RAM. Banshee is a PCI Slave device, that receives commands from the CPU via direct writes or through memory backed fifo writes. Banshee includes an entire VGA core, 2D graphic pipeline, 3D graphics engine, texture raster engine, and video display processor. Banshee supports all VGA modes plus a number of Vesa modes.



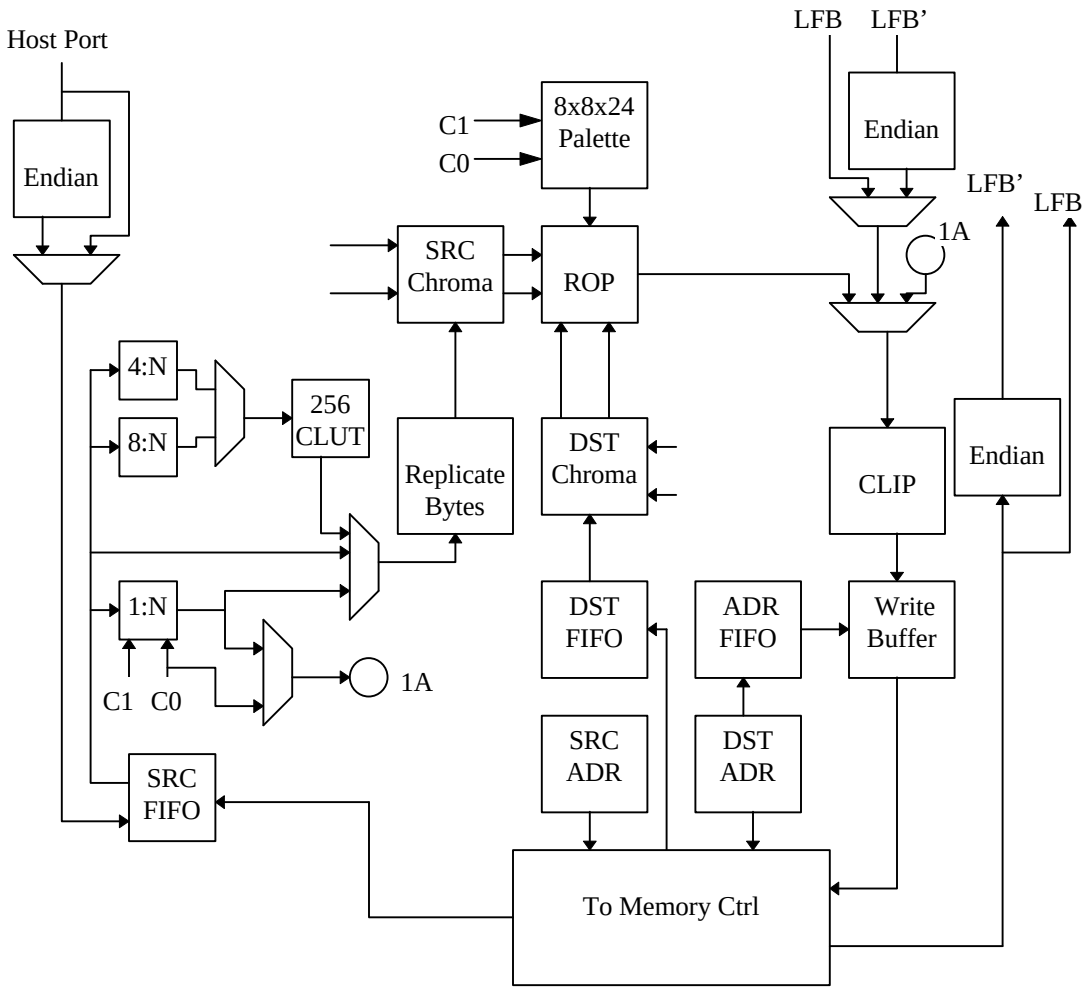
Architectural Overview

Graphical Overview

The diagram below illustrates the overall architecture of the Banshee graphics subsystem.



2D



Functional Overview

Bus Support: Banshee implements both the PCI bus specification 2.1 and AGP specification 1.0 protocols. Banshee is a slave only device on PCI, and a master device on AGP. Banshee supports zero-wait-state transactions and burst transfers.

PCI Bus Write Posting: Banshee uses a synchronous FIFO 32ntries deep which allows sufficient write posting capabilities for high performance. The FIFO is asynchronous to the graphics engine, thus allowing the memory interface to operate at maximum frequency regardless of the frequency of the PCI bus. Zero-wait-state writes are supported for maximum bus bandwidth.

VGA: Banshee includes a 100% IBM PS/2 model 70 compatible VGA core, which is highly optimized for 128 bit memory transfers. The VGA core supports PC '97 requirements for multiple adapter, and vga disable.

Memory Architecture: The frame buffer controller of Banshee has a 128-bit wide datapath with support for up to 100 MHz SGRAMs or SDRAMS. For 2D fills using the standard 2D bitBLT engine, 8 16-bit pixels are written per clock, resulting in a 800 Mpixel/sec peak fill rate. For screen clears using the color expansion capabilities specific to SGRAM, 64 bytes are written per clock, resulting in a 6.4 Gbytes/sec peak fill rate. The minimum amount of memory supported by Banshee is 4 Mbytes, with a maximum of 16 Mbytes supported.

Host Bus Addressing Schemes: Banshee occupies a combined 64 Mbytes of memory mapped address space, using two PCI memory base address pointers. Banshee also occupies 256 bytes of I/O mapped address space for video and initialization registers. The register space of Banshee occupies 6 Mbytes of address space, the linear frame buffer occupies 32 Mbytes of address space.

2D Architecture: Banshee implements a full featured 128-bit 2D windows accelerator capable of displaying 8, 16, 24, and 32 bits-per-pixel screen formats. Banshee supports 1, 8, 16, 24, and 32 bits-per-pixel RGB source pixel maps for BitBlts. 4:2:2 and 4:1:1 YUV colorspace are supported as source bitmaps for host to screen BitBlts. Banshee supports screen-to-screen and host-to-screen stretch BitBlts at 100 Mpixels/Sec. Banshee supports source and destination colorkeying, multiple clip windows, and full support of ternary ROP's. Patterned Bresenham line drawing with full ROP support, along with polygon fills are supported in Banshee's 2D core. Fast solid fills, pattern fills, and transparent monochrome bitmap BitBlts in 8 bits-per-pixel, 16 bits-per-pixel, and 32 bits-per-pixel modes.

3. Banshee Address Space

MemoryBase0

Memory Address	
0x0000000 - 0x007FFFF	I/O register remap (See I/O section below)
0x0080000 - 0x00FFFFF	CMD/AGP transfer/Misc registers
0x0100000 - 0x01FFFFF	2D registers
0x0200000 - 0x07FFFFF	Reserved.
0x0800000 - 0x0BFFFFF	Reserved.
0x0C00000 - 0x0FFFFFF	YUV planar space

Memory Base1

Memory Address	
0x0000000 - 0x1FFFFFF	LFB space

I/O Base0

I/O Address	
0x00 - 0x03	status Register
	Initialization registers
0x04 - 0x07	pciInit0 register
0x08 - 0x0b	sipMonitor register
0x0c - 0x0f	lfbMemoryConfig register
0x10 - 0x13	miscInit0 register
0x14 - 0x17	miscInit1 register
0x18 - 0x1b	dramInit0 register
0x1c - 0x1f	dramInit1 register
0x20 - 0x23	agpInit register
0x24 - 0x27	tmuGbeInit register
0x28 - 0x2b	vgaInit0 register
0x2c - 0x2f	vgaInit1 register
0x30 - 0x33	dramCommand register (see 2D offset 0x70)
0x34 - 0x37	dramData register (see 2D offset 0x064)
0x38 - 0x3b	reserved
	PLL and Dac registers
0x40 - 0x43	pllCtrl0
0x44 - 0x47	pllCtrl1
0x48 - 0x4b	pllCtrl2
0x4c - 0x4f	dacMode register.
0x50 - 0x53	dacAddr register.
0x54 - 0x57	dacData register.
	Video Registers part I
0x58 - 0x5b	rgbMaxDelta register
0x5c - 0x5f	vidProcCfg register.
0x60 - 0x63	hwCurPatAddr register.

0x64 - 0x67	hwCurLoc register.
0x68 - 0x6b	hwCurC0 register
0x6c - 0x6f	hwCurC1 register.
0x70 - 0x73	vidInFormat register
0x74 - 0x77	vidInStatus register
0x78 - 0x7b	vidSerialParallelPort register
0x7c - 0x7f	vidInXDecimDeltas register.
0x80 - 0x83	vidInDecimInitErrs register.
0x84 - 0x87	vidInYDecimDeltas register.
0x88 - 0x8b	vidPixelBufThold register
0x8c - 0x8f	vidChromaMin register.
0x90 - 0x93	vidChromaMax register.
0x94 - 0x97	vidCurrentLine register.
0x98 - 0x9b	vidScreenSize register.
0x9c - 0x9f	vidOverlayStartCoords register.
0xa0 - 0xa3	vidOverlayEndScreenCoord register.
0xa4 - 0xa7	vidOverlayDudx register
0xa8 - 0xab	vidOverlayDudxOffsetSrcWidth register.
0xac - 0xaf	vidOverlayDvdy register.
	VGA Registers
0xb0 - 0xdf	vga registers (only in I/O space, not memory mapped)
	Video Registers part II
0xe0 - 0xe3	vidOverlayDvdyOffset register.
0xe4 - 0xe7	vidDesktopStartAddr register.
0xe8 - 0xeb	vidDesktopOverlayStride register.
0xec - 0xef	vidInAddr0 register
0xf0 - 0xf3	vidInAddr1 register.
0xf4 - 0xf7	vidInAddr2 register.
0xf8 - 0xfb	vidInStride register.
0xfc - 0xff	vidCurrOverlayStartAddr register.

VGA Address Space

Memory Address
0x00A0000 - 0x00BFFFF
I/O Addresses (8 bit / 16 bit) addressable
0x0102
0x03B4 - 0x03B5
0x03BA
0x3C0 - 0x03CA
0x03CE - 0x03CF
0x03DA
0x46E8

4. Basic Init Procedures

Primary device

When Banshee is a primary device, and it's installed in a system capable of running x86 BIOS, the Banshee BIOS will perform the basic card init. This includes setting up the memory timings, clock settings, and other startup paramaters.

Secondary device, or on non-x86 platforms

If Banshee is running as a secondary device, the BIOS does not have a chance to setup the proper startup values. The proper values are stored in the banshee ROM. In order to setup the Banshee correctly, these values will need to be read out of the ROM installed on the card, and programmed in to the appropriate registers.

On non-x86 systems, the expansion ROM will often already have a PCI Expansion ROM base address. However, most PC systems blank out the expansion rom address after shadowing the x86 part of the expansion ROM into system memory. If for some reason the expansion rom does not have a PCI base address, on Banshee, it is safe to *borrow* pci address space from the linear frame buffer. The address decoding hardware guarantees that if the expansion rom is mapped on top of the linear frame buffer, the expansion rom will have priority. In order to do this, set the PCI config space expansion rom base address to be the same as the frame buffer base address for that card, perform the necessary accesses to rom, and then set the expansion rom base address back to it's previous value. Below is some example code to perform this address space borrowing and access the ROM. It makes use of standard pci access functions, and the external "mdc_crc_getbuffer()" which performs the CRC32 of the buffer.

```
FxBool bansheeChecksumRom(LPCARDINFO card, FxU32 *checksum_1sthalf, FxU32 *romSize) {
  SstIORegs *ioregs = (SstIORegs *)(card->NatMem0.MappedAddr);
  FxU32 old_miscInit1;
  FxU32 base_addr,old_base_addr;
  FxU32 phys_addr;
  FxU32 mapped_addr;
  FxU32 size;
  FxU32 checksum_1;
  char *ptr;

  pciGetConfigData( PCI_ROM_BASE_ADDRESS, card->pciDevNum, &base_addr);
  old_base_addr = base_addr;

  /* we are going to borrow address space from the MMIO registers */
  base_addr = card->PCIBase0 + 0x800000;

  phys_addr = base_addr & ~0x7FF;
  if (ioregs->reservedZ[1] & BIT(1)) {
    size = BANSHEE_ROM_SIZE; /* 64k */
    if (romSize) { *romSize = 64; }
```

```
  } else {
    size = BANSHEE_ROM_SIZE/2; /* 32k */
    if (romSize) { *romSize = 32; }
  }

  mapped_addr = card->NatMem0.MappedAddr + 0x800000;
  printf("old miscInit1 = 0x%X\n",old_miscInit1 = ioregs->miscInit1);
  ioregs->miscInit1 |= BIT(25);

  printf("banshee ROM base (0x%X,0x%X,size = %d)\n",
         base_addr,phys_addr,size);

  /* enable ROM address decoding */
  base_addr &= ~0x1;
  pciSetConfigData( PCI_ROM_BASE_ADDRESS, card->pciDevNum, &base_addr);
  base_addr |= 1;
  pciSetConfigData( PCI_ROM_BASE_ADDRESS, card->pciDevNum, &base_addr);


  /* checksum the rom */
  ptr = (char *)mapped_addr;

  if ((*(ptr) != 0x55) || (*(ptr + 1) != 0xAA)) {
    /* signature didn't match! */

    /* disable ROM address decoding */
    old_base_addr &= ~0x1;
    pciSetConfigData( PCI_ROM_BASE_ADDRESS, card->pciDevNum, &old_base_addr);

    printf("Couldn't find ROM signature: 0x55, 0xAA\n");
    ioregs->miscInit1 = old_miscInit1;
    return (FXFALSE);

  }
  printf("ptr = 0x%X, size=0x%X\n",ptr,size);

  checksum_1 = mdc_getcrc_buffer(ptr,size/2);
  printf("checksum 1    = 0x%X\n",checksum_1);

  if (checksum_1sthalf) {
    *checksum_1sthalf = checksum_1;
  }
  ioregs->miscInit1 = old_miscInit1;
  /* disable ROM address decoding */
  old_base_addr &= ~0x1;
  pciSetConfigData( PCI_ROM_BASE_ADDRESS, card->pciDevNum, &old_base_addr);

  return FXTRUE;
}
```

After you can access the expansion rom, you will need to find the important startup data. In the ROM, there is a table of configuration data which looks like this:

```
struct tblOEMConfig_struct {
        UINT32      regPCIInit0;        /* IOBase[04h] */
        UINT32      regMiscInit0;       /* IOBase[10h] */
        UINT32      regMiscInit1;       /* IOBase[14h] */
        UINT32      regDRAMInit0;       /* IOBase[18h] */
        UINT32      regDRAMInit1;       /* IOBase[1Ch] */
        UINT32      regAGPInit0;        /* IOBase[20h] */
        UINT32      regPLLCtrl1;        /* IOBase[44h] */
        UINT32      regPLLCtrl2;        /* IOBase[48h] */
        UINT32      regSGRAMMode;       /* IOBase[30h] and IOBase[10Dh] */
};
```

To find the OEM Config Table, you need to following these steps:

1) At offset 50h of the x86 ROM Image, there is a word value. This value is the offset within the ROM image, of the ROM Config Table.

2) The first WORD of the ROM Config Table is an offset within the ROM image, of the above OEM Config Table.

3) If the pointer to the OEM Config Table is “0”, then there is no OEM Config Table. (This will not happen in any release versions of the Banshee ROM.)

Setting up VESA desktop modes

The easiest way to setup the VESA standard desktop modes is to make use of the VGA BIOS Parameter Tables. The contents of these tables have been included here for your convenience. The layout of these Parameter tables is standardized, and should be able in popular VGA Books. However, the layout is also included below for convinence. Keep in mind that the “byte number” in the table below is the location in the BIOS Parameter list, and is not related to the register location. You can find the register locations for the listed registers in any standard VGA book. Also keep in mind that in order to write the extended CRTC registers, you must unlock them by writing a 0 to bit 7 of CR11.

Byte Number	Contents
0	Number of text columns
1	Number of text rows
2	Character height (in pixels)
3 and 4	Display page length (in bytes)
Sequencer Register Values:	
5	Clock Mode Register
6	Color Plane Write Enable Register
7	Character Generator Select Register
8	Memory Mode Register
9	Miscellaneous Register
CRT Controller Register values:	
0ah	Horizontal Total Register
0bh	Horizontal Display End Register
0ch	Start Horizontal Blanking Register
0dh	End Horizontal Blanking Register
0eh	Start Horizontal Retrace Register
0fh	End Horizontal Retrace Register

10h	Vertical Total Register
11h	Overflow Register
12h	Preset Row Scan Register
13h	Maximum Scan Line Register
14h	Cursor Start
15h	Cursor End
16h-19h	Unused
1ah	Vertical Retrace Start Register
1bh	Vertical Retrace End Register
1ch	Vertical Display End Register
1dh	Offset Register
1eh	Underline Location Register
1fh	Start Vertical Blanking Register
20h	End Vertical Blanking Register
21h	Mode Control Register
22h	Line Compare Register
Attribute Controller Register Values:	
23h	Palette Register 0
24h	Palette Register 1
25h	Palette Register 2

26h	Palette Register 3
27h	Palette Register 4
28h	Palette Register 5
29h	Palette Register 6
30h	Palette Register 7
31h	Palette Register 8
32h	Palette Register 9
33h	Mode Control Register
34h	Screen Border Color (Overscan) Register
35h	Color Plane Enable Register
36h	Horizontal Panning Register
Graphics Controller register values:	
37h	Set/Reset Register
38h	Set/Reset Enable Register
39h	Color Compare Register
3ah	Data Rotate & Function Select Register
3bh	Read Plane Select Register
3ch	Mode Register
3dh	Miscellaneous Register

Below are the actual mode paramater tables for all the supported VESA standard modes.

```
tblExtModeParms       label   byte
;
; Mode 55h / VESA Mode 109h / Internal Mode 1Dh
;       132x25 Color Text (8x16 font) - 40.0 MHz, 31.5 KHz, 70 Hz
;
        db      084h, 018h, 010h
        dw      02000h
        db      001h, 003h, 000h, 002h
        db      06Fh
        db      09Ah, 083h, 084h, 09Dh, 085h, 013h
        db      0BFh, 01Fh, 000h, 04Fh, 00Dh, 00Eh
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 042h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 54h / VESA Mode 10Ah / Internal Mode 1Eh
;       132x43 Color Text (8x9 font) - 40.0 MHz, 31.5 KHz, 70 Hz
;
        db      084h, 02Ah, 009h
        dw      04000h
```

```
        db      001h, 003h, 000h, 002h
        db      06Fh
        db      09Ah, 083h, 084h, 09Dh, 085h, 013h
        db      0BFh, 01Fh, 000h, 048h, 007h, 008h
        db      000h, 000h, 000h, 000h, 092h, 084h
        db      082h, 042h, 01Fh, 089h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 65h / VESA Mode 10Bh / Internal Mode 1Fh
;       132x50 Color Text (8x8 font) - 40.0 MHz, 31.5 KHz, 70 Hz
;
        db      084h, 031h, 008h
        dw      04000h
        db      001h, 003h, 000h, 002h
        db      06Fh
        db      09Eh, 083h, 084h, 081h, 08Ah, 09Eh
        db      0BFh, 01Fh, 000h, 047h, 006h, 007h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 042h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 64h / VESA Mode 10Ch / Internal Mode 20h
;       132x60 Color Text (8x8 font) - 40.0 MHz, 31.5 KHz, 60 Hz
;
        db      084h, 03Bh, 008h
        dw      04000h
        db      001h, 003h, 000h, 002h
        db      0EFh
        db      09Eh, 083h, 084h, 081h, 08Ah, 09Eh
        db      00Bh, 03Eh, 000h, 047h, 006h, 007h
        db      000h, 000h, 000h, 000h, 0EAh, 08Ch
        db      0DFh, 042h, 01Fh, 0E7h, 004h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
IF BANSHEE_DOUBLESCAN
;
; Mode 78 / VESA Mode 180 / *Internal Mode 21h*
; Mode 79 / VESA Mode 10E / Internal Mode 25h
; Mode 7A / VESA Mode 10F / Internal Mode 26h
;       320x200 - 256-color, 32K-color, 16M-color (8x8 font, 40x25 "Text")
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 008h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      06Fh
        db      02Dh, 027h, 028h, 090h, 029h, 08Fh
        db      0BFh, 01Fh, 000h, 0C0h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 01Fh, 096h, 0B9h, 0E3h
        db      0FFh
```

```
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode 20 / VESA Mode 181 / *Internal Mode 22h*
; Mode 21 / VESA Mode 182 / Internal Mode 27h
; Mode 22 / VESA Mode 183 / Internal Mode 28h
;       320x240 - 256-color, 32K-color, 16M-color (8x8 font, 40x30 "Text")
;       25.175 MHz, 31.5 KHz, 60 Hz
;
        db      028h, 01Dh, 008h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      0EFh
        db      02Dh, 027h, 028h, 090h, 029h, 08Fh
        db      00Bh, 03Eh, 000h, 0C0h, 000h, 000h
        db      000h, 000h, 000h, 000h, 0EAh, 00Ch
        db      0DFh, 028h, 01Fh, 0E7h, 004h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode 23 / VESA Mode 184 / *Internal Mode 23h*
; Mode 24 / VESA Mode 185 / Internal Mode 29h
; Mode 25 / VESA Mode 186 / Internal Mode 2Ah
;       400x300 - 256-color, 32K-color, 16M-color (8x8 font, 50x37 "Text")
;       40.000/2 MHz, 35.5 KHz, 60 Hz
;
        db      032h, 024h, 008h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      02Fh
        db      03Dh, 031h, 032h, 080h, 035h, 01Dh
        db      072h, 0F0h, 000h, 060h, 000h, 000h
        db      000h, 000h, 000h, 000h, 059h, 00Dh
        db      057h, 064h, 000h, 058h, 073h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode 26 / VESA Mode 187 / *Internal Mode 24h*
; Mode 27 / VESA Mode 188 / Internal Mode 2Bh
; Mode 28 / VESA Mode 189 / Internal Mode 2Ch
;       512x384 - 256-color, 32K-color, 16M-color (8x14 font, 64x27 "Text")
;       65.000/2 MHz, 48.0 KHz, 60 Hz
;
        db      040h, 01Ch, 00Eh
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      02Fh
        db      04Fh, 03Fh, 040h, 083h, 042h, 00Ch
        db      024h, 0F5h, 000h, 060h, 000h, 000h
        db      000h, 000h, 000h, 000h, 003h, 009h
        db      0FFh, 080h, 000h, 0FFh, 025h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 041h, 000h
```

```
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
tblStdParameters       label   byte
;
; Mode 0 / Internal Mode 00h
;       40x25 - Color Text (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 008h
        dw      00800h
        db      009h, 003h, 000h, 002h
        db      063h
        db      02Dh, 027h, 028h, 090h, 02Bh, 0A0h
        db      0BFh, 01Fh, 000h, 0C7h, 006h, 007h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 014h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 1 / Internal Mode 01h
;       40x25 - Color Text (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 008h
        dw      00800h
        db      009h, 003h, 000h, 002h
        db      063h
        db      02Dh, 027h, 028h, 090h, 02Bh, 0A0h
        db      0BFh, 01Fh, 000h, 0C7h, 006h, 007h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 014h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 2 / Internal Mode 02h
;       80x25 - Color Text (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 008h
        dw      01000h
        db      001h, 003h, 000h, 002h
        db      063h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 0C7h, 006h, 007h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 3 / Internal Mode 03h
;       80x25 - Color Text (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
;
```

```
        db      050h, 018h, 008h
        dw      01000h
        db      001h, 003h, 000h, 002h
        db      063h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 0C7h, 006h, 007h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 4 / Internal Mode 04h
;       320x200 - 4-color CGA (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 008h
        dw      04000h
        db      009h, 003h, 000h, 002h
        db      063h
        db      02Dh, 027h, 028h, 090h, 02Bh, 080h
        db      0BFh, 01Fh, 000h, 0C1h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 014h, 000h, 096h, 0B9h, 0A2h
        db      0FFh
        db      000h, 013h, 015h, 017h, 002h, 004h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 001h, 000h
        db      003h, 000h
        db      000h, 000h, 000h, 000h, 000h, 030h
        db      00Fh, 000h, 0FFh
;
; Mode 5 / Internal Mode 05h
;       320x200 - 4-color CGA (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 008h
        dw      04000h
        db      009h, 003h, 000h, 002h
        db      063h
        db      02Dh, 027h, 028h, 090h, 02Bh, 080h
        db      0BFh, 01Fh, 000h, 0C1h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 014h, 000h, 096h, 0B9h, 0A2h
        db      0FFh
        db      000h, 013h, 015h, 017h, 002h, 004h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 001h, 000h
        db      003h, 000h
        db      000h, 000h, 000h, 000h, 000h, 030h
        db      00Fh, 000h, 0FFh
;
; Mode 6 / Internal Mode 06h
;       640x200 - 2-color CGA (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 008h
        dw      04000h
        db      001h, 001h, 000h, 006h
        db      063h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      0BFh, 01Fh, 000h, 0C1h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 000h, 096h, 0B9h, 0C2h
        db      0FFh
```

```
        db      000h, 017h, 017h, 017h, 017h, 017h
        db      017h, 017h, 017h, 017h, 017h, 017h
        db      017h, 017h, 017h, 017h, 001h, 000h
        db      001h, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      00Dh, 000h, 0FFh
;
; Mode 7 / Internal Mode 07h
;       80x25 - Mono Text (9x14 Font)
;       28.321 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 00Eh
        dw      01000h
        db      000h, 003h, 000h, 003h
        db      0A6h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 04Dh, 00Bh, 00Ch
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 028h, 00Dh, 063h, 0BAh, 0A3h
        db      0FFh
        db      000h, 008h, 008h, 008h, 008h, 008h
        db      008h, 008h, 010h, 018h, 018h, 018h
        db      018h, 018h, 018h, 018h, 00Eh, 000h
        db      00Fh, 008h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Ah, 000h, 0FFh
;
; Mode 5Bh / VBE Mode 100h / *Internal Mode 08h*
; Mode 29h / VBE Mode 18Ah / Internal Mode 2Dh
; Mode 2Ah / VBE Mode 18Bh / Internal Mode 2Fh
;       640X400 - 256-color, 32K-color, 16M-color (8x16 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 010h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      063h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 00Eh
        db      08Fh, 050h, 01Fh, 096h, 0B9h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 041h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode 5Fh / VBE Mode 101h / *Internal Mode 09h*
; Mode 6E / VESA Mode 111h / Internal Mode 2Fh
; Mode 69 / VESA Mode 112h / Internal Mode 30h
;       640X480 - 256-color, 32K-color, 16M-color (8x16 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 01Dh, 010h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      0E3h
        db      05Fh, 04Fh, 050h, 082h, 052h, 09Eh
        db      00Bh, 03Eh, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 0EAh, 00Ch
        db      0DFh, 050h, 000h, 0E7h, 004h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 041h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
```

```
        db      005h, 00Fh, 0FFh
;
; Mode 6Ah / VBE Mode 102h / Internal Mode 0Ah
;       800X600 - 16-color (8x16 Font)
;       40.000 MHz, 38.000 KHz, 60 Hz
;
        db      064h, 024h, 010h
        dw      0FA00h
        db      001h, 00Fh, 000h, 006h
        db      02Fh
        db      07Fh, 063h, 064h, 082h, 06Bh, 01Bh
        db      072h, 0F0h, 000h, 060h, 000h, 000h
        db      000h, 000h, 000h, 000h, 059h, 00Dh
        db      057h, 032h, 000h, 057h, 073h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      005h, 00Fh, 0FFh
;
; Mode 5Ch / VBE Mode 103h / *Internal Mode 0Bh*
; Mode 70h / VBE Mode 114h / Internal Mode 31h
; Mode 71h / VBE Mode 115h / Internal Mode 32h
;       800X600 - 256-color, 32K-color, 16M-color (8x16 Font)
;       40.000 MHz, 38.000 KHz, 60 Hz
;
        db      064h, 024h, 010h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      02Fh
        db      07Fh, 063h, 064h, 082h, 069h, 019h
        db      072h, 0F0h, 000h, 060h, 000h, 000h
        db      000h, 000h, 000h, 000h, 059h, 00Dh
        db      057h, 064h, 000h, 058h, 073h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode 6B / VESA Mode 107 / *Internal Mode 0Ch*
; Mode 74 / VESA Mode 11A / Internal Mode 35h
; Mode 75 / VESA Mode 11B / Internal Mode 36h
;       1280x1024 - 256-color, 32K-color, 16M-color (8x16 Font)
;       108.0 MHz, 64 KHz, 60 Hz
;
        db      0A0h, 03Fh, 010h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      02Fh
        db      0CEh, 09Fh, 0A0h, 091h, 0A6h, 014h
        db      028h, 052h, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 001h, 004h
        db      0FFh, 0A0h, 000h, 001h, 028h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 041h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode D / Internal Mode 0Dh
;       320x200 - 16-color planar (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
```

```
;
        db      028h, 018h, 008h
        dw      02000h
        db      009h, 00Fh, 000h, 006h
        db      063h
        db      02Dh, 027h, 028h, 090h, 02Bh, 080h
        db      0BFh, 01Fh, 000h, 0C0h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 014h, 000h, 096h, 0B9h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      005h, 00Fh, 0FFh
;
; Mode E / Internal Mode 0Eh
;       640x200 - 16-color planar (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 008h
        dw      04000h
        db      001h, 00Fh, 000h, 006h
        db      063h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      0BFh, 01Fh, 000h, 0C0h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 000h, 096h, 0B9h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 010h, 011h, 012h, 013h
        db      014h, 015h, 016h, 017h, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      005h, 00Fh, 0FFh
;
; Mode 5Eh / VBE Mode 105h / *Internal Mode 0Fh*
; Mode 72h / VBE Mode 117h / Internal Mode 33h
; Mode 73h / VBE Mode 118h / Internal Mode 34h
;       1024X768 - 256-color, 32K-color, 16M-color (8x16 Font)
;       65.000 MHz, 48.500 KHz, 60 Hz
;
        db      080h, 02Fh, 010h
        dw      0FFFFh
        db      001h, 00Fh, 000h, 00Eh
        db      02Fh
        db      0A3h, 07Fh, 080h, 087h, 083h, 094h
        db      024h, 0F5h, 000h, 060h, 000h, 000h
        db      000h, 000h, 000h, 000h, 003h, 009h
        db      0FFh, 080h, 000h, 0FFh, 025h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 041h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
; Mode 68h / VESA Mode 108h / Internal Mode 10h
;       80x60 Color Text (8x8 font) - 25.175 MHz, 31.5 KHz, 60 Hz
;
        db      050h, 03Bh, 008h
        dw      2600h
        db      001h, 003h, 000h, 002h
        db      0E3h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      00Bh, 03Eh, 000h, 047h, 006h, 007h
        db      000h, 000h, 000h, 000h, 0EAh, 08Ch
```

```
        db      0DFh, 028h, 01Fh, 0E7h, 004h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode F / Internal Mode 11h
;       640x350 - 2-bit mono pseudo-planar (8x14 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 00Eh
        dw      08000h
        db      001h, 00Fh, 000h, 006h
        db      0A2h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      0BFh, 01Fh, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 028h, 00Fh, 063h, 0BAh, 0E3h
        db      0FFh
        db      000h, 008h, 000h, 000h, 018h, 018h
        db      000h, 000h, 000h, 008h, 000h, 000h
        db      000h, 018h, 000h, 000h, 00Bh, 000h
        db      005h, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      005h, 005h, 0FFh
;
; Mode 10h / Internal Mode 12h
;       640x350 - 16-bit planar (8x14 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 00Eh
        dw      08000h
        db      001h, 00Fh, 000h, 006h
        db      0A3h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      0BFh, 01Fh, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 028h, 00Fh, 063h, 0BAh, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      005h, 00Fh, 0FFh
;
; Mode 0* / Internal Mode 13h
;       40x25 - Color Text (8x14 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 00Eh
        dw      00800h
        db      009h, 003h, 000h, 002h
        db      0A3h
        db      02Dh, 027h, 028h, 090h, 02Bh, 0A0h
        db      0BFh, 01Fh, 000h, 04Dh, 00Bh, 00Ch
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 014h, 01Fh, 063h, 0BAh, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
```

```
; Mode 1* / Internal Mode 14h
;       40x25 - Color Text (8x14 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 00Eh
        dw      00800h
        db      009h, 003h, 000h, 002h
        db      0A3h
        db      02Dh, 027h, 028h, 090h, 02Bh, 0A0h
        db      0BFh, 01Fh, 000h, 04Dh, 00Bh, 00Ch
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 014h, 01Fh, 063h, 0BAh, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 2* / Internal Mode 15h
;       80x25 - Color Text (8x14 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 00Eh
        dw      01000h
        db      001h, 003h, 000h, 002h
        db      0A3h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 04Dh, 00Bh, 00Ch
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 028h, 01Fh, 063h, 0BAh, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 3* / Internal Mode 16h
;       80x25 - Color Text (8x14 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 00Eh
        dw      01000h
        db      001h, 003h, 000h, 002h
        db      0A3h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 04Dh, 00Bh, 00Ch
        db      000h, 000h, 000h, 000h, 083h, 085h
        db      05Dh, 028h, 01Fh, 063h, 0BAh, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 008h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 0+/1+ / Internal Mode 17h
;       40x25 - Color Text (9x16 Font)
;       28.321 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 010h
        dw      00800h
        db      008h, 003h, 000h, 002h
        db      067h
        db      02Dh, 027h, 028h, 090h, 02Bh, 0A0h
```

```
        db      0BFh, 01Fh, 000h, 04Fh, 00Dh, 00Eh
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 014h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 008h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 2+/3+ / Internal Mode 18h
;       80x25 - Color Text (9x16 Font)
;       28.321 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 010h
        dw      01000h
        db      000h, 003h, 000h, 002h
        db      067h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 04Fh, 00Dh, 00Eh
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 01Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 00Ch, 000h
        db      00Fh, 008h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Eh, 000h, 0FFh
;
; Mode 7+ / Internal Mode 19h
;       80x25 - Mono Text (9x16 Font)
;       28.321 MHz, 31.5 KHz, 70 Hz
;
        db      050h, 018h, 010h
        dw      01000h
        db      000h, 003h, 000h, 002h
        db      066h
        db      05Fh, 04Fh, 050h, 082h, 055h, 081h
        db      0BFh, 01Fh, 000h, 04Fh, 00Dh, 00Eh
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 00Fh, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 008h, 008h, 008h, 008h, 008h
        db      008h, 008h, 010h, 018h, 018h, 018h
        db      018h, 018h, 018h, 018h, 00Eh, 000h
        db      00Fh, 008h
        db      000h, 000h, 000h, 000h, 000h, 010h
        db      00Ah, 000h, 0FFh
;
; Mode 11h / Internal Mode 1Ah
;       640x480 - 2-color planar (8x16 Font)
;       25.175 MHz, 31.5 KHz, 60 Hz
;
        db      050h, 01Dh, 010h
        dw      0A000h
        db      001h, 00Fh, 000h, 006h
        db      0E3h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      00Bh, 03Eh, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 0EAh, 08Ch
        db      0DFh, 028h, 000h, 0E7h, 004h, 0C3h
        db      0FFh
        db      000h, 03Fh, 03Fh, 03Fh, 03Fh, 03Fh
        db      03Fh, 03Fh, 03Fh, 03Fh, 03Fh, 03Fh
        db      03Fh, 03Fh, 03Fh, 03Fh, 001h, 000h
        db      001h, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
```

```
        db      005h, 001h, 0FFh
;
; Mode 12h / Internal Mode 1Bh
;       640x480 - 16-color planar (8x16 Font)
;       25.175 MHz, 31.5 KHz, 60 Hz
;
        db      050h, 01Dh, 010h
        dw      0A000h
        db      001h, 00Fh, 000h, 006h
        db      0E3h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      00Bh, 03Eh, 000h, 040h, 000h, 000h
        db      000h, 000h, 000h, 000h, 0EAh, 08Ch
        db      0DFh, 028h, 000h, 0E7h, 004h, 0E3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      014h, 007h, 038h, 039h, 03Ah, 03Bh
        db      03Ch, 03Dh, 03Eh, 03Fh, 001h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 000h
        db      005h, 00Fh, 0FFh
;
; Mode 13h / Internal Mode 1Ch
;       320x200 - 256-color (8x8 Font)
;       25.175 MHz, 31.5 KHz, 70 Hz
;
        db      028h, 018h, 008h
        dw      02000h
        db      001h, 00Fh, 000h, 00Eh
        db      063h
        db      05Fh, 04Fh, 050h, 082h, 054h, 080h
        db      0BFh, 01Fh, 000h, 041h, 000h, 000h
        db      000h, 000h, 000h, 000h, 09Ch, 08Eh
        db      08Fh, 028h, 040h, 096h, 0B9h, 0A3h
        db      0FFh
        db      000h, 001h, 002h, 003h, 004h, 005h
        db      006h, 007h, 008h, 009h, 00Ah, 00Bh
        db      00Ch, 00Dh, 00Eh, 00Fh, 041h, 000h
        db      00Fh, 000h
        db      000h, 000h, 000h, 000h, 000h, 040h
        db      005h, 00Fh, 0FFh
;
;       Copyright (c) 1990-1998 Elpin Systems, Inc.
;       All rights reserved.
;
```

5. 2D

2D Register Map

Memory Base 0: Offset 0x0100000

Register Name	Address	Reg	Bits	R/W	Description
status	0x000(0)	0x0	31:0	R	Banshee status register
intCtrl	0x004(4)	0x1	31:0	R/W	Interrupt control and status
clip0Min	0x008(8)	0x2	28:0	R/W	Min X & Y clip values when clip select is 0
clip0Max	0x00c(12)	0x3	28:0	R/W	Max X & Y clip values when clip select is 0
dstBaseAddr	0x010(16)	0x4	23:0	R/W	Destination base address
dstFormat	0x014(20)	0x5	17:0	R/W	Destination stride and bits per pixel
srcColorkeyMin	0x018(24)	0x6	23:0	R/W	Source Colorkey range (min)
srcColorkeyMax	0x01c(28)	0x7	23:0	R/W	Source Colorkey range (max)
dstColorkeyMin	0x020(32)	0x8	23:0	R/W	Destination Colorkey range (min)
dstColorkeyMax	0x024(36)	0x9	23:0	R/W	Destination Colorkey range (max)
bresError0	0x028(40)	0xA	31:0	R/W	Initial error for lines, right edges & stretch blt x
bresError1	0x02c(44)	0xB	31:0	R/W	Initial error for left poly edges & stretch blt y
rop	0x030(48)	0xC	31:0	R/W	4 Ternary Raster operations
srcBaseAddr	0x034(52)	0xD	23:0	R/W	Source base address
commandExtra	0x038(56)	0xE	31:0	R/W	Extra control bits
lineStipple	0x03c(60)	0xF	31:0	R/W	Monochrome pattern for lines
lineStyle	0x040(64)	0x10	28:0	R/W	Style register for lines
pattern0Alias	0x044(68)	0x11	31:0	R/W	Alias to colorPattern(0)
pattern1Alias	0x048(72)	0x12	31:0	R/W	Alias to colorPattern(1)
clip1Min	0x04c(76)	0x13	28:0	R/W	Min X & Y clip values when clip select is 1
clip1Max	0x050(80)	0x14	28:0	R/W	Max X & Y clip values when clip select is 1
srcFormat	0x054(84)	0x15	18:0	R/W	Source stride and bits per pixel
srcSize	0x058(88)	0x16	28:0	R/W	Height and width of source for stretch blts
srcXY	0x05c(92)	0x17	28:0	R/W	Starting pixel of blt source data Starting position for lines Top-most point for a polygon fill
colorBack	0x060(96)	0x18	31:0	R/W	Background color
colorFore	0x064(100)	0x19	31:0	R/W	Foreground color
dstSize	0x068(104)	0x1A	28:0	R/W	Destination width and height for blts and rectangle fills
dstXY	0x06c(108)	0x1B	28:0	R/W	Starting X and Y of destination for blts End point for lines
command	0x070(112)	0x1C	31:0	R/W	2D command mode & control bits
RESERVED	0x074(116)	0x1D	31:0		Do not write
RESERVED	0x078(120)	0x1E	31:0		Do not write
RESERVED	0x07c(124)	0x1F	31:0		Do not write
launchArea	0x080(128) to 0x0ff(255)	0x20 to 0x3F	31:0	R	Initiates 2D commands

colorPattern	0x100(256) to 0x1fc(508)	0x40 to 0x7F	31:0	R/W	Pattern Registers (64 entries)

Register Descriptions

The 2D register set is described in the sections below.

All 2D registers can be read, and all registers except for the status register are fully write-able. Reading a 2D register will always return the value that will be used if a new operation is begun without writing a new value to that register. This value will either be the last value written to the register, or, if an operation has been performed since the value was written, the value after all operations have completed.

All registers for the 2D section are unsigned unless specified otherwise.

status Register

The **status** register provides a way for the CPU to interrogate the graphics processor about its current state and FIFO availability. The **status** register is read only, but writing to **status** clears any Banshee generated PCI interrupts.

Bit	Description
5:0	PCI FIFO freespace (0x3f=FIFO empty). Default is 0x3f.
6	Vertical retrace (0=Vertical retrace active, 1=Vertical retrace inactive). Default is 1.
7	FBI graphics engine busy (0=engine idle, 1=engine busy). Default is 0.
8	TREX busy (0=engine idle, 1=engine busy). Default is 0.
9	Banshee busy (0=idle, 1=busy). Default is 0.
10	2D busy (0=idle, 1=busy). Default is 0.
11	Reserved
27:12	reserved
30:28	Swap Buffers Pending. Default is 0x0.
31	PCI Interrupt Generated. Default is 0x0. (not currently implemented).

Bits(5:0) show the number of entries available in the internal host FIFO. The internal host FIFO is 64 entries deep. The FIFO is empty when bits(5:0)=0x3f. Bit(6) is the state of the monitor vertical retrace signal, and is used to determine when the monitor is being refreshed. Bit(7) of **status** is used to determine if the graphics engine of FBI is active. Note that bit(7) only determines if the graphics engine of FBI is busy -- it does not include information as to the status of the internal PCI FIFOs. Bit(8) of **status** is used to determine if TREX is busy. Note that bit(8) of **status** is set if any unit in TREX is not idle -- this includes the graphics engine and all internal TREX FIFOs. Bit(9) of **status** determines if all units in the Banshee system (including graphics engines, FIFOs, etc.) are idle. Bit(9) is set when any internal unit in Banshee is active (e.g. graphics is being rendered or any FIFO is not empty). When the Memory FIFO is enabled, bits(27:12) show the number of entries available in the Memory FIFO. Depending upon the amount of frame buffer memory available, a maximum of 65,536 entries may be stored in the Memory FIFO. The Memory FIFO is empty when bits(27:12)=0xffff. Bits (30:28) of **status** track the number of outstanding SWAPBUFFER commands. When a SWAPBUFFER command is received from the host cpu, bits (30:28) are incremented -- when a SWAPBUFFER command completes, bits (30:28) are decremented. Bit(31) of status is used to monitor the status of the PCI interrupt signal. If Banshee generates a vertical retrace interrupt (as defined in **pciInterrupt**), bit(31) is set and the PCI interrupt signal line is activated to generate a hardware interrupt. An interrupt is cleared by writing to **status** with "dont-care" data. NOTE THAT BIT(31) IS CURRENTLY NOT IMPLEMENTED IN HARDWARE, AND WILL ALWAYS RETURN 0X0.

intrCtrl Register

The **intrCtrl** register controls the interrupt capabilities of Banshee. Bits 1:0 enable video horizontal sync signal generation of interrupts. Generated horizontal sync interrupts are detected by the CPU by reading bits 7:6 of **intrCtrl**. Bits 3:2 enable video vertical sync signal generation of interrupts. Generated vertical sync interrupts are detected by the CPU by reading bits 9:8 of **intrCtrl**. Bit 4 of **intrCtrl** enables generation of interrupts when the frontend PCI FIFO is full. Generated PCI FIFO Full interrupts are detected by the CPU by reading bit 10 of **intrCtrl**. PCI FIFO full interrupts are genered when **intrCtrl** bit 4 is set and the number of free entries in the frontend PCI FIFO drops below the value specified in **fbiInit0** bits(10:6). Bit 5 of **intrCtrl** enables the user interrupt command USERINTERRUPT generation of interrupts. Generated user interrupts are detected by the CPU by reading bit 11 of **intrCtrl**. The tag associated with a generated user interrupt is stored in bits 19:12 of **intrCtrl**.

Generated interrupts are cleared by writing a 0 to the bit signaling a particular interrupt was generated and writing a 1 to **interCtrl** bit(31). For example, a PCI FIFO full generated interrupt is cleared by writing a 0 to bit 10 of **intrCtrl**, and a generated user interrupt is cleared by writing a 0 to bit 11 of **intrCtrl**. For both cases, bit 31 of **intrCtrl** must be written with the value 1 to clear the external PCI interrupt. Care must be taken when clearing interrupts not to accidentally overwrite the interrupt mask bits (bits 5:0) of **intrCtrl**) which enable generation of particular interrupts.

Note that writes to the **intrCtrl** register are not pushed on the PCI frontend FIFO, so writes to **intrCtrl** are processed immediately. Since **intrCtrl** is not FIFO'ed, writes to **intrCtrl** may be processed out-of-order with respect to other queued writes in the PCI and memory-backed FIFOs.

Bit	Description
0	Horizontal Sync (rising edge) interrupts enable (1=enable). Default is 0.
1	Horizontal Sync (falling edge) interrupts enable (1=enable). Default is 0.
2	Vertical Sync (rising edge) interrupts enable (1=enable). Default is 0.
3	Vertical Sync (falling edge) interrupts enable (1=enable). Default is 0.
4	PCI FIFO Full interrupts enable (1=enable). Default is 0.
5	User Interrupt Command interrupts enable (1=enable). Default is 0.
6	Horizontal Sync (rising edge) interrupt generated (1=interrupt generated).
7	Horizontal Sync (falling edge) interrupt generated (1=interrupt generated).
8	Vertical Sync (rising edge) interrupt generated (1=interrupt generated).
9	Vertical Sync (falling edge) interrupt generated (1=interrupt generated).
10	PCI FIFO Full interrupt generated (1=interrupt generated).
11	User Interrupt Command interrupt generated (1=interrupt generated).
19:12	User Interrupt Command Tag. Read only.
20	Hole counting interupts enable (1=enable). Default is 0.
21	VMI interrupts enable. (1=enable). Default is 0.
22	Hole counting interrupt generated (1=interrupt generated).
23	VMI interrupt generated (1=interrupt generated).
29:24	reserved
30	VGA Interrupt generated (1=interrupt generated).
31	External pin pci_inta value, active low (0=PCI interrupt is active, 1=PCI interrupt is inactive)

command Register

The command register sets the command mode for the 2D engine, as well as selecting a number of options.

Bits (3:0) set the command mode for the 2D drawing engine as shown in the table below. If bit(8) is set, the command will be initiated as soon as the **command** register is written. If bit(8) is cleared, drawing will be initiated by a write to the launch area. For descriptions and examples of each command, see the 2D launch area section.

Command[3:0]	Command
0	Nop - wait for idle
1	Screen to screen blt
2	Screen to screen stretch blt
3	Host to screen blt
4	Host to screen stretch blt
5	Rectangle fill
6	Line
7	Polyline
8	Polygon fill
13	Write Sgram Mode register
14	Write Sgram Mask register
15	Write Sgram Color register

Setting Bit(9) makes line drawing reversible. If this bit is set, drawing a line from point A to point B will result in the same pixels being drawn as drawing a line from point B to point A.

Bits(11:10) control the value placed in dstXY after each blt or rectangle fill command is executed. If bit(10) is 0, dst_x is unchanged. If bit(10) is 1, dst_x gets dst_x + dst_width. If bit(11) is 0, dst_y is unchanged. If bit(11) is 1, dst_y gets dst_y + dst_height.

Bit(12) controls whether lines are stippled or solid. If bit(12) is 0, lines will be a solid color. If bit(12) is 1, lines will either be made up of either a two color pattern using **colorFore** and **colorBack** or will be a transparent stipple using **colorFore**, as determined by the transparency bit - bit(16).

Bit(13) controls the format of the pattern data. If bit(13) is set to 0, the pattern must be stored in the destination format. If it is set to 1, the pattern will be stored as a monochrome bitmap; Pattern registers 0 and 1 will be used as an 8x8x1bpp pattern, which will be expanded into the destination format using the **colorBack** and **colorFore** registers. Note that if Bit(13) is set, and Bit(16) is set to indicate that monochrome data is transparent, the pattern will be used to determine pixel transparency without regard to the contents of the ROP register.

Bits(15:14) control the direction of blting during screen-to-screen copies. Note that the corner of the source and destination rectangles passed in the **srcXY** and **dstXY** registers will change depending on the blting direction. Bit(15) also controls the direction of blting for host-to-screen copies. This can be used to flip a pixel map so that the top span in host memory is drawn as the bottom span on the screen. Note that

the direction bits only apply to “pure” screen to screen blits, but not to stretch blits. Also, destination and source color keying, along with color conversions, cannot be used with right to left blits.

Bit(16) controls whether monochrome source bitmaps, and monochrome patterns will be transparent or opaque. When bit(16) is 0, source bitmaps are opaque; a 0 in the bitmap will result in **colorBack** being written to the destination. When bit(16) is 1, source bitmaps and monochrome patterns are transparent. In this case, a 0 in the bitmap will result in the corresponding destination pixel being left unchanged.

The X and Y pattern offsets give the coordinates within the pattern of the pixel which corresponds to the destination pixel pointed to by the destination base address register. In other words, if a pattern fill is performed which covers the origin, pixel (0,0) in the destination pixel map will be written with the color in pattern pixel (x_pat_offset, y_pat_offset).

Bit(23) controls whether the clip0 or clip1 registers will be used for clipping. When bit(31) is 0, clipping values from **clip0Min** and **clip0Max** will be used, when bit(31) is 1, clipping values from **clip1Min** and **clip1Max** will be used.

Bits(31:24) contain ROP0, the ternary ROP that is used when colorkeying is disabled. For more information on ROPs, see the description of the **rop** register.

Command

Bit	Description
3:0	Command
7:4	RESERVED
8	Initiate command (1=initiate command immediately, 0 = wait for launch write)
9	Reversible lines (1=reversible, 0=non-reversible)
10	Increment destination x-start after blt or rectangle command (1=increment, 0=don’t)
11	Increment destination y-start after blt or rectangle command (1=increment, 0=don’t)
12	Stipple line mode (1 = stippled lines, 0 = solid lines)
13	Pattern Format (1 = monochrome, 0 = color)
14	X direction (0 = left to right, 1 = right to left)
15	Y direction (0 = top to bottom, 1 = bottom to top)
16	Transparent monochrome (1 = transparent, 0 = opaque)
19:17	X pattern offset
22:20	Y pattern offset
23	Clip select (0=clip0 registers, 1 = clip1 registers)
31:24	ROP0

commandExtra Register

This register contains miscellaneous control bits in addition to those in the command register.

Bits(1:0) enable colorkeying, if the bit is 0, colorkeying is disabled. Enabling source colorkeying with monochrome source, or in line, polyline, polygon, or rectangle modes has no effect. For further explanation of these bits, see the description of the colorkey registers.

If bit(2) is set, the current command, and any following it will not proceed until the next vertical blanking period begins. Wait for Vsync should not be used when performing non-DMA host blts.

If bit(3) is set, only row 0 of the pattern will be used, rather than the usual 8 pattern rows.

Command

Bit	Description
0	Enable source colorkey (1=source colorkeying enabled, 0=source colorkeying disabled)
1	Enable destination colorkey (1=enable dst colorkeying, 0=disable dst colorkeying)
2	Wait for Vsync (1=wait for vsync, 0=execute immediately)
3	Force pattern row 0 (1 = use only row 0, 0 = use all 8 pattern rows)

colorBack and colorFore Registers

The **colorBack** and **colorFore** registers specify the foreground and background colors used in solid-fill and monochrome bitmap operations, and operations using a monochrome pattern. The color registers must be stored in the destination color format.

The following tables shows the format of the color registers for each destination format.

P = palette index
R = red color channel
G = green color channel
B = blue color channel

Dst Format	Bits stored
8 bpp	0000_0000_0000_0000_0000_0000_PPPP_PPPP
15 bpp	0000_0000_0000_0000_ARRR_RRGG_GGGB_BBBB
16 bpp	0000_0000_0000_0000_RRRR_RGGG_GGGB_BBBB
24 bpp	0000_0000_RRRR_RRRR_GGGG_GGGG_BBBB_BBBB
32 bpp	AAAA_AAAA_RRRR_RRRR_GGGG_GGGG_BBBB_BBBB

colorFore

Bit	Description
31:0	foreground color

colorBack

Bit	Description
31:0	background color

Pattern Registers

The **pattern** registers contain an 8 pixel by 8 pixel pattern. The pixels must be either in the color format of the destination surface, or in 1bpp (monochrome) format. The pixels are to be written to the **pattern** registers in packed format. So, only registers 0 and 1 will be used for monochrome patterns, registers 0 through 15 will be used when the destination is 8 bpp, registers 0 through 31 will be used when the destination is 16 bpp.

Pixels should be written into the **pattern** registers starting with the upper left-hand corner of the pattern, proceeding horizontally left to right, and then vertically top to bottom. The least-significant bits of **pattern**[0] should always contain pixel(0,0) of a color pattern.

The table below give the bit position of monochrome pixels within the **pattern** registers. The bits are numbered such that bit(0) represents the lsb of a register, and bit(31) represents the msb.

Order of pixel storage in the pattern registers for a monochrome pattern

pattern(0)

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

pattern(1)

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

pattern(0-64)

Bit	Description
31:0	pattern color data

srcBaseAddr and dstBaseAddr Registers

Bits(23:0) of these registers contain the addresses of the pixels at x=0, y=0 on the source and destination surfaces in frame-buffer memory. Bit(31) of each register is reserved and must be zero.

The **srcBaseAddr** register is used only for screen-to-screen blts. For host-blts, the alignment of the initial pixel sent from the host is determined by the x entry in the **srcXY** register.

For YUYV422 and UYVY422 surfaces, the base address must be dword aligned. Thus bits(1:0) of **srcBaseAddr** must be 0.

SrcBaseAddr

Bit	Description
23:0	Source base address
30:24	RESERVED
31	Reserved (must be 0)

dstBaseAddr

Bit	Description
23:0	Destination base address
30:24	RESERVED

31	Reserved (must be 0)

srcSize and dstSize Registers

These registers are used only for blts and rectangle fills. They contain the height and width in pixels of the source and destination rectangles. The **srcSize** register will only be used in Stretch-blt modes. For non-stretched blts, the blt source size will be the same as the blt destination size, determined by the **dstSize** register.

srcSize

Bit	Description
12:0	Blt Source Width
15:13	RESERVED
28:16	Blt Source Height
31:29	RESERVED

dstSize

Bit	Description
12:0	Blt Destination Width
15:13	RESERVED
28:16	Blt Destination Height
31:29	RESERVED

srcXY and dstXY Registers

During screen-to-screen blts, the **srcXY** registers sets the position from which blt data will be read. Note that the starting position for a blit depends on the direction of the blt as shown in the table below. For lines and polylines, **srcXY** is the starting point of the first line segment. For polygons, the **srcXY** should be the topmost vertex of the polygon - that is, the vertex with the lowest y value. If there are multiple vertices sharing the lowest y value, the **srcXY** should be set to the leftmost vertex with that y value. Reading the **srcXY** register while in polygon mode will always return the last polygon vertex that the host sent for the left side of the polygon.

The values in the **srcXY** are signed, however for blts **srcXY** must contain only positive values.

During host-to-screen blts, only the x entry of the **srcXY** register is used. This entry determines the alignment of the initial pixel in the blt within the first dword sent from the host. For monochrome bitmaps, bits[4:0] are used to determine the bit position within the dword of the initial pixel. For color bitmaps, bits[1:0] are give the position within the dword of the first byte of pixel data. Host blts are always performed left-to-right (the x-direction bit in the **command** register is ignored), so the offset given will always be that of the leftmost pixel in the first span. The alignment of the initial pixel of all spans after the first is determined by adding the source stride (from the **srcFormat** register) to the alignment of the previous span.

For blts, the **dstXY** should be the starting pixel of destination rectangle as shown in the table below. For line and polyline modes, the **dstXY** will be the endpoint of the first line segment.

In polygon mode, the **dstXY** register is used to store the last vertex sent for the right side of the polygon. If command[8] is set when the command register is written in polygon mode, the value from **srcXY** will be copied to **dstXY**. If command[8] is cleared, **dstXY** can be written with the rightmost pixel in the top span of the polygon.

Command[15:14]	Starting X/Y
00	Upper Left-hand corner
01	Upper Right-hand corner
10	Lower Left-hand corner
11	Lower Right-hand corner

dstXY

Bit	Description
12:0	Signed X position on the destination surface
15:14	RESERVED
28:16	Signed Y position on the destination surface
31:30	RESERVED

srcXY

Bit	Description
12:0	Signed x position of the first source pixel
15:14	RESERVED
28:16	Signed y position of the first source pixel
31:30	RESERVED

srcFormat and dstFormat Registers

These register specify the format and strides of the source and destination surfaces

For linear surfaces, the stride of a pixel map is the number of bytes between the starting addresses of adjacent scan lines. For these surfaces, the units of the stride is always bytes, regardless of the pixel format.

The number of bits per pixel is determined as described by the tables below. The ’32 bpp’ format contains 24 bits of RGB, along with a byte of unused data, the ’24 bpp’ is packed 24 bit color.

Data coming through the host port can be byte swizzled to allow conversion between big and little endian formats, as selected by Bit 19 and 20 of src Format register. If both byte and word swizzling are enabled, the byte swizzling occurs first, followed by word swizzling.

The source packing bits control how the stride of the source will be determined during blts. If both bits are zero, the stride is set by the stride entry. Otherwise, the stride is based off of the width of the blt being performed, as shown in the table below. The stride will equal the number of bytes in a row of the rectangle being blted plus as many bytes as are required to get the necessary alignment.

For YUYV422 and UYVY422 source formats, linear strides must always be a dword multiple. Thus, bits(1:0) of the **srcFormat** register must be 0.

When necessary, the blt engine will convert source pixels to the destination format.

When source pixels in 15bpp or 16bpp format are converted to 24bpp or 32bpp, color conversion is performed by replicating the msbs of each channel into the extra lsbs required. When pixels are converted from 32bpp or 24bpp formats to 15 or 16bpp, 16bpp, the extra lsbs are removed from each channel. When any non-32bpp format is converted to 32bpp, the 8msbs of each pixel (i.e. the alpha channel) are filled with zeros.

Destination pixel formats are stored as shown in the description of the **colorFore** and **colorBack** registers. RGB source formats match these, the other source formats are shown in the table below. For monochrome source, p0 represents the leftmost pixel on the screen and p31 represents the rightmost. For YUV formats, ya represents the left pixel and yb represents the pixel to the right of ya, etc. Thus, ya7 is the msb of the y channel for the left pixel and ya0 is the lsb of the y channel for that pixel. In the diagram, the dword with the lower address (which will be quadword aligned) is shown first, followed by the dword with the higher address.

Source formats

Monochrome

p24	p25	p26	p27	p28	p29	p30	p31	p16	p17	p18	p19	p20	p21	p22	p23	p8	p9	p10	p11	p12	p13	p14	p15	p0	p1	p2	p3	p4	p5	p6	p7

UYVY 4:2:2

yb7	yb6	yb5	yb4	yb3	yb2	yb1	yb0	v7	v6	v5	v4	v3	v2	v1	v0	ya7	ya6	ya5	ya4	ya3	ya2	ya1	ya0	u7	u6	u5	u4	u3	u2	u1	u0

YUYV 4:2:2

v7	v6	v5	v4	v3	v2	v1	v0	yb7	yb6	yb5	yb4	yb3	yb2	yb1	yb0	u7	u6	u5	u4	u3	u2	u1	u0	ya7	ya6	ya5	ya4	ya3	ya2	ya1	ya0

Methods of color translation used for Blts

	1bpp src	**8bpp src**	**15bpp src**	**16bpp src**	**24bpp src**	**32bpp src**	**YUV src**
8bpp dst	color registers	direct or palette	not supported	not supported	not supported	not supported	not supported
15bpp dst	color registers	not supported	direct	lsb removal	lsb removal	lsb removal, alpha dropped	YUV => RGB
16bpp dst	color registers	not supported	msb duplication	direct	lsb removal	lsb removal, alpha dropped	YUV => RGB
24bpp dst	color registers	not supported	msb duplication	msb duplication	direct	direct, alpha dropped	YUV => RGB
32bpp dst	color registers	not supported	msb duplication, zero alpha	msb duplication, zero alpha	rgb direct, zero alpha	direct	YUV => RGB zero alpha

srcFormat

Bit	**Description**

13:0	Source Stride in bytes
15:14	RESERVED
19:16	Source color format: 1, 8, 16, 24, 32 bpp RGB, YUYV422, UYVY422
20	Host port byte swizzle (1=enable)
21	Host port word swizzle (1=enable)
23:22	Source packing
31:24	RESERVED

dstFormat

Bit	Description
13:0	Destination Stride in bytes
15:14	RESERVED
18:16	Destination bits per pixel: 8, 15, 16, 24, or 32
31:19	RESERVED

srcFormat [19:16]	Source Format
0	1 bpp mono
1	8 bpp palettized
3	16 bpp RGB
4	24 bpp RGB
5	32 bpp RGB
8	packed 4:2:2 YUYV
9	packed 4:2:2 UYVY

dstFormat [18:16]	Destination Bpp
1	8
3	16
4	24
5	32

srcFormat[23:22]	Packing	Stride calculation
0	Use stride register	srcFormat[13:0]
1	Byte packed	ceil(src_width * src_bpp/8)
2	Word packed	ceil(src_width * src_bpp/16)*2
3	Double-word packed	ceil(src_width * src_bpp/32)*4

clip0Min, clip0Max, clip1Min, and clip1Max Registers

The clip registers define the maximum and minimum x & y values of pixel that can be written in the destination pixel map. There are two sets of clip registers, however, only one set is used at a time, as determined by the clip select bit in the **command** register.

Clipping is inclusive of the minimum values, and exclusive of the maximum values. Thus if the clip select bit is zero, only pixels with x values in the range [clip0Min_x, clip0Max_x) and y values in the range [clip0Min_y, clip0Max_y) will be written.

clip0Min

Bit	Description
11:0	x minimum clip when clip select is 0

15:12	RESERVED
27:16	y minimum clip when clip select is 0
31:28	RESERVED

clip0Max

Bit	Description
11:0	x maximum clip when clip select is 0
15:12	RESERVED
27:16	y maximum clip when clip select is 0
31:28	RESERVED

clip1Min

Bit	Description
11:0	x minimum clip when clip select is 1
15:12	RESERVED
27:16	y minimum clip when clip select is 1
31:28	RESERVED

clip1Max

Bit	Description
11:0	x maximum clip when clip select is 1
15:12	RESERVED
27:16	y maximum clip when clip select is 1
31:28	RESERVED

colorkey Registers

These registers define the range of colors that will be transparent when color keying is enabled.

Different ROPs are selected for each pixel depending the result of that pixels colorkey test. A source pixel passes the colorkey test if it is within the inclusive range defined by the **srcColorkeyMin** and **srcColorkeyMax** registers. A destination pixel passes the colorkey test if it is within the inclusive range defined by the **dstColorkeyMin** and **dstColorkeyMax** registers.

For Pixels with 8bpp formats, the color indices are compared directly. For pixels with 16, 24, or 32bpp formats, each color channel (R, G, and B) is compared separately, and each channel must pass for the colorkey test to be passed. In the 32bpp format, the upper 8 bits are ignored during colorkey testing. Source colorkeying cannot be enabled if the source format is 1 bpp.

If colorkeying is disabled for the source or destination surfaces, that colorkey test is failed.

For further information on ROP selection by the colorkey test results, see the description of the ROP register.

The colorkey test uses the following formula:
pass = (((color>=colorkey_min) && (color<=colorkey_max)) && colorkey_enable)

srcColorkeyMin

Bit	Description
23:0	minimum color key value for source pixels
31:24	RESERVED

srcColorkeyMax

Bit	Description
23:0	maximum color key value for source pixels
31:24	RESERVED

dstColorkeyMin

Bit	Description
23:0	minimum color key value for destination pixels
31:24	RESERVED

dstColorkeyMax

Bit	Description
23:0	maximum color key value for destination pixels
31:24	RESERVED

rop Register

This is a set of ternary ROPs used to determine how the source, destination, and pattern pixels will be combined. The default ROP, ROP0 is stored in the command register. Which of the four ROPs will be used is determined on a per-pixel basis, based on the results of the source and destination colorkey tests, as shown in the following table:

Source Color Key Test	Destination Color Key Test	ROP
Fail	Fail	ROP 0
Fail	Pass	ROP 1
Pass	Fail	ROP 2
Pass	Pass	ROP 3

rop

Bit	Description
7:0	ROP 1
15:8	ROP 2
23:16	ROP 3

lineStyle register

The **lineStyle** register specifies how lines will be drawn.

The bit pattern used for line stippling can be set to repeat every 1-32 bits, as set by the bit-mask size part of this register. The bit-mask size entry gives the number of bits *minus one* that will be used from the **lineStipple** register. Thus, if you want to use 2 bits to represent a dashed line, you would set the bit-mask size to 1.

Each bit from the **lineStipple** register will determine the color or transparency of from 1-256 pixels. The repeat count determines the number of pixels along the line that will be drawn (or skipped) for each bit in the line pattern register. The number of pixels associated with each bit of the line pattern *minus one* must be written to the repeat count entry.

The start position give the offset within the line pattern register for the first pixel drawn in a line. It consists of an integer index of the current bit in the line pattern, and a fractional offset that will determine the number of pixels that will be drawn using that bit of the pattern. The number of pixels drawn using the initial bit in the line pattern will equal the repeat count (i.e. the repeat count entry+1) minus the fractional part of the start position. The bit positions within the **lineStipple** registers are numbered starting with the lsb at 0, going up to the msb at 31.

It is illegal to set the integer part of the stipple position to be greater than the bit-mask size. It is illegal to set the fractional part to be greater than the repeat count. If either part of the stipple position is too large, the behavior of the line drawing engine is undefined.

Writing the **lineStyle** register will cause the stipple position to be loaded from the register. If the **lineStyle** register is not written to between the execution of two line commands, the stipple position at the start of the new line will be whatever if was after the completion of the last line. If the **lineStyle** register is read while the 2D engine is idle, the stipple position read will always be that which will be used in the next line operation - thus, if the **lineStyle** register has been written since the last stippled line was drawn the value written will be returned, otherwise the value that remained after the last stippled line will returned. Reading the **lineStyle** register while the 2D engine is not idle will return an indeterminable value for the stipple position.

In the following examples,. ‘x’ represents a pixel colored with **colorFore**, ‘o’ represents a pixel colored with **colorBack** or that is transparent. ‘_S_’ Shows that the line engine is starting at bit 0 in the **lineStipple** register. ‘_’ shows that the line engine is using a new bit from the **lineStipple** register.

Example

Say the bit-mask size is set to 6 (thus, the entry in the register is 5) and the line pattern is:
lineStipple <= 010111b

The pixel pattern that will be repeated is:

repeat_count	repeating pixel pattern
1	x_x_x_o_x_o_S_x_x_x_o_x_o
2	xx_xx_xx_oo_xx_oo_S_xx_xx_xx_oo_xx_oo
3	xxx_xxx_xxx_ooo_xxx_ooo_S_xxx_xxx_xxx_ooo_xxx_ooo

Example

Say the repeat count is 5 (the register entry is 4), the integer part of the start position is 7, and the fractional part of the start position is 2. The color of the first 3 pixels drawn for the line will be determined by bit 7 in the line pattern register, the next 5 pixels will be determined by bit 8, and so on.

lineStyle <= 07020904h
lineStipple <= 1010110111b

pixels generated, where x=**colorFore** and o=**colorBack**:

xxx_ooooo_xxxxx_S_xxxxx_xxxxx_xxxxx_ooooo_xxxxx_xxxxx_ooooo_xxxxx_ooooo_xxxxx_S

Pseudo code for line pixel generation

Here is the pseudo-code for determining the color of pixels generated by the line engine:

```
<bit_position> = <start_position_integer>
<pixel_position> = <start_position_fraction>

while (<need_another_pixel>) {
   if ( <line_pattern> & (1 << <bit_position>) ) {
         <new_pixel_color> = <colorFore>
   } else {
         if (<transparent>) {
            <new_pixel_color> = <transparent>
         } else {
            <new_pixel_color> = <colorBack>
         }
   }

if ( <pixel_position> == <repeat_count> ) {
         <pixel_position> = 0
         if (<bit_position> == <bit_mask_size>) {
                  <bit_position> = 0;
         } else {
                  <bit_position> = <bit_position> + 1
         }
   } else {
         <pixel_position> = <pixel_position> +1
   }

}
```

lineStyle

Bit	Description
7:0	Repeat count
12:8	Stipple size

15:13	RESERVED
23:16	Start position - fractional part
28:24	Start position - integer part
31:29	RESERVED

lineStipple Register

The line bit-mask register contains a mask that determines how lines will be drawn. Bits that are ones will be drawn with the color in the **colorFore** register. Bits that are zeros will be filled with the color in the **colorBack** register, or will not be filled, depending on the 'transparent' bit in the command register. The pattern in the bit mask can be set to repeat every 1-32 bits, as set by the bit-mask size part of the line style register. If the bit-mask size is set to less than 31, some of the bits of the line bit-mask will not be used, starting with the most-significant bit. For example, if the bit-mask size is set to 7, bits 0-7 of the **lineStipple** register will contain the line bit-mask.

lineStipple

Bit	Description
31:0	Line bit-mask

bresenhamError registers

These registers allows the user to specify the initial Bresenham error terms used when performing line drawing, polygon drawing, and stretch blts. The Bresenham error terms are signed values.

Bit 31 of each registers determines whether or not the error term given in the lower bits will be used. If this bit is 0, the line and stretch blt engines will generate the initial error term automatically. If the bit is set to 1, the error term given in bits 16-0 will be used. If a bresenham error register is used, the register should be written with bit[31] set to 0 after completion of the operation, so that subsequent operations will not be affected.

bresError0 can be used to set the initial error value for lines, for the left edge of a polygon, and for blt stretching along the y-axis.

bresError1 can be used to set the initial error value for the right edge of a polygon, and for blt stretching along the x-axis.

bresError0

Bit	Description
15:0	Signed Bresenham error term for stretch blt y, lines, and left polygon edges
30:17	RESERVED
31	Use the error term given in bits 16-0

bresError1

Bit	Description
15:0	Signed Bresenham error term for stretch blt x and right polygon edges
30:17	RESERVED
31	Use the error term given in bits 16-0

Launch Area

Screen-to-screen Blt Mode

Writing the launch area while in screen-to-screen blt mode results in a rectangle being copied from one area of display memory to another. The position of the source rectangle is given by the write to the launch area. The write to the launch area will be used to fill the **srcXY** register.

screenBltLaunch

Bit	Description
12:0	X position of the source rectangle
15:13	RESERVED
28:16	Y position of the source rectangle
31:29	RESERVED

Screen-to-screen Stretch Blt Mode

Writing the launch area while in screen-to-screen blt mode results in pixels being copied from rectangle in display memory to another of a different size. The write to the launch area will be used to fill the **srcXY** register. The x and y direction bits do not apply to stretch blits. I.e., only top-down, left-to-right stretch blits can be done.

stretchBltLaunch

Bit	Description
12:0	X position of the source rectangle
15:13	RESERVED
28:16	Y position of the source rectangle
31:29	RESERVED

Host-to-screen Blt Mode

In host-to-screen blt mode, writes to the launch area should contain packed pixels to be used as source data. When performing a host-to-screen blt, the blt engine does not generate source addresses. However, it is still necessary for the driver to specify the **srcFormat**, in order for the blt engine to determine how the source data is packed. The driver must also write the **srcXY** register in order to specify the first byte or bit to use from the first dword. In monochrome source mode, the 5 lsbs will specify the initial bit. In all other modes, the 2 lsbs of **srcXY** will specify the initial byte of the initial span. The alignment of the first pixel of each span after the first is determined by adding the source stride (from the **srcFormat** register) to the alignment of the previous span.

If more data is written to the launch area than is required for the host blt specified, the extra data will be discarded, or may be used in the following host blt, if it is requested while the 2D is operating on the first hblt. If too little data is written to the launch area, the hblt will be aborted, and pixels on an incomplete span at the end of the host blt may or may not be drawn.

Host Blt Example 1

In this example, the driver is drawing text to a 1024x768x16bpp screen using monochrome bitmaps of various widths. The monochrome data is packed, with each row byte aligned. First, it sets up the necessary registers before giving the data specific to the first blt:

colorBack <= the background color

colorFore <= the foreground color

dstXY <= the starting position of the first character

dstBaseAddr <= base address of the primary surface

clip0Min <= 0x00000000

clip0Max <= 0xFFFFFFFF

command <= SRC_COPY || HOST_BLT_MODE = 0xCC000003

dstFormat <= 0x00030800

srcFormat <= 0x00400000

The command mode is set to host-to-screen blt, with all other features disabled. Since colorkeying is disabled, only ROP0 is needed. The format register sets the host format to unswizzled monochrome, using byte-packing. This means that the stride will not have to be set for each blt, but will be set to the number of bytes required to store the number of pixels in the source width (Since this is not a stretch blt, the source width equals the destination width, as set later in the **dstSize** register). The clip registers are set such that the results will not be clipped. Although this is a host to screen blt, the **srcXY** register must be set in order to specify the initial alignment of the bitmask. For this example, the source data begins with the lsb of the first dword of host data, so the **srcXY** register is set to zero.

Now, the driver is ready to start the first blt. It will blt a 11x7 pixel character.

dstSize <= 0x0007000B

srcXY <= 0x00000000

launch <= 0xc0608020

launch <= 0xC460C060

launch <= 0x3B806EC0

launch <= 0x00001100

Host Blt Example 2

In this example, the driver is drawing a pixel map

colorBack <= the background color

colorFore <= the foreground color

dstXY <= the starting position of the first character

clip0Min <= 0x00000000

clip0Max <= 0xFFFFFFFF

command <= SRC_COPY || HOST_BLT_MODE = 0xCC000003

srcFormat <= 0x00240000

The command mode is set to host-to-screen blt, with all other features disabled. Since colorkeying is disabled, only ROP0 is needed. The format register sets the host format to unswizzled monochrome, using byte-packing. This means that the stride will not have to be set for each blt, but will be set to the number of bytes required to store the number of pixels in the source width (Since this is not a stretch blt, the source width equals the destination width, as set later in the **dstSize** register). The clip registers are set such that the results will not be clipped. Although this is a host to screen blt, the **srcXY** register must be set in order to specify the initial alignment of the bitmask. For this example, the source data begins with the lsb of the first dword of host data, so the **srcXY** register is set to zero.

Now, the driver is ready to start the first blt. It will blt a 11x7 pixel character.

dstXY <= 0x0007000B

srcXY <= 0x00000000

launch <= 1st 2 rows

launch <= 2nd 2 rows

launch <= 3rd 2 rows

launch <= last row

hostBltLaunch

Bit	Description
31:0	Source pixel data

Host-to-screen Stretch Blt Mode

Writing the launch area while in host-to-screen blt mode results in the pixels written to the launch area being stretched onto the destination rectangle. Pixel data for Host-to-screen stretch blts is written just as for non-stretched host-to-screen blts, except when the destination height differs from the source height. In this case, the host must replicate or decimate the source spans to match the number of destinations spans required.

hostStretchLaunch

Bit	Description
31:0	Source pixel data

Rectangle Fill Mode

Rectangle fill mode is similar to screen-to-screen blt mode, but in this mode, the **colorFore** register is used as source data rather than data from display memory. The size of the rectangle is determined by the

dstSize register. The write to the launch area gives the position of the destination rectangle, which is used to fill the **dstXY** register.

rectFillLaunch

Bit	Description
12:0	X position of the destination rectangle
15:13	RESERVED
28:16	Y position of the destination rectangle
31:29	RESERVED

Line Mode

Writing the launch area while in line mode will write the launch data to the **dstXY** register and draw a line from **srcXY** to **dstXY**. After the line has been drawn, **dstXY** is copied to **srcXY**. In line mode, all pixels in the line will be drawn (as specified by the line style register), including both the start and endpoint.

The ROP used for lines can use the pattern and the destination, but not source data. **colorFore** will be used in the ROP in place of source data. Source colorkeying must be turned off, destination colorkeying is allowed.

Line drawing example

srcXY <= 0x00020003 // line start-point = (3, 2)

lineStipple <= 0x00000006 // bit mask is 110 binary

lineStyle <= 0x02010202 // start position = 2 1/3, repeat count = 2, bit-mask size=2

colorBack <= BLACK

colorFore <= GREY

command <= LINE_MODE || OPAQUE

launch <= 0x000c0016 // line end-point = (22,12)

The line drawn will appear as shown below:

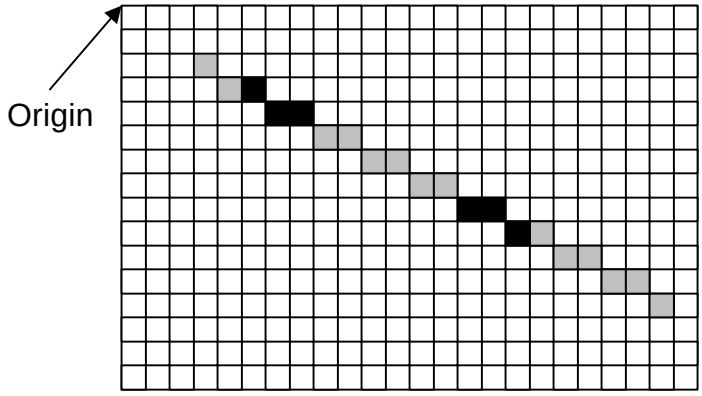


Figure 1

lineLaunch

Bit	Description
12:0	X position of the line endpoint
15:13	RESERVED
28:16	Y position of the line endpoint
31:29	RESERVED

Polyline Mode

Writing the launch area while in line mode will write the launch data to the **dstXY** register and draw a line from **srcXY** to **dstXY**. After the line has been drawn, **dstXY** is copied to **srcXY**. In polyline mode, the endpoint of the line (the pixel at dstXY) will not be written. This ensures that each pixel in a non-overlapping polyline will be written only once.

The ROP used for lines can use the pattern and the destination, but not source data. **colorFore** will be used in the ROP in place of source data. Source colorkeying must be turned off, destination colorkeying is allowed.

polylineLaunch

Bit	Description
12:0	X position of the line endpoint
15:13	RESERVED
28:16	Y position of the line endpoint
31:29	RESERVED

Polygon Fill Mode

The polygon fill mode can be used to draw simple polygons. A polygon may be drawn using the method described below if no horizontal span intersects more than two non-horizontal polygon edges. Polygons are drawn by first determining the top vertex - that is the vertex with the lowest y coordinate. The

coordinates of this vertex should be written to the **srcXY** register. If multiple vertices share the lowest y coordinate, any vertex with the lowest y coordinate may be used as the starting point. If **command**[8] is set when the **command** register is written when **command**[3:0] indicates polygon mode, the value in the **srcXY** register will be copied to the **dstXY** register. The value in the **srcXY** register determines the starting point for the left side of the polygon, while the value in the **dstXY** register determines the starting point for the right side of the polygon. If bit[8] of the command register is not set, the starting position of the right side of the polygon can be set by writing to the **dstXY** register.

Once the starting vertex is set, as well as the desired colors, ROP, pattern, and options for the polygon fill, the polygon can be drawn by writing polygon vertices to the launch area. When multiple vertices share the lowest y coordinate, the starting vertex chosen will determine which of those vertices are on the 'right' edge of the polygon and which are on the 'left' edge. Pixels with the same y value as the starting point are on the left edge if they are to the left of the starting point.

For optimum performance, software should determine the leftmost and rightmost of all vertices that share the lowest y coordinate. The coordinates of the leftmost vertex should be written to **srcXY** and the coordinates of the rightmost vertex should be written to **dstXY**. When the **command** register is written, **command**[8] (the 'start command' bit) should be low.

In Polygon fill mode, polygon vertices should be written to the launch area in order of increasing y value. Whenever 2 vertices share the same y value, the leftmost vertex *must* be written first. The driver should keep track of the last y value sent for the left and right sides. If the y value for the last vertex sent for the left side is *less than or equal to* the last y value sent for the right side, the next vertex on the left side should be written to the launch area. Otherwise, the next vertex for the right side should be written to the launch area.

The ROP used for filling polygons can use the pattern and the destination, but not source data. **colorFore** will be used in the ROP in place of source data. Source colorkeying must be turned off, destination colorkeying is allowed.

Pixels that are on the line that forms the left edge of the polygon will be drawn. Pixels that fall on the line that forms the right edge of the polygon will not be drawn. For Horizontal edges, pixels on a horizontal polygon edge that is on the 'top' of the polygon (i.e. above the edge is outside the polygon and below the edge is inside the polygon) will be drawn, while pixels on a horizontal polygon edge that is on the bottom of the polygon will not be drawn.

Polygon drawing example

As an example of polygon drawing, say we are drawing the polygon shown in figure 2. Traversing the vertex list in counterclockwise order gives the following list of vertices:

(4, 1) (2, 4) (3, 6) (1, 6) (2,8) (5, 11) (8,8) (13,8) (11,6) (11,3) (10,1)

Figures 2a through 2m show the steps in drawing the polygon. Filled circles are vertices of the left polygon edge. Open circles are vertices of the right polygon edge. Pixels that are drawn at the end of each step are shaded in the figures.

The polygon engine keeps track of four vertices at a time. The top vertex of the current left polygon edge (L0), the bottom vertex of the current left polygon edge (L1), the top vertex of the current right polygon edge (R0), and the bottom vertex of the current right polygon edge (R1). The values of these variables at each step in drawing the polygon are shown in the figures. The arrows in the figures indicate when a variable changes between the start of the step and the end of pixel filling for that step.

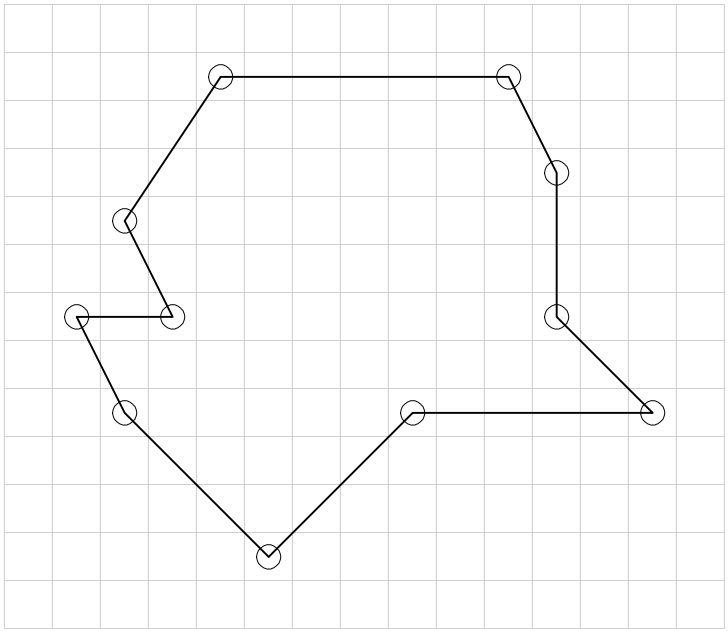


Figure 2

First, all required registers must be written, including the **dstFormat** register to specify the drawing surface, color or pattern registers, and the command register. Write the coordinates of the starting vertex (4, 1) to the **srcXY** register:

srcXY <= 0x00010004

command <= POLYGON_MODE || INITIATE_COMMAND

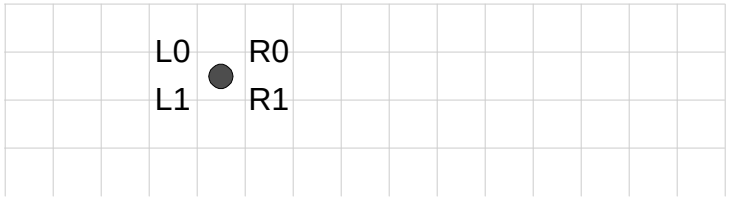


Figure 2a

R1.y>=L1.y, so we have to write the next vertex for the left edge (2, 4):

launch <= 0x00040002

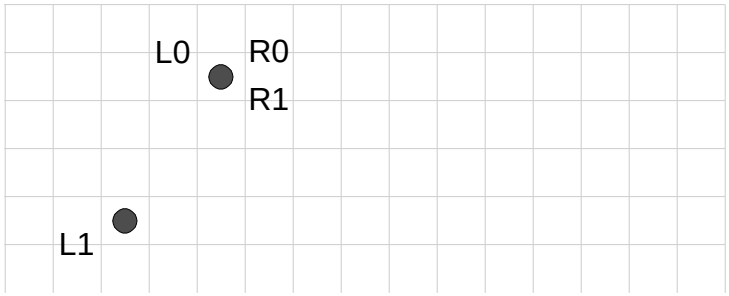


Figure 2b

R1.y<L1.y, so we write the next vertex for the right edge (10, 2). The drawing engine now has edges for both the left and right edges. So, it will draw all spans up to min(R1.y, L1.y). Because R1.y=R0.y, no pixels will be drawn, but R0 will be updated to vertex R1:

launch <= 0x0001000a

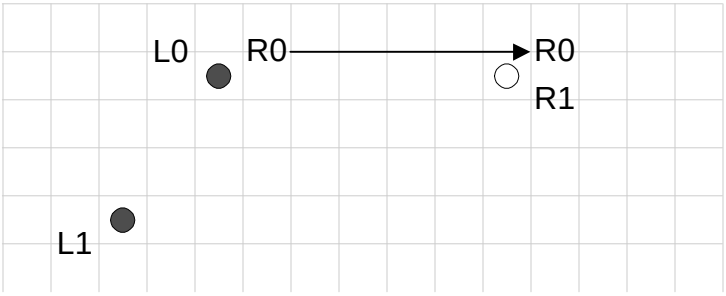


Figure 2c

R1.y<L1.y, so we again write the next vertex on the right polygon edge (11, 3). Pixels on all spans from max(L0.y, R0.y) to min(L1.y, R1.y)-1 will be drawn, as shown below. Because R1.y<L1.y, R0 is updated to R1.

launch <= 0x0003000b

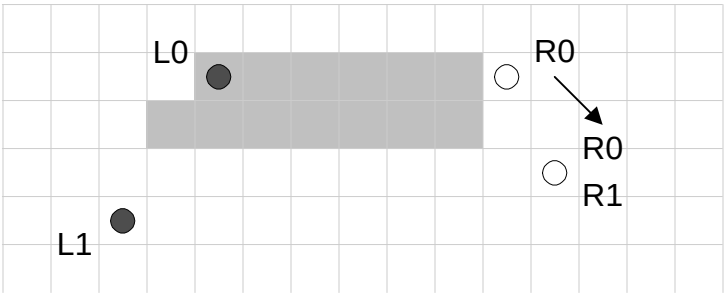


Figure 2d

R1.y<L1.y, so we write the next vertex on the right edge (11, 6). Again, pixels on all spans from max(L0.y, R0.y) to min(L1.y, R1.y)-1 will be drawn. This time R1.y>L1.y, however, so L0 is updated to L1.

launch <= 0x0006000b

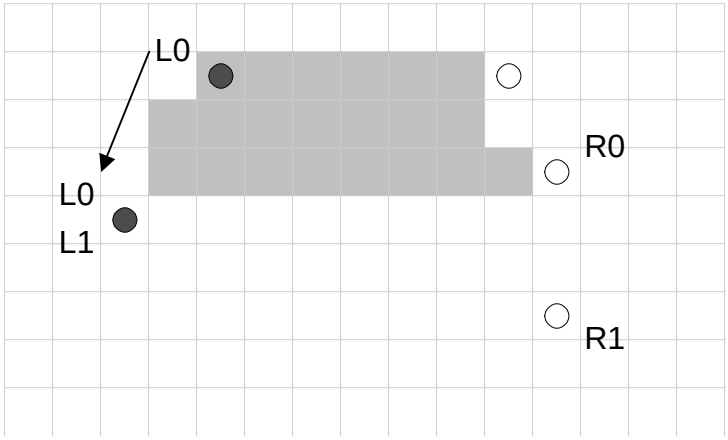


Figure 2e

R1.y>=L1.y, so we write the next vertex on the left edge (3, 6). L1.y=R1.y, so R0 is updated to R1 and L0 is updated to L1.

launch <= 0x00060003

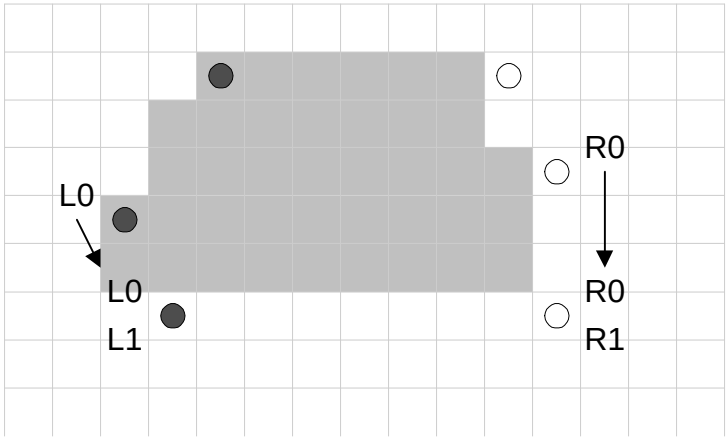


Figure 2f

R1.y>=L1.y, so we write the next vertex on the left edge (1, 6). L1.y=R1.y, so R0 is updated to R1 and L0 is updated to L1. R1 did not change, so updating R0 to R1 has no effect.

launch <= 0x00060001

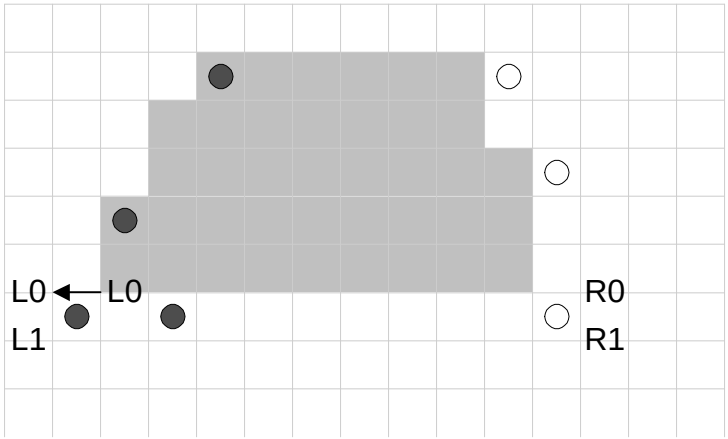


Figure 2g

R1.y>=L1.y, so we again write the next vertex on the left edge (2, 8). L1.y>R1.y, so R0 is updated to R1, again with no effect.

launch <= 0x00080002

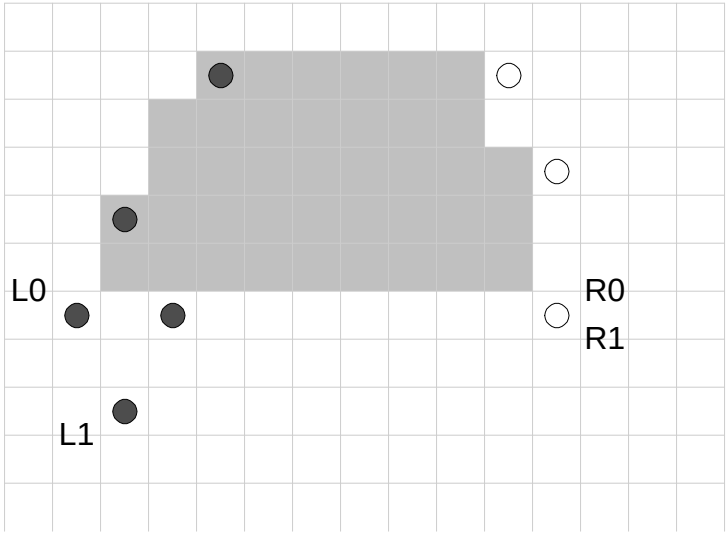


Figure 2h

R1.y<L1.y, so we write the next vertex on the right edge (11, 8). L1.y=R1.y, so R0 is updated to R1, and L0 is updated to L1.

launch <= 0x0008000b

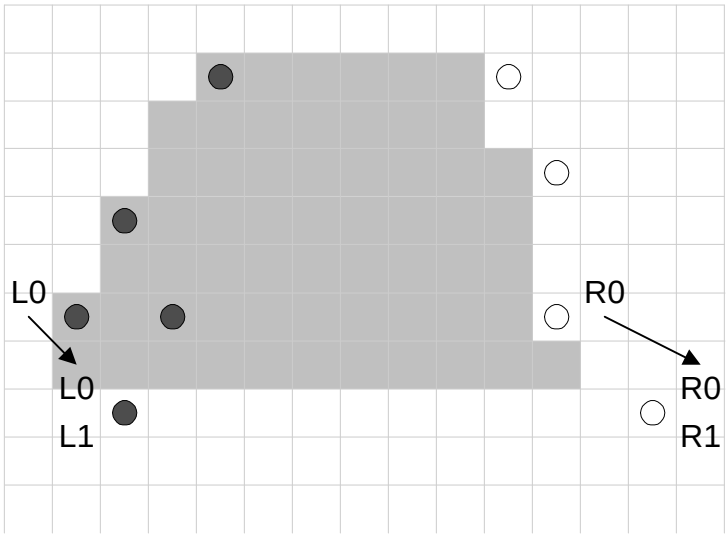


Figure 2i

R1.y>=L1.y, so we write the next vertex on the left edge (5, 11). L1.y>R1.y, so R0 is updated to R1.

launch <= 0x000b0005

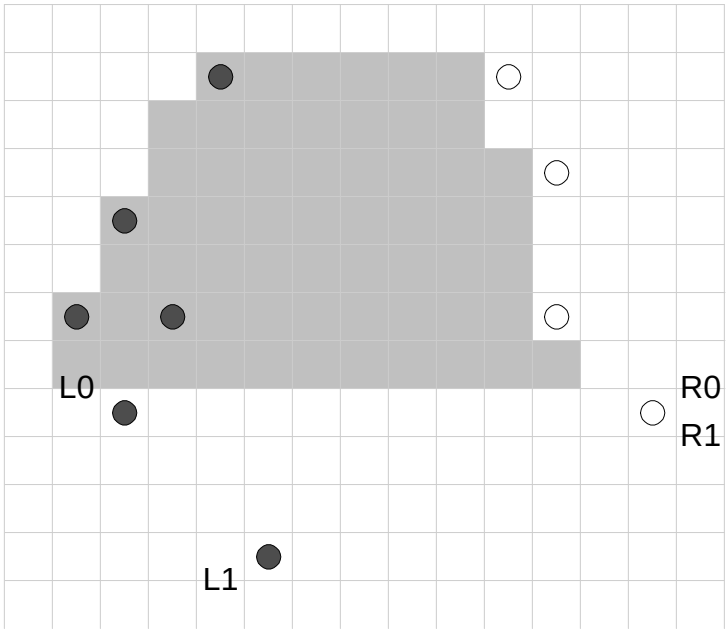


Figure 2j

R1.y<L1.y, so we write the next vertex on the right edge (8, 8). L1.y>R1.y, so R0 is updated to R1, but no pixels are drawn.

launch <= 0x00080008

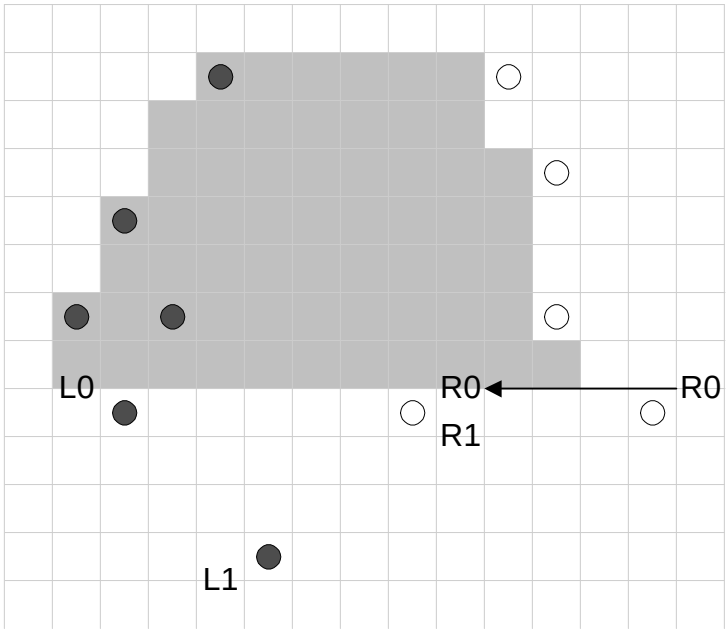


Figure 2k

R1.y<L1.y, so we write the next vertex on the right edge. This is the final vertex in the polygon, which doesn't have a horizontal span at the bottom, so this vertex is the same as the last vertex for the left edge (5, 11). L1.y=R1.y, so R0 is updated to R1, and L0 is updated to L1. No pixels on the final span are drawn (this would be true even if L1.x did not equal R1.x). If the launch area is written again before any registers are written the polygon engine will begin a new polygon starting at (5,11).

launch <= 0x000b0005

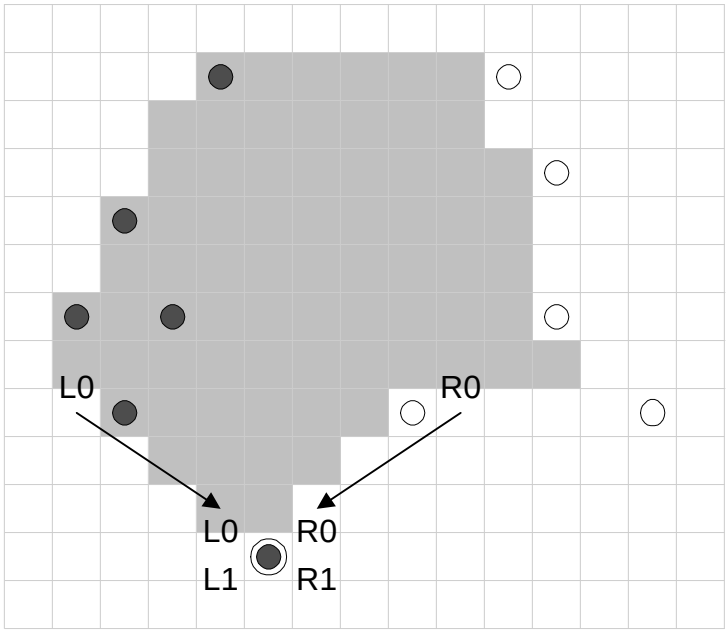


Figure 2m

polygonLaunch

Bit	Description
12:0	X position of a polygon vertex
15:13	RESERVED
28:16	Y position of a polygon vertex
31:29	RESERVED

Miscellaneous 2D

Write Sgram Mode Register

Executing this command causes the value in srcBaseAddr[10:0] to be set as the sgram mode register via a special bus cycle in the memory controller.

SGRAM mode register

Bit	Description
2:0	burst length
3	burst type (0=sequential, 1=interleave)
6:4	CAS latency
8:7	test mode

9	write burst length (0=burst, 1=single bit).
10	sgram-defined.

Write Sgram Color Register

Executing this command causes the value in srcBaseAddr[31:0] to be set as the sgram color register via a special bus cycle in the memory controller. Since H3 has a 128-bit wide bus, the register is replicated across the four sets of sgram memories.

Write Sgram Mask Register

Executing this command causes the value in srcBaseAddr[31:0] to be set as the sgram mask register via a special bus cycle in the memory controller. Since H3 has a 128-bit wide bus, the register is replicated across the four sets of sgram memories.

6. 3D Memory Mapped Register Set

Memory Base 0: Offset 0x0200000

Register Name	Address	Reg Num	Bits	Chip	R/ W	Sync? /Fifo?	Description
status	0x000(0)	0x0	31:0	FBI	R	No / n/a	Banshee Status
intrCtrl	0x004(4)	0x1	31:0	FBI	R/W	No / No	Interrupt Status and Control
nopCMD	0x120(288)	0x48	0	FBI+TREX[%]	W	Yes/Yes	Execute NOP command

nopCMD Register

Writing any data to the **nopCMD** register executes the NOP command. Executing a NOP command flushes the graphics pipeline. The lsb of the data value written to **nopCMD** is used to optionally clear the **fbiPixelsIn**, **fbiChromaFail**, **fbiZfuncFail**, **fbiAfuncFail**, and **fbiPixelsOut** registers. Writing a '1' to the lsb of **nopCMD** will clear the aforementioned registers. Writing a '0' to the lsb of nopCMD will not modify the values of the aforementioned registers.

Bit	Description
0	Clear **fbiPixelsIn**, **fbiChromaFail**, **fbiZfuncFail**, **fbiAfuncFail**, and **fbiPixelsOut** registers (1=clear registers)

lfbMode Register

The **lfbMode** register controls linear frame buffer accesses.

Bit	Description
3:0	Linear frame buffer write format (see table below)
5:4	Reserved

7:6	Reserved
8	Enable Banshee pixel pipeline-processed linear frame buffer writes (1=enable)
10:9	Linear frame buffer RGBA lanes (see tables below)
11	16-bit word swap linear frame buffer writes (1=enable)
12	Byte swizzle linear frame buffer writes (1=enable)
13	LFB access Y origin (0=top of screen is origin, 1=bottom of screen is origin)
14	Linear frame buffer write access W select (0=LFB selected, 1=**zacolor**[15:0]).
15	Reserved
16	Reserved

The following table shows the supported Banshee linear frame buffer write formats:

Value	Linear Frame Buffer Write Format
	16-bit formats
0	16-bit RGB (5-6-5)
1	16-bit RGB (x-5-5-5)
2	16-bit ARGB (1-5-5-5)
3	Reserved
	32-bit formats
4	24-bit RGB (x-8-8-8)
5	32-bit ARGB (8-8-8-8)
7:6	Reserved
11:8	Reserved
12	16-bit depth, 16-bit RGB (5-6-5)
13	16-bit depth, 16-bit RGB (x-5-5-5)
14	16-bit depth, 16-bit ARGB (1-5-5-5)
15	16-bit depth, 16-bit depth

When accessing the linear frame buffer, the cpu accesses information from the starting linear frame buffer (LFB) address space (see section 4 on Banshee address space) plus an offset which determines the <x,y> coordinates being accessed. Bits(3:0) of **lfbMode** define the format of linear frame buffer writes.

When writing to the linear frame buffer, **lfbMode** bit(8)=1 specifies that LFB pixels are processed by the normal Banshee pixel pipeline -- this implies each pixel written must have an associated depth and alpha value, and is also subject to the fog mode, alpha function, etc. If bit(8)=0, pixels written using LFB access bypass the normal Banshee pixel pipeline and are written to the specified buffer unconditionally and the values written are unconditionally written into the color/depth buffers except for optional color dithering [depth function, alpha blending, alpha test, and color/depth write masks are all bypassed when bit(8)=0]. If bit(8)=0, then only the buffers that are specified in the particular LFB format are updated. Also note that if **lfbMode** bit(8)=0 that the color and Z mask bits in **fbzMode**(bits 9 and 10) are ignored for LFB writes. For example, if LFB modes 0-2, or 4 are used and bit(8)=0, then only the color buffers are updated for LFB writes (the depth buffer is unaffected by all LFB writes for these modes, regardless of the status of the Z-mask bit **fbzMode** bit 10). However, if LFB modes 12-14 are used and bit(8)=0, then both the color and depth buffers are updated with the LFB write data, irrespective of the color and Z mask bits in **fbzMode**. If LFB mode 15 is used and bit(8)=0, then only the depth buffer is updated for LFB writes (the color buffers are unaffected by all LFB writes in this mode, regardless of the status of the color mask bits in **fbzMode**).

If **lfbMode** bit(8)=0 and a LFB write format is selected which contains an alpha component (formats 2, 5, and 14) and the alpha buffer is enabled, then the alpha component is written into the alpha buffer. Conversely, if the alpha buffer is not enabled, then the alpha component of LFB writes using formats 2, 5, and 14 when bit(8)=0 are ignored. Note that anytime LFB formats 2, 5, and 14 are used when bit(8)=0 that blending and/or chroma-keying using the alpha component is not performed since the pixel-pipeline is bypassed when bit(8)=0.

If **lfbMode** bit(8)=0 and LFB write format 14 is used, the component that is ignored is determined by whether the alpha buffer is enabled -- If the alpha buffer is enabled and LFB write format 14 is used with bit(8)=0, then the depth component is ignored for all LFB writes. Conversely, if the alpha buffer is disabled and LFB write format is used with bit(8)=0, then the alpha component is ignored for all LFB writes.

If **lfbMode** bit(8)=1 and a LFB write access format does not include depth or alpha information (formats 0-5), then the appropriate depth and/or alpha information for each pixel written is taken from the **zaColor** register. Note that if bit(8)=1 that the LFB write pixels are processed by the normal Banshee pixel pipeline and thus are subject to the per-pixel operations including clipping, dithering, alpha-blending, alpha-testing, depth-testing, chroma-keying, fogging, and color/depth write masking.

Bits(10:9) of **lfbMode** specify the RGB channel format (color lanes) for linear frame buffer writes. The table below shows the Banshee supported RGB lanes:

Value	RGB Channel Format
0	ARGB
1	ABGR
2	RGBA
3	BGRA

Bit(11) of **lfbMode** defines the format of 2 16-bit data types passed with a single 32-bit writes. For linear frame buffer formats 0-2, two 16-bit data transfers can be packed into one 32-bit write -- bit(11) defines which 16-bit shorts correspond to which pixels on screen. The table below shows the pixel packing for packed 32-bit linear frame buffer formats 0-2:

lfbMode bit(11)	Screen Pixel Packing
0	Right Pixel(host data 31:16), Left Pixel(host data 15:0)
1	Left Pixel(host data 31:16), Right Pixel(host data 15:0)

For linear frame buffer formats 12-14, bit(11) of **lfbMode** defines the bit locations of the 2 16-bit data types passed. The table below shows the data packing for 32-bit linear frame buffer formats 12-14:

lfbMode bit(11)	Screen Pixel Packing
0	Z value(host data 31:16), RGB value(host data 15:0)
1	RGB value(host data 31:16), Z value(host data 15:0)

For linear frame buffer format 15, bit(11) of **lfbMode** defines the bit locations of the 2 16-bit depth values passed. The table below shows the data packing for 32-bit linear frame buffer format 15:

lfbMode bit(11)	Screen Pixel Packing
0	Z Right Pixel(host data 31:16), Z Left Pixel(host data 15:0)
1	Z left Pixel(host data 31:16), Z Right Pixel(host data 15:0)

Note that bit(11) of **lfbMode** is ignored for linear frame buffer writes using formats 4 or 5.

Bit(12) of **lfbMode** is used to enable byte swizzling. When byte swizzling is enabled, the 4-bytes within a 32-bit word are swizzled to correct for endian differences between Banshee and the host CPU. For little endian CPUs (e.g. Intel x86 processors) byte swizzling should not be enabled, however big endian CPUs (e.g. PowerPC processors) should enable byte swizzling. For linear frame buffer writes, the bytes within a word are swizzled prior to being modified by the other control bits of **lfbMode**. When byte swizzling is enabled, bits(31:24) are swapped with bits(7:0), and bits(23:16) are swapped with bits(15:8).

Very Important Note: The order of swapping and swizzling operations for LFB writes is as follows: byte swizzling is performed first on all incoming LFB data, as defined by **lfbMode** bit(12) and irrespective of the LFB data format. After byte swizzling, 16-bit word swapping is performed as defined by **lfbMode** bit(11). Note that 16-bit word swapping is never performed on LFB data when data formats 4 and 5 are used. Also note that 16-bit word swapping is performed on the LFB data that was previously optionally swapped. Finally, after both swizzling and 16-bit word swapping are performed, the individual color channels are selected as defined in **lfbMode** bits(10:9). Note that the color channels are selected on the LFB data that was previously swizzled and/or swapped

Bit(13) of **lfbMode** is used to define the origin of the Y coordinate for all linear frame buffer writes when the pixel pipeline is bypassed (**lfbMode** bit(8)=0). Note that bit(13) of **lfbMode** does not affect rendering operations (FASTFILL and TRIANGLE commands) -- bit(17) of **fbzMode** defines the origin of the Y coordinate for rendering operations. Note also that if the pixel pipeline is enabled for linear frame buffer writes (**lfbMode** bit(8)=1), then **fbzMode** bit(17) is used to determine the location of the Y origin. When cleared, the Y origin (Y=0) for all linear frame buffer accesses is defined to be at the top of the screen. When bit(13) is set, the Y origin for all linear frame buffer accesses is defined to be at the bottom of the screen.

Bit(14) of **lfbMode** is used to select the W component used for LFB writes processed through the pixel pipeline. If bit(14)=0, then the MSBs of the fractional component of the 48-bit W value passed to the pixel pipeline for LFB writes through the pixel pipeline is the 16-bit Z value associated with the LFB write. [Note that the 16-bit Z value associated with the LFB write is dependent on the LFB format, and is either passed down pixel-by-pixel from the CPU, or is set to the constant **zaColor**(15:0)]. If bit(14)=1, then the MSBs of the fractional component of the 48-bit W value passed to the pixel pipeline for LFB writes is **zacolor**(15:0). Regardless of the setting of bit(14), when LFB writes go through the pixel pipeline, all other bits except the 16 MSBs of the fractional component of the W value are set to 0x0. Note that bit(14) is ignored if LFB writes bypass the pixel pipeline.

Linear Frame Buffer Writes

Linear frame buffer writes -- format 0:
When writing to the linear frame buffer with 16-bit format 0 (RGB 5-6-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha and depth information for LFB format 0 is taken from the

zaColor register. The following table shows the color channels for 16-bit linear frame buffer access format 0:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Red (15:11), Green(10:5), Blue(4:0)
1	15:0	Blue (15:11), Green(10:5), Red(4:0)
2	15:0	Red (15:11), Green(10:5), Blue(4:0)
3	15:0	Blue (15:11), Green(10:5), Red(4:0)

Linear frame buffer writes -- format 1:
When writing to the linear frame buffer with 16-bit format 1 (RGB 5-5-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha and depth information for LFB format 1 is taken from the **zaColor** register. The following table shows the color channels for 16-bit linear frame buffer access format 1:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Ignored(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Ignored(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Ignored(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Ignored(0)

Linear frame buffer writes -- format 2:
When writing to the linear frame buffer with 16-bit format 2 (ARGB 1-5-5-5), the RGB channel format specifies the RGB ordering within a 16-bit word. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then depth information for LFB format 2 is taken from the **zaColor** register. Note that the 1-bit alpha value passed when using LFB format 2 is bit-replicated to yield the 8-bit alpha used in the pixel pipeline. The following table shows the color channels for 16-bit linear frame buffer access format 2:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Alpha(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Alpha(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Alpha(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Alpha(0)

Linear frame buffer writes -- format 3:
Linear frame buffer format 3 is an unsupported format.

Linear frame buffer writes -- format 4:
When writing to the linear frame buffer with 24-bit format 4 (RGB x-8-8-8), the RGB channel format specifies the RGB ordering within a 24-bit word. Note that the alpha/A channel is ignored for 24-bit access format 4. Also note that while only 24-bits of data is transfered for format 4, all data access must be 32-bit aligned -- packed 24-bit writes are not supported by Banshee. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha and depth information for LFB format 4 is taken

from the **zaColor** register. The following table shows the color channels for 24-bit linear frame buffer access format 4:

RGB Channel Format Value	24-bit Linear frame buffer access bits (aligned to 32-bits)	RGB Channel
0	31:0	Ignored(31:24), Red (23:16), Green(15:8), Blue(7:0)
1	31:0	Ignored(31:24), Blue(23:16), Green(15:8), Red(7:0)
2	31:0	Red(31:24), Green(23:16), Blue(15:8), Ignored(7:0)
3	31:0	Blue(31:24), Green(23:16), Red(15:8), Ignored(7:0)

Linear frame buffer writes -- format 5:
When writing to the linear frame buffer with 32-bit format 5 (ARGB 8-8-8-8), the RGB channel format specifies the ARGB ordering within a 32-bit word. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then depth information for LFB format 5 is taken from the **zaColor** register. The following table shows the color channels for 32-bit linear frame buffer access format 5.

RGB Channel Format Value	24-bit Linear frame buffer access bits (aligned to 32-bits)	RGB Channel
0	31:0	Alpha(31:24), Red (23:16), Green(15:8), Blue(7:0)
1	31:0	Alpha(31:24), Blue(23:16), Green(15:8), Red(7:0)
2	31:0	Red(31:24), Green(23:16), Blue(15:8), Alpha(7:0)
3	31:0	Blue(31:24), Green(23:16), Red(15:8), Alpha(7:0)

Linear frame buffer writes -- formats 6-11:
Linear frame buffer formats 6-11 are unsupported formats.

Linear frame buffer writes -- format 12:
When writing to the linear frame buffer with 32-bit format 12 (Depth 16, RGB 5-6-5), the RGB channel format specifies the RGB ordering within the 32-bit word. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha information for LFB format 12 is taken from the **zaColor** register. Note that the format of the depth value passed when using LFB format 12 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). The following table shows the 16-bit color channels within the 32-bit linear frame buffer access format 12:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Red (15:11), Green(10:5), Blue(4:0)
1	15:0	Blue (15:11), Green(10:5), Red(4:0)
2	15:0	Red (15:11), Green(10:5), Blue(4:0)
3	15:0	Blue (15:11), Green(10:5), Red(4:0)

Linear frame buffer writes -- format 13:
When writing to the linear frame buffer with 32-bit format 13 (Depth 16, RGB x-5-5-5), the RGB channel format specifies the RGB ordering within the 32-bit word. If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then alpha information for LFB format 13 is taken from the **zaColor**

register. Note that the format of the depth value passed when using LFB format 13 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). The following table shows the 16-bit color channels within the 32-bit linear frame buffer access format 13:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Ignored(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Ignored(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Ignored(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Ignored(0)

Linear frame buffer writes -- format 14:

When writing to the linear frame buffer with 32-bit format 14 (Depth 16, ARGB 1-5-5-5), the RGB channel format specifies the RGB ordering within the 32-bit word. Note that the format of the depth value passed when using LFB format 14 must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). Also note that the 1-bit alpha value passed when using LFB format 14 is bit-replicated to yield the 8-bit alpha used in the pixel pipeline. The following table shows the 16-bit color channels within the 32-bit linear frame buffer access format 14:

RGB Channel Format Value	16-bit Linear frame buffer access bits	RGB Channel
0	15:0	Alpha(15), Red (14:10), Green(9:5), Blue(4:0)
1	15:0	Alpha(15), Blue (14:10), Green(9:5), Red(4:0)
2	15:0	Red (15:11), Green(10:6), Blue(5:1), Alpha(0)
3	15:0	Blue (15:11), Green(10:6), Red(5:1), Alpha(0)

Linear frame buffer writes -- format 15:

When writing to the linear frame buffer with 32-bit format 15 (Depth 16, Depth 16), the format of the depth values passed must precisely match the format of the type of depth buffering being used (either 16-bit integer Z or 16-bit floating point 1/W). If the Banshee pixel pipeline is enabled for LFB accesses (**lfbMode** bit(8)=1), then RGB color information is taken from the **color1** register, and alpha information for LFB format 15 is taken from the **zaColor** register.

userIntrCMD Register

Writing to the **userIntrCMD** register executes the USERINTERRUPT command:

Bit	Description
0	Wait for USERINTERRUPT to be cleared before continuing (1=stall graphics engine until interrupt is cleared)
1	Wait for interrupt generated by USERINTERRUPT (visible in **intrCtrl** bit(11)) to be cleared before continuing (1=stall graphics engine until interrupt is cleared)
9:2	User interrupt Tag

If the data written to **userIntrCMD** bit(0)=0, then a user interrupt is generated (**intrCtrl** bit(11) is set to 1). If the data written to **userIntrCMD** bit(1)=1, then the graphics engine stalls and waits for the USERINTERRUPT interrupt to be cleared before continuing processing additional commands. If no

USERINTERRUPT interrupt is set and the data written to **userIntrCMD** bit(1)=1, then the graphics engine will not stall and will continue to process additional commands. Software may also use combinations of intrCtrl bits(1:0) to generate different functionality.

The tag associated with a user interrupt is written to **userIntrCMD** bits 9:2. When a user interrupt is generated, the respective tag associated with the user interrupt is read from **IntrCtrl** bits 19:12.

If the USERINTERRUPT command does not stall the graphics engine (**userIntrCMD**(0)=1), then a potential race condition occurs between multiple USERINTERRUPT commands and software user interrupt processing. In particular, multiple USERINTERRUPT commands may be generated before software is able to process the first interrupt. Irrespective of how many user interrupts have been generated, the user interrupt tag field in **intrCtrl** (bits 19:12) always reflects the tag of last USERINTERRUPT command processed. As a result of this behavior, early tags from multple USERINTERRUPT commands may be lost. To avoid this behavior, software may force a single USERINTERRUPT command to be executed at a time by writing **userIntrCMD**(1:0)=0x3 and cause the graphics engine to stall until the USERINTERRUPT interrupt is cleared.

Note that bit 5 of **intrCtrl** must be set to 1 for user interrupts to be generated – writes to **userIntrCMD** when **intrCtrl**(5)=0 do not generate interrupts or cause the processing of commands to wait on clearing of the USERINTERRUPT command (regardless of the data written to **userIntrCMD**), and are thus in effect "dropped."

Command Descriptions

NOP Command

The NOP command is used to flush the graphics pipeline. When a NOP command is executed, all pending commands and writes to the texture and frame buffers are flushed and completed, and the graphics engine returns to its IDLE state. While this command is used primarily for debugging and verification purposes, it is also used to clear the 3D status registers (**fbiTriangles, fbiPixelsIn**, **fbiPixelsOut**, **fbiChromaFail**, **fbiZfuncFail**, and **fbiAfuncFail**). Setting **nopCMD** bit(0)=1 clears the 3D status registers and flushes the graphics pipeline, while setting **nopCMD** bit(0)=0 has no affect on the 3D status registers but flushes the graphics pipeline. See the description of the **nopCMD** register in section 5 for more information.

USERINTERRUPT Command

The USERINTERRUPT command allows for software-generated interrupts. A USERINTERRUPT command is generated by writing to the **userIntrCMD** register. **userIntrCMD** bit(0) controls whether a write to **userIntrCMD** generates a USERINTERRUPT. Setting **userIntrCMD** bit(0)=1 generates a USERINTERRUPT. **userIntrCMD** bit(1) determines whether the graphics engine stalls on software clearing of the user interrupt. By setting **userIntrCMD** bit(1)=1, the graphics engine stalls until the USERINTERRUPT is cleared. Alternatively, setting **userIntrCMD** bit(1)=0 does not stall the graphics engine upon execution of the USERINTERRUPT command, and additional graphics commands are processed without waiting for clearing of the user interrupt. A identification, or Tag, is also associated

with an individual USERINTERRUPT command, and is specified by writing an 8-bit value to **userIntrCMD** bits(9:2).

User interrupts must be enabled before writes to the **userIntrCMD** are allowed by setting **intrCtrl** bit(5)=1. Writes to **userIntrCMD** when **intrCtrl** bit(5)=0 are "dropped" and do not affect functionality. A user interrupt is detected by reading **intrCtrl** bit (11), and is cleared by setting **intrCtrl** bit(11)=0. The tag of a generated user interrupt is read from **intrCtrl** bits (19:12). See the description of the **intrCtrl** and **userIntrCMD** registers in section 5 for more information.

Linear Frame Buffer Access

The Banshee linear frame buffer base address is located at a 8 Mbyte offset from the **memBaseAddr** PCI configuration register and occupies 4 Mbytes of Banshee address. Regardless of actual frame buffer resolution, all linear frame buffer accesses assume a 2048-pixel logical scan line width. The number of bytes per scan line depends on the format of linear frame buffer access format selected in the **lfbMode** register. Note for all accesses to the linear frame buffer, the status of bit(16) of **fbzMode** is used to determine the Y origin of data accesses. When bit(16)=0, offset 0x0 into the linear frame buffer address space is assumed to point to the upper-left corner of the screen. When bit(16)=1, offset 0x0 into the linear frame buffer address space is assumed to point to the bottom-left corner of the screen. Regardless of the status of **fbzMode** bit(16), linear frame buffer addresses increment as accesses are performed going from left-to-right across the screen. Also note that clipping is not automatically performed on linear frame buffer writes if scissor clipping is not explicitly enabled (**fbzMode** bit(0)=1). Linear frame buffer writes to areas outside of the monitor resolution when clipping is disabled result in undefined behavior.

Linear frame buffer Writes

The following table shows the supported linear frame buffer write formats as specified in bits(3:0) of **lfbMode**:

Value	Linear Frame Buffer Access Format
	16-bit formats
0	16-bit RGB (5-6-5)
1	16-bit RGB (x-5-5-5)
2	16-bit ARGB (1-5-5-5)
3	Reserved
	32-bit formats
4	24-bit RGB (8-8-8)
5	32-bit ARGB (8-8-8-8)
7:6	Reserved
11:8	Reserved
12	16-bit depth, 16-bit RGB (5-6-5)
13	16-bit depth, 16-bit RGB (x-5-5-5)
14	16-bit depth, 16-bit ARGB (1-5-5-5)
15	16-bit depth, 16-bit depth

When writing to the linear frame buffer with a 16-bit access format (formats 0-3 and format 15 in **lfbMode**), each pixel written is 16-bits, so there are 2048 bytes per logical scan line. Remember when utilizing 16-bit access formats, two 16-bit values can be packed in a single 32-bit linear frame buffer write

-- the location of each 16-bit component in screen space is defined by bit(11) of **lfbMode**. When using 16-bit linear frame buffer write formats 0-3, the depth components associated with each pixel is taken from the **zaColor** register. When using 16-bit format 3, the alpha component associated with each pixel is taken from the 16-bit data transfered, but when using 16-bit formats 0-2 the alpha component associated with each pixel is taken from the **zaColor** register. The format of the individual color channels within a 16-bit pixel is defined by the RGB channel format field in **lfbMode** bits(12:9). See the **lfbMode** description in section 5 for a detailed description of the rgb channel format field.

When writing to the linear frame buffer with 32-bit access formats 4 or 5, each pixel is 32-bits, so there are 4096 bytes per logical scan line. Note that when utilizing 32-bit access formats, only a single pixel may be written per 32-bit linear frame buffer write. Also note that linear frame buffer writes using format 4 (24-bit RGB (8-8-8)), while 24-bit pixels, must be aligned to a 32-bit (doubleword) boundary -- packed 24-bit linear frame buffer writes are not supported by Banshee. When using 32-bit linear frame buffer write formats 4-5, the depth components associated with each pixel is taken from the **zaColor** register. When using format 4, the alpha component associated with each pixel is taken from the **zaColor** register, but when using format 5 the alpha component associated with each pixel is taken from the 32-bit data transfered. The format of the individual color channels within a 24/32-bit pixel is defined by the rgb channel format field in **lfbMode** bits(12:9).

When writing to the linear frame buffer with a 32-bit access formats 12-14, each pixel is 32-bits, so there are 4096 bytes per logical scan line. Note that when utilizing 32-bit access formats, only a single pixel may be written per 32-bit linear frame buffer write. If depth or alpha information is not transfered with the pixel, then the depth/alpha information is taken from the **zaColor** register. The format of the individual color channels within a 24/32-bit pixel is defined by the rgb channel format field in **lfbMode** bits(12:9). The location of each 16-bit component of formats 12-15 in screen space is defined by bit(11) of **lfbMode**. See the **lfbMode** description in section 5 for more information about linear frame buffer writes.

Linear frame buffer Reads

It is important to note that reads from the linear frame buffer bypass the PCI host FIFO (as well as the memory FIFO if enabled) but are blocking. If the host FIFO has numerous commands queued, then the read can potentially take a very long time before data is returned, as data is not read from the frame buffer until the PCI host FIFO is empty and the graphics pixel pipeline has been flushed. One way to minimize linear frame buffer read latency is to guarantee that the Banshee graphics engine is idle and the host FIFOs are empty (in the **status** register) before attempting to read from the linear frame buffer.

Programming Caveats

The following is a list of programming guidelines which are detailed elsewhere but may have been overlooked or misunderstood:

Memory Accesses

All Memory accesses to Banshee registers must be 32-bit word accesses only. Linear frame buffer accesses may be 32-bit or 16-bit accesses, depending upon the linear frame buffer access format specified in **lfbMode**. Byte(8-bit) accesses are only allowed to Banshee linear frame buffer.

Determining Banshee Idle Condition

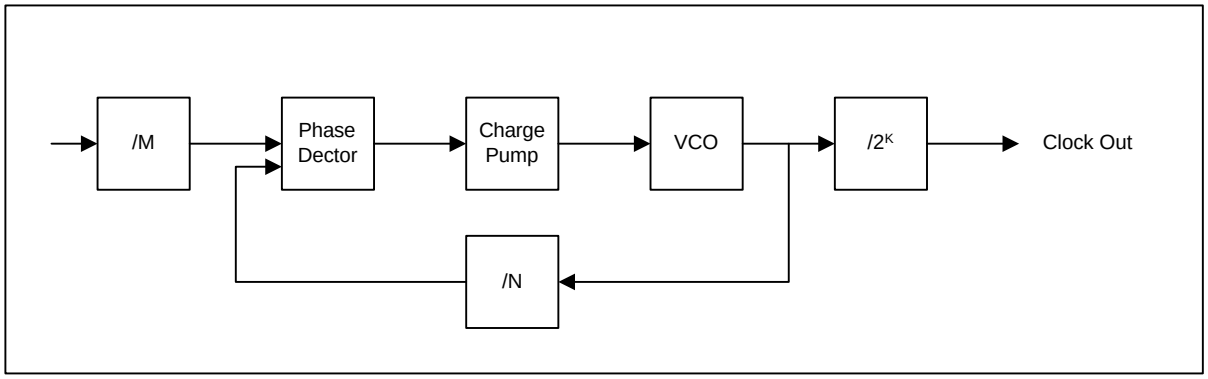
After certain Banshee operations, and specifically after linear frame buffer acceses, there exists a potential deadlock condition between internal Banshee state machines which is manifest when determining if the

Banshee subsystem is idle. To avoid this problem, always issue a NOP command before reading the **status** register when polling on the Banshee busy bit. Also, to avoid asynchronous boundary conditions when determing the idle status, always read Banshee inactive in **status** three times. A sample code segment for determining Banshee idle status is as follows:

```
/**********************************************************
 * SST_IDLE:
 *    returns 0 if SST is not idle
 *    returns 1 if SST is idle
 **********************************************************/
SST_IDLE()
{
  ulong j, i;

  // Make sure SST state machines are idle
  PCI_MEM_WR(NOPCMD, 0x0);
  i = 0;
  while(1) {
    j = PCI_MEM_RD(STATUS);
    if(j & SST_BUSY)
      return(0);
    else
      i++;
    if(i > 3)
      return(1);
  }
}
```

7. PLL Registers



Register Name	Address	Bits	R/W	Description
pllCtrl0	0x40-0x43	31:0	R/W	Video Clock PLL
pllCtrl1	0x44-0x47	31:0	R/W	GRX Clock PLL
pllCtrl2	0x48-0x4b	31:0	R/W	Mem Clock PLL

Genlock mode: in order for the register 3da (vga register) to reflect the status of vsync correct, **vgaInit0**[1]

needs to be set

PllCtrl registers

Bit	Description
1:0	K, Post divider value
7:2	M, PLL input divider
15:8	N, PLL multiplier
16	Test. (0=normal operation, 1 = CLK is output of VCO). Default = 0.

Frequency output of PLL's is given:

fout = 14.31818 * (N + 2) / (M + 2) / (2 ^ K).

8. DAC Registers

Register Name	I/O Address	Bits	R/W	Description
dacMode	0x4c-0x4f	4:0	R/W	Dac Mode 2:1 or 1:1
dacAddr	0x50-0x53	8:0	R/W	Dac pallette address
dacData	0x54-0x57	23:0	R/W	Dac data register
reserved	0x58-0x5b	na		

dacMode

Bit	Description
0	Dac Mode 2:1 or 1:1
1	Enable DPMS on Vsync
2	Force Vsync value.
3	Enable DPMS on Hsync
4	Force Hsync value.
5	Control crc2 collection mode (see crc2 register)

dacAddr

Bit	Description
8:0	Pallette Address

This is the 9 bit CLUT address used for programming the CLUT. Unlike the VGA mechanism, this address does not auto increment, but has access to the entire 512 entries in the CLUT.

dacData

Bit	Description
23:0	Dac color value

This is the 24 bit RGB value at the index programmed into **dacAddr**. The color values are always stored with red in bits [23:16], green in bits [15:8] and blue in bits [7:0].

9. Video Registers(PCI)

Register Name	I/O Addr	Bits	R/W	Description
vidMaxRgbDelta	0x58	23:0	R/W	Maximum delta values for video filtering
vidProcCfg	0x5c	31:0	R/W	Video Processor configuration
hwCurPatAddr	0x60	23:0	R/W	Cursor Pattern Address
hwCurLoc	0x64	26:0	R/W	X and Y location of HW cursor
hwCurC0	0x68	23:0	R/W	Hw cursor color 0
hwCurC1	0x6c	23:0	R/W	Hw cursor color 1
vidInFormat	0x70	31:0	R/W	Video In Format
vidInStatus	0x74	2:0	R	Video In Status register
vidSerialParallelPort	0x78	31:0	R/W	Serial and Parallel Ports
vidInXDecimDeltas	0x7c	31:0	R/W	Video In horizontal decimation delta 1 & 2.
vidInDecimInitErrs	0x80	31:0	R/W	Video In horizontal and vertical decimation initial error term
vidInYDecimDeltas	0x84	31:0	R/W	Video In vertical decimation delta 1 & 2
vidPixelBufThold	0x88	17:0	R/W	Video Pixel Buffer Threshold
vidChromaMin	0x8c	31:0	R/W	Chroma Key minimum value
vidChromaMax	0x90	31:0	R/W	Chroma Key maximum value
vidCurrentLine	0x94	10:0	R	Current Scan line
vidScreenSize	0x98	23:0	R/W	Screen resolution
vidOverlayStartCoords	0x9c	31:0	R/W	Start Surface Coordinates [31:28] Overlay Start Screen Coordinates
vidOverlayEndScreenCoord	0xa0	23:0	R/W	Overlay End Screen Coordinates
vidOverlayDudx	0xa4	19:0	R/W	Overlay horizontal magnification factor
vidOverlayDudxOffsetSrcWidth	0xa8	31:0	R/W	Overlay horizontal magnification factor initial offset (bit 18:0) Overlay source surface width (bit 31:19)
vidOverlayDvdy	0xac	19:0	R/W	Overlay vertical magnification factor
vidOverlayDvdyOffset	0xe0	18:0	R/W	Overlay vertical magnification factor initial offset
vidDesktopStartAddr	0xe4	23:0	R/W	Desktop start address
vidDesktopOverlayStride	0xe8	31:0	R/W	Desktop and Overlay strides
vidInAddr0	0xec	23:0	R/W	Video In Buffer 0 starting address
vidInAddr1	0xf0	23:0	R/W	Video In Buffer 1 starting address
vidInAddr2	0xf4	23:0	R/W	Video In Buffer 2 starting address
vidInStride	0xf8	14:0	R/W	Video In Buffer stride
vidCurrOverlayStartAddr	0xfc	23:0	R	Current overlay start address in use

vidMaxRgbDelta

The vidMaxRgbDelta register specifies the maximum delta values allowed for a pixel's color components to be filtered in the video filter (4x1 tap filter or 2x2 box filter). Each of the three neighbor pixels is compared with the center pixel, and if any of the RGB or YCbCr components exceed that of the center pixel by +delta or -delta, that color component will be replaced by that of the center pixel in the filter. The purpose of this is to prevent the high frequency pixels from being filtered in the tap or box filter. Putting 0x01 in each of the delta values minimizes the amount of filtering while 0x3f maximizes it. The value 0x0 is undefined.

In order to avoid stepping outside the source surface (3d or video surface), the tap and box filters uses the value programmed in vidOverlayEndScreenCoord to determine when it is getting to the right and bottom edges of the overlay window, and performs point sampling on the edges of the source surface. For a full screen 3d surface to be displayed with the box filter on a 640 x 480 resolution, vidOverlayEndScreenCoord need to be programmed with x=639 and y=479 for clamping to be performed properly.

Bit	Description
5:0	Maximum blue/V/Cr delta for video filtering (unsigned). Range from 0x1 to 0x3f. 0x0 is undefined.
13:8	Maximum green/U/Cb delta for video filtering (unsigned). Range from 0x1 to 0x3f. 0x0 is undefined.
21:16	Maximum red/Y delta for video filtering (unsigned). Range from 0x1 to 0x3f. 0x0 is undefined.

vidProcCfg Register

The **vidProcCfg** register is the general configuration register for the Video Processor. It is written by the host upon reset only.

Bit	Description
0	1: Video Processor on, VGA mode off; 0: Video Processor off, VGA mode on.
1	1: X11 cursor mode; 0: Microsoft Windows cursor mode.
2	Overlay stereo enable. 0 = disabled, 1 = enabled.
3	Interlaced video out enable. For Banshee, since interlaced video out is not supported, this bit should remain 0 all the time.
4	Half mode. 0 = disabled. 1 = enabled where desktop stride is added every other lines.
5	ChromaKeyEnable. 0 = off. 1 = on.
6	ChromaKeyResultInversion: (0 = desktop transparent if desktop color matches or falls within the chroma-key color range; 1 = desktop transparent if desktop color does not match or fall within the chroma-key range)
7	Desktop surface enable. 0 = do not fetch the desktop surface, 1 = fetch desktop surface
8	Overlay surface enable. 0 = do not fetch the overlay surface, 1 = fetch overlay surface
9	Video-in data displayed as overlay enable. 0 = do not display the video-in buffer directly as overlay. 1= use the video-in buffer address as the overlay start address (auto-flipping).
10	Desktop clut bypass. 0 = do not bypass the clut in the RAMDAC, 1 = bypass the clut
11	Overlay clut bypass. 0 = do not bypass the clut in the RAMDAC, 1 = bypass the clut
12	Desktop clut select. 0 = use the lower 256 entries of the clut. 1 = use the upper 256 entries.
13	Overlay clut select. 0 = use the lower 256 entries of the clut. 1 = use the upper 256 entries.
14	Overlay horizontal scaling enable. 0=disabled. 1=enabled. Magnification factor determined by vidOverlayDudx.
15	Overlay vertical scaling enable. 0=disabled. 1=enabled. Magnification factor determined by vidOverlayDvdy.

17:16	Overlay filter mode 00: point sampling 01: 2x2 dither subtract followed by 2x2 box filter (for 3d only) 10: 4x4 dither subtract followed by 4x1 tap filter (for 3d only) 11: bilinear scaling
20:18	Desktop pixel format 000: 8bit palettized 001: RGB565 undithered 010: RGB24 packed 011: RGB32 100: Reserved 101: Reserved 110: Reserved 111: Reserved
23:21	Overlay pixel format 000: Reserved 001: RGB565 undithered 010: Reserved 011: Reserved 100: YUV411 101: YUYV422 110: UYVY422 111: RGB565 dithered
24	Reserved (must be 0)
25	Reserved (must be 0)
26	2X mode which refreshes two screen pixels per video clock. 0 = 1X mode, 1 = 2X mode.
27	HW cursor enable. 0 = disabled, 1 = enabled.
28	Reserved
29	Reserved.
30	Reserved.
31	Backend deinterlacing for video overlay. 0 = No deinterlacing in the backend pipe. 1 = Backend deinterlacing (Bob method). Bob method displays either the even or odd frame at a time, and interpolates two interlaced lines to get the missing field. It is not supported in 2X mode.

How to program for Backend deinterlacing (Bob method):

The only thing this option effects is that when the video processor displays the even field, it adds 0.5 to the initial vertical offset (initial dvdy offset) used by the backend bilinear scaler. Everything else is the same.

Since deinterlacing in the backend uses the bilinear scaler unit to interpolate between two interlaced lines, the host needs to enable bilinear filtering, overlay vertical scaling, overlay horizontal scaling, and set up the initial dvdy offset, dvdy, initial dudx offset and dudx correctly according to the desired magnification factor between the source video and displayed video. The suggested setting for the parameters for backend deinterlacing without horizontal magnification are: bilinear filter enable = 1, overlay vertical scaling enable = 1, overlay horizontal scaling enable = 0, initial dvdy offset = 0.25, dvdy = 0.5. Initial dudx offset and dudx are don’t cares.

Backend deinterlacing is not supported for 2x mode (2-pixel per video clk mode) since bilinear filtering is not available in 2x mode.

How does the hardware work in half-mode/ low-resolution mode?
The video refresh has an internal register which stores the memory address of where a scanline starts in the desktop surface. At vertical retrace, this internal register is loaded with the value of vidDesktopStartAddr. When half-mode is disabled, a stride is added to this internal register at the end of every scanline to move to the next scanline. However, when half-mode is enabled, the stride is added at the end of every other scanline, i.e., at the end of scanline 1, 3, 5, 7, …… etc. As a result, each line of the desktop surface will be displayed twice, and the height of the video display will be double the height of the desktop surface.

As one could see, the half-mode bit doubles the video display in the y-direction only. To double the number of pixels in the x-direction, one needs to half the video clock frequency. For example, to display a desktop surface of 320 x 240 on the monitor as 640 x 480 at 60 frames per second, a video clock of frequency 12.59MHz (25.175MHz / 2) is needed. Also, while the vertical VGA timing parameters are the same as those for 640 x 480, the horizontal parameters (e.g., Total number of horizontal pixels, width of Hsync, number of pixels in horizontal blank, …… etc) need to be halved. With half of the video clock frequency and the horizontal timing parameters, the monitor will see a timing which is equivalent to a width of 640 pixels. Lastly, the video register, vidScreenSize, need to be programmed with x = 320 and y = 480, and each pixel presented by the video refresh unit to the monitor will be displayed twice in the x-direction.

Line doubling is implemented for the desktop surface only, and is not available for overlay surface and hardware cursor. Therefore if the overlay and the hardware cursor are enabled in half-mode, they will be doubled in the x-dimension while they y-dimension will remain the same.

hwCurPatAddr Register

The **hwCurPatAddr** register stores the starting address of two monochrome cursor patterns. Each pattern is a bitmap of 64-bit wide and 64-bit high (a total of 8192 bits). The two patterns are stored in such a way that pattern 0 always resides in the lower half (least significant 64-bit) of a 128-bit word and pattern 1 the upper half. In other words, each 128-bit word consists of one line from pattern 0 and one line from pattern 1. At each horizontal retrace, the Video Processor checks to see whether the cursor location falls on the current scanline. If so, it fetches from the memory eight words of cursor patterns at a time. The eight words are then stored in the on-chip ram for use in the next eight scanlines. This reduces the number of memory accesses for cursor patterns from 64 to 8 times per screen refresh. Cursor patterns always reside in linear address space, and the linear stride is always 16 bytes. The video processor figures out the shape and color of the cursor for the current scanline according to the following table:

Bit from Pattern 0	Bit from Pattern 1	Displayed cursor (Microsoft window)	Displayed cursor (X11)
0	0	HWCurC0	Current Screen Color
0	1	HWCurC1	Current Screen Color
1	0	Current Screen color	HWCurC0
1	1	NOT current screen color	HWCurC1

Bit	Description
23:0	Physical address of where the cursor pattern resides in the memory.

hwCurLoc Register

The **hwCurLoc** register stores the x and y coordinates of the bottom right corner of the cursor. The coordinates are unsigned, and range from 0 to 2047. This allows a partial cursor to be displayed in all edges of the screen.

Bit	Description
10:0	X coordinate of the bottom right corner of the cursor. Undefined upon reset.
26:16	Y coordinate of the bottom right corner of the cursor. Undefined upon reset.

hwCurC0 Register

The **hwCurC0** register stores color 0 of the cursor.

Bit	Description
7:0	Blue value of cursor color0
15:8	Green value of cursor color0
23:16	Red value of cursor color0

hwCurC1 Register

The **hwCurC1** register stores color 1 of the cursor.

Bit	Description
7:0	Blue value of cursor color1
15:8	Green value of cursor color1
23:16	Red value of cursor color1

vidInFormat

The VidInFormat register allows the host to specify the data format of the video-in data.

Bit	Description
0	Reserved
3:1	(VMI only) Video-In data format 110: 8bit YCbCr 4:2:2 (UYVY) 101: 8bit YCbCr 4:2:2 (YUYV) 100: 8bit YCbCr 4:1:1
4	(VMI only) Video-In de-interlacing mode. (0 = No deinterlacing applied to the video data coming in; 1 = Weave method deinterlacing, i.e. the video-in port will merge two consecutive VMI frames into one inside the frame buffer before signaling a frame is done in the vidStatus register.)
5	(VMI/TV out master mode) Vmi_vsync_in polarity. (1=active low; 0=active high (default))
6	(VMI/TV out master mode) Vmi_hsync_in polarity. (1=active low; 0=active high (default))
7	(VMI only) Vmi_vactive_in polarity. (1=active low; 0=active high (default))

8	(TV out only) G4 for posedge (1=Brooktree TV out support; 0=Chrontel) 1: Brooktree TV encoder samples at falling edge for the following data; 0: Chrontel TV encoder samples at rising edge for the following data Data[11] G0[4] Data[10] G0[3] Data[9] G0[2] Data[8] B0[7] Data[7] B0[6] Data[6] B0[5] Data[5] B0[4] Data[4] B0[3] Data[3] G0[0] Data[2] B0[2] Data[1] B0[1] Data[0] B0[0] 1: Brooktree TV encoder samples at rising edge for the following data; 0: Chrontel TV encoder samples at falling edge for the following data Data[11] R0[7] Data[10] R0[6] Data[9] R0[5] Data[8] R0[4] Data[7] R0[3] Data[6] G0[7] Data[5] G0[6] Data[4] G0[5] Data[3] R0[2] Data[2] R0[1] Data[1] R0[0] Data[0] G0[1]
10:9	(VMI only) VideoIn buffering mode select (00=single buffer, 01=double buffer, 10= triple buffer)
11	Reserved (must be 0)
12	Reserved (Was VMI parallel interface enable)
13	Reserved (Was DPMS enable)
15:14	(VMI/ TV out) VideoIn interface configuration 00: VideoIn Interface not used 01: VMI (Also need to clear vidSerialParallelPort reg bit[29] to 0. This controls GPIO[1], which is used in the reference board to select the VMI device as the driver of some shared pins between VMI and TV out). 10: VideoIn Interface reconfigured to digital TV out (Also need to set vidSerialParallelPort reg bit[1] to 1. This controls GPIO[1], which is used in the reference board to de-select the VMI device as the driver of some shared pins between VMI and TV out). 11: Reserved

16	(VMI/TV out) TV Out Genlock enable. 0: The VMI logic of the video controller uses vmi_pixclk_in as its clock while the remaining video logic uses a separate video clock from the on-chip PLL. For TV out mode, vmi_vsync and vmi_hsync are output of the video controller. Vmi_vactive is always output of the video controller. (TV encoder is the slave device.) 1: Both the VMI logic and the remaining video logic use vmi_pixclk_in as their clock. For TV out mode, vmi_vsync and vmi_hsync are input of the video controller. Vmi_vactive is always output of the video controller.(TV encoder is the master device.) By setting bit 16 to 1, it allows Banshee to genlock to the clock of an external VMI device or TV encoder.
17	(VMI/TV out) not_use_vga_timing_signal (Timing signals include vert_exra, display_ena, vfrontporch_active, vbackporch_active, vblank, vga_blank_n, vga_vsync, vga_hsync) 0: Use the timing signals supplied by the VGA. For VMI and TV out slave mode. 1: Do not use the timing signals from the VGA. Timing signals are either supplied by the TV encoder (in its master mode) or generated internally by the video controller. For TV out master mode only.
18	Reserved. Used to be Field detection timing. (1=leading edge of Vsync; 0=trailing edge of Vsync (default))
19	Reserved. Used to be Video-In blank time data capture enable. 0 = disabled, 1 = enabled.
20	Video-In horizontal decimation enable. (0=off; 1=on)
21	Video-In vertical decimation enable. (0=off; 1=on)
31:22	Reserved

VMI field detection

Note that the polarity of the VMI Vsync, Hsync, and Vactive signals is programmable. The inactive going edge of the Vsync signal indicated whether the field is odd or even. If Hsync is active during the inactive going edge of Vsync, the field is even. If Hsync is inactive, the field is odd.

vidInStatus

The VidInStatus register allows the host to read the status of the video-in port, and implement manual buffer flipping for the video-in data.

Bit	Description
2	Even/odd field of the frame VMI just finishes drawing. 1=even; 0=odd.
1:0	Video-in buffer VMI just finishes writing to. 00=buffer 0 (as specified by vidInAddr0); 01=buffer 1 (as specified by vidInAddr1); 10=buffer 2 (as specified by vidInAddr2); 11=No buffer is ready yet, video processor is still working on the first frame

vidSerialParallelPort Register

The vidSerialParallelPort register controls the chip's I2C, DDC, GPIO, and the host port interface. Bit[17:0] of the register are shared between the I2C and GPIO interface. If VideoIn interface is configured to VMI (vidInFormat bit[15:14] == 2'b01), vidSerialParallelPort[17:0] are for VMI's host port interface. If configured to TV out (vidInFormat bit[15:14] == 2'b10), the bits are used to control digital TV out's

additional GPIO interface. This is in addition to the two GPIO pins (one input and one output), which are controlled by vidSerialParallelPort[30:29].

Since VMI, TV out, and ROM share pins for their interface, a pin can be input or output depending on which interface has control of the pin at that time. GPIO[0] (different from TV_out_GPIO[0]) is a hardwired output pin designed to be an output enable of the on-board tristate drivers. GPIO[0] is asserted low, when the VMI device has control of the shared pins, and is driving pixdata[7:0], vmi_rdy_n, and vmi_intreq_n as input to Banshee. GPIO[0] is pulled high, when either ROM or TV out controls the shared pins, and pixdata[7:0], vmi_rdy_n, and vmi_intreq_n are output of Banshee.

GPIO[1] is software programmable, and is used to control the output enable of the on-board tristate drivers for vmi_pixclk, vmi_vsync, vmi_hsync, and vmi_vactive. These are the signals that should be continually driven by the external vmi device even when the ROM is using the shared pins (ROM does not use the vmi_pixclk, vmi_vsync, vmi_hsync, and vmi_vactive pins). Otherwise the internal state of the vmi controller in Banshee may be messed up. Vmi_cs_n cannot be used in lieu of GPIO[1] for this purpose because the chip select pin can be turned off by vmi parallel host interface enable bit (bit 0 below).

Bit	Written By	Description (VMI is enabled)		(digital TV out interface is enabled)
		VMI host port interface		**General purpose IO interface**
0	host	VMI parallel host interface enable. (0=off, 1=on); Default to 0 upon reset.		Not used
1	host	VMI CS_N (Chip Select)		Not used
		Mode A	Mode B	
2	host/ encoder	VMI DS_N (Data Strobe)	VMI RD_N (Read)	TV out GPIO[0] (Output ONLY!)
3	host	VMI RW_N (Read/ Write_n)	VMI WR_N (Write)	TV out GPIO[0] output enable_n
4	VMI	VMI data DTACK_N (Data Acknowledge)	VMI data RDY (Data Ready)	Not used
5	host	VMI Data output enable_n. (0 = enabled, 1 = disabled); Default to 1 upon reset.		Not used
13:6	host/VMI	VMI Data (Input / Output)		Not used
17:14	host/ encoder (for bits 14 and 16 only)	VMI Address		Bit 14: TV out GPIO[2] (Output ONLY!) Bit 15: TV out GPIO[2] output enable_n Bit 16: TV out GPIO[1] (Input/Output) Bit 17: TV out GPIO[1] output enable_n
		DDC interface		**DDC interface**
18	host	DDC port enable (0 = disabled, 1 = enabled) Default to 0 upon reset.		DDC port enable (0 = disabled, 1 = enabled) Default to 0 upon reset.
19	host	DDC DCK write (0 = DCK pin is driven low, 1= DCK pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 26. Default to 1 upon reset.		DDC DCK write (0 = DCK pin is driven low, 1= DCK pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 26. Default to 1 upon reset.

20	host	DDC DDA write (0 = DDA pin is driven low, 1= DDA pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 27. Default to 1 upon reset.	DDC DDA write (0 = DDA pin is driven low, 1= DDA pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 27. Default to 1 upon reset.
21	Monitor	DDC DCK state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)	DDC DCK state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)
22	Monitor	DDC DDA state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)	DDC DDA state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)
		I2C interface	**I2C interface**
23	host	I2C port enable (0 = disabled, 1 = enabled) Default to 0 upon reset.	I2C port enable (0 = disabled, 1 = enabled) Default to 0 upon reset.
24	host	I2C SCK write (0 = SCK pin is driven low, 1= SCK pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 21. Default to 1 upon reset.	I2C SCK write (0 = SCK pin is driven low, 1= SCK pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 21. Default to 1 upon reset.
25	host	I2C SDA write (0 = SDA pin is driven low, 1= SDA pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 22. Default to 1 upon reset.	I2C SDA write (0 = SDA pin is driven low, 1= SDA pin is tri-stated) When this pin is tri-stated, other devices can drive this line, and the final state of the pin is reflected in bit 22. Default to 1 upon reset.
26	VMI/ encoder	I2C SCK state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)	I2C SCK state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)
27	VMI/ encoder	I2C SDA state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)	I2C SDA state (read only, 0 = low, 1 = tri-stated which means no device is driving this pin)
		Misc.	**Misc.**
28	host	VMI reset_n (1 = normal. 0 = reset VMI device.) Default to 0 upon reset.	Reset_n
29	host	output only gpio GPIO[1] output	output only gpio GPIO[1] output
30	VMI/ encoder	input only gpio GPIO[2] input	input only gpio GPIO[2] input
31		Not used	Not used

H3 Pinout sharing VMI, ROM and TV Out **3/3/98**

Notes:

1. **Rom access, VMI Video data/host port access and TV Out can only be performed separately**
2. **The only exception to 1, is "bypass" mode, where the VMI device can stream data and timing info directly into the TV encoder, while H3 snoops the data to display on the RGB monitor, either windowed or full screen.**
3. **This solution allows simulaneous RGB and TV out of the Windows desktop, simultaneous VMI and TV out for full screen VMI on the TV and windowed VMI on the RGB, but not windowed VMI on the TV.**
4. **The type in the table below is reference to H3**
5. **The programmability of the VMI or TV Encoder can be done via I2C, e.g. PAL mode,**
6. **The TV encoder must be able to operate in Master mode where it supplies the clock, vsync,hsync,blank and H3 outputs clock_out(delayed version of clock_in) and synchronous data**
7. **We must route a reference board to make sure the pin functions have been shared to provide a decent route**
8. **The ROM cs_n is tied to GND, the oe_n and we_n are used to control read/write respectively**

Pin Name	Pin Number	Pin Function Rom Access	Type	Pin Function VMI	Type	Pin Function Digital TV out rising/falling edge	
pixdata/a0		A0	out	Y0/Cr0/Cb0*	in	B0/G1	out
pixdata/a1		A1	out	Y1/Cr1/Cb1*	in	B1/R0	out
pixdata/a2		A2	out	Y2/Cr2/Cb2*	in	B2/R1	out
pixdata/a3		A3	out	Y3/Cr3/Cb3*	in	G0/R2	out
pixdata/a4		A4	out	Y4/Cr4/Cb4*	in	B3/G5	out
pixdata/a5		A5	out	Y5/Cr5/Cb5*	in	B4/G6	out
pixdata/a6		A6	out	Y6/Cr6/Cb6*	in	B5/G7	out
pixdata/a7		A7	out	Y7/Cr7/Cb7*	in	B6/R3	out
vmi_adr/a8		A8	out	vaddr0	out	B7/R4	out
vmi_adr/a9		A9	out	vaddr1	out	G2/R5	out
vmi_adr/a10		A10	out	vaddr2	out	G3/R6	out
vmi_adr/a11		A11	out	vaddr3	out	G4/R7	out
vmi_cs_n		***CANNOT USE!***		vmi_cs_n	out	***CANNOT USE!***	
vmi_rw		A14	out	vmi_rw_n/vmi_wr_n	out	clock_out	out
vmi_ds_n/a15		A15	out	vmi_ds_n/vmi_rd_n	out	TV_OUT_GPIO_0	out
vmi_rdy/busy		A12	out	vmi_dtack_n/vmi_rdy_n*	in	TV_OUT_GPIO_1	in/out
vmi_hdata		D0	in/out	vmi_hd_0	in/out	***NOT USED***	
vmi_hdata		D1	in/out	vmi_hd_1	in/out	***NOT USED***	
vmi_hdata		D2	in/out	vmi_hd_2	in/out	***NOT USED***	
vmi_hdata		D3	in/out	vmi_hd_3	in/out	***NOT USED***	
vmi_hdata		D4	in/out	vmi_hd_4	in/out	***NOT USED***	
vmi_hdata		D5	in/out	vmi_hd_5	in/out	***NOT USED***	
vmi_hdata		D6	in/out	vmi_hd_6	in/out	***NOT USED***	
vmi_hdata		D7	in/out	vmi_hd_7	in/out	***NOT USED***	
hsync		***NOT USED***		hsync	in	hsync	in/out
vsync		***NOT USED***		vsync_n	in	vsync	in/out
blank_n		***NOT USED***		blank_n	in	blank_n	in/out
pix_clk_in		***NOT USED***		vid_clk_in	in	clock_in	in
vmi_insert_n		Can this be detected by Software instead of wasting a pin? -- ***Pin Removed***					
vmi_intreq_n		A13	out	vmi_int_n*	in	TV_OUT_GPIO_2	out
reset_out		***NOT USED***		reset_n	out	reset_n	out
rom_oe_n		rom_oe_n	o	***CANNOT USE!***		***CANNOT USE!***	
rom_we_n		rom_we_n	o	***CANNOT USE!***		***CANNOT USE!***	
i2c_clk		***NOT USED***		i2c_clk	out	i2c_clk	out
i2c_data		***NOT USED***		i2c_data	in/out	i2c_data	in/out
gp_io[0]		vmi_oe_n	out	vmi_oe_n	out	vmi_oe_n	out
gp_io[1]		vmi_sync_oe_n	out	vmi_sync_oe_n	out	vmi_sync_oe_n	out
gp_io[2]		gp_in	in	gp_in	in	gp_in	in

* means the signal may be buffered from the VMI data bus to ensure that it is not driven during ROM accesses.

Issues:

1. Brooktree part does not support CCIR656 where a data is transferred on rising edges only with a 2X clock. The Brooktree part uses a 1X clock and pumps data on both edges.

vidInXDecimDeltas (for VMI downscaling Brensenham Engine)/ vidTvOutBlankHCount (for TV out master mode)

If VideoIn Interface is configured to VMI mode (i.e., VidInFormat[15:14] == 2'b01), vidInXDecimDeltas bits [11:0] contain the width of the destination video-in surface (width of the video overlay stored in the frame buffer) in number of pixels. VidInXDecimDeltas bits[27:16] contain the width difference between the source video-in surface (from VMI port) and destination video-in surface in number of pixels (Source - Destination)

Bit	Description
11:0	The positive (unsigned) value added to the error term when the horizontal Bresenham error term is <0. It is programmed to be the **width of the destination video-in surface in number of pixels.**
15:12	reserved
27:16	The positive (unsigned) value added to the error term when the horizontal Bresenham error term is >0. It is programmed to be the difference between the width of the source and destination video-in surfaces. **(Source - Destination) in number of pixels.**
31:28	reserved

If VideoIn Interface is configured to digitial TV out (i.e., VidInFormat[15:14] == 2'b10), TV Out Genlock is enabled (VidInFormat[16] == 1'b1), and Not_use_vga_timing_signal is asserted, vidInXDecimDeltas bits[10:0] contains the number of clock cycles after the leading edge of vmi_hsync before the horizontal active region starts (i.e., horizontal blank becomes deasserted).

VidInXDecimDelata bits[26:16] contains the number of clock cycles after leading edge of vmi_hsync before the horizontal active region ends (i.e., horizontal blank is re-asserted).

Output blank_n == horizontal blank_n AND vertical blank_n.

Note that the value in bits[26:16] needs to be greater than bits[10:0]. The clock cycles are based on the clock coming in through the vmi_pixclk pin.

Bit	Description
10:0	The number of clock cycles after the leading edge of vmi_hsync before the horizontal active region starts (i.e., horizontal blank becomes deasserted).
15:11	reserved
26:16	The number of clock cycles after leading edge of vmi_hsync before the horizontal active region ends (i.e., horizontal blank is re-asserted).
31:27	reserved

vidInDecimInitErrs

	Description
	The signed (2's complement) initial value of the error term in the horizontal Bresenham accumulator
	reserved
	The signed (2's complement) initial value of the error term in the vertical Bresenham accumulator
	reserved

vidInYDecimDeltas

If VideoIn Interface is configured to VMI mode (i.e., VidInFormat[15:14] == 2'b01), vidInYDecimDelta bits[11:0] contain the height of the destination video input window (height of the video overlay stored in the frame buffer) in number of lines. vidInYDecimDeltas contains the height difference between the source video surface (from VMI port) and destination video input window in number of lines (Source - Destination).

Bit	Description
11:0	The positive value added to the error term when the vertical Bresenham error term is <0
15:12	reserved
27:16	The positive value added to the error term when the vertical Bresenham error term is >0
31:28	reserved

If VideoIn Interface is configured to digitial TV out (i.e., VidInFormat[15:14] == 2'b10), and TV Out Genlock is enabled (VidInFormat[16] == 1'b1), vertical blank_n signal is de-asserted when the number of positive edges of vmi_hsync after the positive edge of vmi_vsync == vidInYDecimDeltas bits[10:0].

Vertical blank_n signal is re-asserted when the number of positive edges of vmi_hsync after the positive edge of vmi_vsync == vidInYDecimDelta bits[26:16].

Output blank_n == horizontal blank_n AND vertical blank_n.

Note that the value in bits[26:16] needs to be greater than bits[10:0]. The clock cycles are based on the clock coming in through the vmi_pixclk pin.

Bit	Description
10:0	The number of vmi_hsync LEADING edges after the LEADING edge of vmi_vsync before the vertical active region starts (i.e., vertical blank becomes deasserted).
15:11	reserved
26:16	The number of vmi_hsync LEADING edges after the LEADING edge of vmi_vsync before the vertical active region ends (i.e., vertical blank is re-asserted).
31:27	reserved

Bresenham scaler for scaling down a video window in the horizontal direction:

error = vidInXDecimInitErr;

repeat until the source pixels of a video window scanline are exhausted

```
if (error < 0)
        move to next source pixel
        error = error + vidInXDecimDelta1
else
        select the current source pixel as the destination pixel
        move to next source pixel
        error = error - vidInXDecimDelta2
```

Bresenham scaler for scaling down a video window in the vertical direction:

error = vidInYDecimInitErr

at each VideoIn Hsync

```
if (error < 0)
        skip the whole line of video in data
        error = error + vidInYDecimDelta1
else
        select the current line of video in data
        error = error - vidInYDecimDelta2
```

vidPixelBufThold

The vidPixelBufThold determines how many empty slots in each of the three pixel buffers will trigger refilling of the buffers.

Bit	Description
5:0	Primary pixel buffer low watermark (0x0 – 1 empty slot; 0x3f – 64 empty slots)
11:6	Secondary pixel buffer 0 low watermark (0x0 – 1 empty slot; 0x3f – 64 empty slots)
17:12	Secondary pixel buffer 1 low watermark (0x0 – 1 empty slot; 0x3f – 64 empty slots)

vidChromaKeyMin Register

The vidChromaKeyMin register contains the lower bound of the chroma key color.

Bit	Description
	8-bit desktop color format
7:0	chroma key color
31:8	Reserved
	15-bit desktop color format
4:0	Blue value of the chroma -key
9:5	Green value of the chroma -key
14:10	Red value of the chroma -key
31:15	Reserved
	16-bit desktop color format
4:0	Blue value of the chroma -key
10:5	Green value of the chroma -key
15:11	Red value of the chroma -key
31:16	Reserved
	24-bit desktop color format
7:0	Blue value of the chroma -key
15:8	Green value of the chroma -key
23:16	Red value of the chroma -key
31:24	Reserved
	32-bit desktop color format

7:0	Blue value of the chroma -key
15:8	Green value of the chroma -key
23:16	Red value of the chroma -key
31:24	Reserved

vidChromaKeyMax Register

The vidChromaKeyMax register contains the upper bound of the chroma key color. It is the same as vidChromaKeyMin if the chroma-key is a single color instead of a range.

Bit	Description
31:0	Format same as vidChromaKeyMin Register

vidCurrentLine Register

The vidCurrentLine register contains the current scan out line. As the vertical beam scans down the display this register is incremented.

Bit	Description
10:0	Current Video scan line.

vidScreenSize

NOTE: Whenever the screen resolution is changed, video processor needs to be re-enabled by clearing vidProcCfg bit 0 and setting it to 1. This will reset the video processor.

Bit	Description
11:0	Width of the screen in number of pixels. If vidScreenX is specified to be bigger than 1280, 2x mode needs to be enabled.
23:12	Height of the screen in number of lines.

vidOverlayStartCoords

Bit	Description
11:0	The x-coordinate on the screen where the upper left corner of the overlay locates.
23:12	The y-coordinate on the screen where the upper left corner of the overlay locates.
25:24	The lower two bits of the x-coordinate for the first pixel (at the upper left corner) of the overlay window with respect to the beginning of the source surface. Since the overlay window may be partially occluded by the dimension of the screen, the first pixel of the window may not necessarily be the first pixel of the source surface. The lower two bits of the x-coordinate are used for undithering.
27:26	The lower two bits of the y-coordinate for the first pixel (at the upper left corner) of the overlay window with respect to the beginning of the source surface. Since the overlay window may be partially occluded by the dimension of the screen, the first pixel of the window may not necessarily be the first pixel of the source surface. The lower two bits of the y-coordinate are used for undithering.
31:28	reserved

vidOverlayEndScreenCoord

Beware that for a full screen overlay window, for example in 640x480 resolution, the vidOverlayEndScreenCoord should be programmed to be 639x479 since the screen coordinate system starts at the upper left orgin as (0,0).

Bit	Description
11:0	The x-coordinate on the screen where the lower right corner of the overlay locates.
23:12	The y-coordinate on the screen where the lower right corner of the overlay locates.

vidOverlayDudx

When setting the vidOverlayDudx and vidOverlayDvdy, one needs to caution that the video refresh unit does not step outside the 3d or video surface when it gets to the rightmost and bottom edges of the overlay window. Since the video refresh unit does not know the dimension of the 3d or video surface, if vidOverlayDudx and vidOverlayDvdy are set too large, the refresh unit may step outside the source surface and cause artifacts at the edges of the overlay window.

Bit	Description
19:0	Step size in source per horizontal step in screen space for magnification. Format is 0.20.

vidOverlayDudxOffsetSrcWidth

Bit[31:19] specifies the number of bytes of pixels that need to be fetched from the frame buffer to cover a line of overlay window. The value depends on the width of the overlay window, the pixel depth of the overlay, and the x-scaling factor, and the vidOverlayDudxOffset. The vidOverlayDuDxOffset will affect the value only by +/- 1 pixel. The easiest way to figure out the value for the source width is to divide the number of pixels for the width of the overlay window by the x-scaling factor. Round the result up. Add 1 to adjust for any DudxOffset, and finally multiply the value by the overlay pixel depth. This is a conservative way to estimate for the source width since it will give a value slightly bigger than the actual number of bytes that are needed. Putting in a value which is smaller or grossly larger than the actual number of bytes needed will cause serious artifacts.

Bit	Description
18:0	Initial offset of Dudx. Format is 0.19.
31:19	Number of bytes needed to be fetched from the source surface in order to cover a whole un-occluded scanline for the overlay (13 bits allows a max of 8K bytes for an overlay scanline). i.e., ((Overlay width in number of screen pixels * vidOverlayDudx) + vidOverlayDudxOffset)) * overlay pixel depth in bytes. For non-scaled overlay with no offset, vidOverlayDudx becomes 1, and vidOverlayDudxOffset becomes 0 in the above equation.

vidOverlayDvdy

When setting the vidOverlayDudx and vidOverlayDvdy, one needs to caution that the video refresh unit does not step outside the 3d or video surface when it gets to the rightmost and bottom edges of the overlay window. Since the video refresh unit does not know the dimension of the 3d or video surface, if vidOverlayDudx and vidOverlayDvdy are set too large, the refresh unit may step outside the source surface and cause artifacts at the edges of the overlay window.

Bit	Description
19:0	Step size in source per vertical step in screen space for magnification. Format is 0.20.

vidOverlayDvdyOffset

Bit	Description
18:0	Initial offset of Dvdy. Format is 0.19.

Example:

Given source size of 640 x 240 and have it magnified to 1024 x 768 on the screen.

Source width:
Dudx[31:19] = 640 X 2 bytes = 1280 bytes (here 16 bpp assumed)
= 500h

Dudx[19:0] = 640/1024 = 0.625 = a0000h
(Note format is 0.20 means
x x x x x x x x x x x x x x x x x x x x
| | |
| | | .....
| | 0.125
| |
| 0.25
|
0.5)

Dvdy[19:0] = 240/768 = 0.3125 = 0.25 + 0.0625 = 50000h
(Format same as dudx above)

Dudx Offset[18:0] and Dvdy Offset [18:0] = 00000h if no initial offset is needed.
If upper leftmost overlay pixel needs to be the center of
the first pixel of the overlay surface, both offsets needs to be set to 0.5
which is 40000h.

vidDesktopStartAddr

Bit	Description
23:0	Physical starting address of the desktop surface. This is a **byte-aligned** address.

vidDesktopOverlayStride

Bit	Description
14:0	Bit[14:0] contains the linear stride of the surface in bytes. If interlaced video output mode is enabled, the linear stride is still programmed to 1x the regular stride of the surface, and will be multiplied by 2 when used.
15	reserved

For video overlay, the **stride** needs to be **a multiple of 4-bytes for YUV 422** pixel format and **a multiple of 8-bytes for YUV 411** pixel format. This ensures that the right edge of the video source surface to fall on a boundary of 2 pixels for YUV 422 and 4 pixels for YUV 411. The start address for the overlay is sampled from the FIFO'ed leftOverlayBuf and rightOverlayBuf registers. **The start address needs to be aligned on a 32-bit boundary for YUV 422 pixel format and a 64-bit boundary for YUV 411 pixel format.**

Bit	Description
30:16	Bit[30:16] contains the linear stride of the overlay surface in bytes. If interlaced video output mode is enabled, the linear stride is still programmed to 1x the regular stride of the surface, and will be multiplied by 2 when used.
31	reserved

vidInAddr0

Bit	Description
23:0	Starting address of video-in buffer 0

vidInAddr1

Bit	Description
23:0	Starting address of video-in buffer 1

vidInAddr2

Bit	Description
23:0	Starting address of video-in buffer 2

vidInStride

Bit	Description
14:0	This register contains the linear stride of the buffer in bytes. If interlaced video input mode is enabled, the linear stride is still programmed to 1x the regular stride, and will be multiplied by 2 before used.

vidCurrOverlayStartAddr

The vidCurrOverlayStartAddr register allows the host to read the start address which the video processor is using to refresh the overlay window for the current frame.

Bit	Description
23:0	Start physical address the video processor is using to refresh the overlay window. **Read only.**

Video-In Interface

Function

Video In Processor supports several connector interfaces for video data input. The following table shows the signals needed for each interface.

Signals

Ports	DDC(I2C)	VMI*	VMI (Mode A)	VMI (Mode B)
VMI Video Port		Hsync in	Hsync in	Hsync in
VMI Video Port		Vsync in	Vsync in	Vsync in
VMI Video Port		Vactive	Vactive	Vactive
VMI Video Port		P[7:0]	P[7:0]	P[7:0]
VMI Video Port		Pixclk (in)	Pixclk (in)	Pixclk in)
VMI I2C Port		SDA (in/out)		
VMI I2C Port		SCK (in/out)		
VMI Host Port			D[7:0] (in/out)	D[7:0]
VMI Host Port			A[3:0] (out)	A[3:0]
VMI Host Port			cs_n (out)	cs_n
VMI Host Port			ds_n (out)	rd_n
VMI Host Port			r/w_n (out)	wr_n
VMI Host Port			dtack_n (in)	ready
DDC Port	SDA (in/out)			
DDC Port	SCK (in/out)			
System signals		vmi_reset_n (out)	vmi_reset_n (out)	vmi_reset_n (out)
System signals		vmi_int_n (in)	vmi_int_n	vmi_int_n
System signals		vmi_present_n (in)	vmi_present_n	vmi_present_n

A. Video-In Interface:

General Description

When video data arrives through the Video-In interface, they undergo the optional decimation and filtering, packed into words of 128 bits in a FIFO before written into the memory. As writes to the memory is always aligned on a 128-bit boundary, the appropriate byte enables also need to be set with the writes. Supported pixel formats for the video-in data are YUV422 and YUV411. Both pixel formats are stored in a form of 16 bit per pixel, which means that 4 bit are unused per pixel in the case of YUV411.

Video data are stored in the Video-In frame buffers whose starting addresses are specified by the registers VidInAddr0, VidInAddr1, and VidInAddr2. VidInAddr1 and VidInAddr2 are used for double and triple buffering to avoid video tearing. However, since video is coming in at a different rate from the video refresh, switching of the video-in drawing buffers is not synchronous to the Vsync of the video refresh. At the end of each VMI frame, the vmi_int input signal will be asserted. The video processor will then switch to the next video-in frame buffer for the next VMI frame if multiple buffering is enabled. If disabled, the same video-in frame buffer will be overwritten. At the same time, the video processor also updates the VidInStatus register which indicates the VMI buffer VMI just finishes drawing (0, 1, 2), and whether the buffer contains even or odd field. An interrupt signal will signal the host for display buffer flipping for the video-in data. On the other hand, if the “Video_in data displayed as overlay enable” bit in VidProcCfg is

set, the video porcessor will do the display buffer flipping automatically for the overlay provided that all the corresponding configuration registers for the overlay is set up correctly (e.g., overlay surface enable, overlay pixel format, overlay_dudx, …… etc).

If Weave video-in deinterlaced mode is enabled, the video processor detects even/odd field from VREF(Vsync) and HREF(Hsync). If odd, the specified VidInAddr register will be used as the starting address of the video-in frame buffer. If even, VidInStride will be used as the starting address offset, and added to the specified VidInAddr. Video-in buffer will be switched at every other Vsync. VidInStride should be programmed to contain the a value which equals to 1X the regular line stride regardless of whether the video-in data is interlaced or not.

1. VMI

-data:

8-bit YCbCr interface is used. The data format is CCIR-656 YCbCr 422, and pixels arrive in the style of (Cb0[7:0] or U0[7:0]) -> Y0[7:0] -> (Cr0[7:0] or V0[7:0]) -> Y1[7:0].

Video data may be interlaced.

-timing:

Timing signals include VREF, HREF, VACTIVE, and PIXCLOCK.

VREF and HREF are active high VSYNC and HSYNC. If HREF is high during the falling edge of VREF, the field is even. If HREF is low at that time, the field is odd.

VACTIVE is a blanking signal which indicates pixel data is valid across the YCbCr bus.

Video Limitation

1. In 1x mode, 3 streams of pixel fetching will consume more memory bandwidth than available for 32-bit desktop. This means chroma-keying and bilinear filtering cannot be turned on simultaneously for 32-bit desktop.

2. In 2x mode (for any display larger than 1280 X 1024) where we refresh 2 screen pixels per cycle at 110MHz, bilinear filtering is not supported. All backend zoom (magnification) is done by point sampling (replication).

3. 1 - 10X backend zoom (magnification) with increments of 0.1X. Larger magnification is supported, but with bigger increments.

   1 to 1/16X video-in decimation (minimization) with increments of 0.015X.

4. Retain the 3-bit tap filter for RGB565 dithered as an alternative to the 2x2 box filter.

5. Interlaced video output mode is not implemented.

6. Hw cursor is 2 color only.

7. YUV 411 pixel format will be stored as unpacked in the frame buffer. This means each pixel will occupy 16 bits instead of 12 bits. This makes pixel extraction easier, but consumes more memory.

8. Video with YUV 422 format needs to be stored on a 4-byte memory boundary while YUV 411 on a 8-byte boundary. This is necessary because UV are shared between 2 pixels in 422 while UV are shared between 4 pixels in 411.

10.AGP/CMD Transfer/Misc Registers

Memory Base 0: Offset 0x0080000

Register Name	Address	Bits	R/W	Description
AGP				
agpReqSize	0x000(0)	19:0	R/W	Size of AGP request
agpHostAddressLow	0x004(4)	31:0	R/W	Host address bits 31:0
agpHostAddressHigh	0x008(8)	31:0	R/W	Host address bits 35:32
agpGraphicsAddress	0x00C(12)	24:0	R/W	Graphics address bits 24:0
agpGraphicsStride	0x010(16)	16:0	R/W	Graphics stride
agpMoveCMD	0x014(20)	n/a	W	Begin AGP transaction
Misc				
yuvBaseAddress	0x100(256)	25:0	R/W	YUV planar base address
yuvStride	0x104(260)	12:0	R/W	Y, U and V planes stride value

agpReqSize

agpReqSize defines the AGP packet transfer size. The maximum transfer size is 4-Mbyte block of data. This register is read write and has no default value.

Bit	Description
19:0	reserved. Default is 0x0.

agpHostAddressLow

During AGP transfers this address defines the source address bits 31:0 of AGP memory to fetch data from. AGP addresses are 36-bits in length and are byte aligned. The upper 4 bits reside in the **agpHostAddressHigh** register. This register is read write, and defaults to 0.

Bit	Description
31:0	Lower 32 bits of AGP memory. Default is 0x0.

agpHostAddressHigh

The **agpHostAddressHigh** defines the stride, width, and upper 4-bits of source AGP address, during AGP transfers. Stride and width are defined in quadwords. This register is read write, and defaults to 0.

Bit	Description
13:0	AGP Width. Default is 0x0.
27:14	AGP Stride. Default is 0x0.
31:28	Upper 4 bits of AGP memory. Default is 0x0.

agpGraphicsAddress

agpgraphicsAddress defines the destination frame buffer address and type of the AGP transfer. At the beginning of an AGP transfer this address is loaded into an internal address pointer that increments for each data received over AGP. This register is read write, and defaults to 0.

Bit	Description
25:0	Frame buffer offset. Default is 0x0.

agpGraphicsStride

agpGraphicsStride defines the destination stride in bytes of the AGP transfer. Stride is in multiples of bytes. This register is read write, and defaults to 0.

Bit	Description
14:0	Frame buffer Stride. Default is 0x0.

agpMoveCMD

agpMoveCMD starts an AGP transfer. When started **agpHostAddress** is loaded into the source pointer and **agpGraphicsAddress** is loaded into the destination pointer. The source pointer is incremented after data is fetched from AGP memory and written into frame buffer memory addresses by the destination pointer. The destination pointer is then incremented after the data has been written. This register is write only and has no default.

Bit	Description
2:0	reserved
4:3	Dest memory type (0=Linear FB, 1=planar YUV, 2=3D LFB, 3 = texture port). Default is 0x0.
5	Command stream ID. This bit defines which command fifo when using a host initiated AGP data move. Default is 0x0.

yuvBaseAddress

yuvBaseAddress register contains the starting frame buffer location of the yuv aperture.

Bit	Description
24:0	YUV base Address. Default is 0x0.

yuvStride

yuvStride register contains the destination stride value of the U and V planes.

Bit	Description
13:0	Y, U and V stride register. Default is 0x0.
30:14	reserved (must be 0)
31	Reserved (must be 0)

11. AGP/PCI Configuration Register Set

Register Name	Addr	Bits	Description
Vendor_ID	0	15:0	3Dfx Interactive Vendor Identification
Device_ID	2	15:0	Device Identification
Command	4	15:0	PCI bus configuration
Status	6	15:0	PCI device status
Revision_ID	8	7:0	Revision Identification
Class_code	9	23:0	Generic functional description of PCI device
Cache_line_size	12	7:0	Bus Master Cache Line Size
Latency_timer	13	7:0	Bus Master Latency Timer
Header_type	14	7:0	PCI Header Type
BIST	15	7:0	Build In Self-Test Configuration
memBaseAddr0	16	31:0	Memory Base Address (Init/3D/2D regs)
memBaseAddr1	20	31:0	Memory Base Address (LFB)
ioBaseAddr	24	31:0	I/O Base Address
Reserved	28-43		Reserved
subVendorID	44-45	15:0	Subsystem Vendor ID
subSystemID	46-47	15:0	Subsystem ID
romBaseAddr	48	31:0	Expansion Rom Base Address
Capabilites Ptr	52	31:0	Pointer to start of New Capabilities structure [7:0]
Reserved	56-59		Reserved
Interrupt_line	60	7:0	Interrupt Mapping

Interrupt_pin	61	7:0	External Interrupt Connections
Min_gnt	62	7:0	Bus Master Minimum Grant Time
Max_lat	63	7:0	Bus Master Maximum Latency Time
fabID	64	2:0	Fab Identification
Reserved	68-75		Reserved
cfgStatus	76	31:0	Aliased memory-mapped **status** register
cfgScratch	80	31:0	Scratch pad register
AGP Cap_ID	84	31:0	AGP Capability identifier register (read only)
AGP status	88	31:0	AGP status register (read only)
AGP_Cmd	92	31:0	AGP command register (read / write)
ACPI Cap ID	96	31:0	ACPI Capability identifier register (read only)
ACPI cntrl/status	100	31:0	ACPI Control and Status register (read / write)
Reserved	92-255	n/a	Reserved

Vendor_ID Register

The **Vendor_ID** register is used to identify the manufacturer of the PCI device. This value is assigned by a central authority that will control issuance of the values. This register is read only.

Bit	Description
7:0	3Dfx Interactive Vendor Identification. Default is 0x121a.

Device_ID Register

The **Device_ID** register is used to identify the particular device for a given manufacturer. This register is read only.

Bit	Description
15:0	Banshee Device Identification. Default is 0x3.

Command Register

The **Command** register is used to control basic PCI bus accesses. See the PCI specification for more information. Bit 0,1 and 5 are R/W, and bits 15:6 and 4:2 are read only.

Bit	Description
0	I/O Access Enable. Default is 0.
1	Memory Access Enable (0=no response to memory cycles). Default is 0.
2	Master Enable. Default is 0.
3	Special Cycle Recognition. Default is 0.
4	Memory Write and Invalidate Enable. Default is 0.
5	Palette Snoop Enable. Default is 0.
6	Parity Error Respond Enable. Default is 0.
7	Wait Cycle Enable. Default is 0. (strapped)
8	System Error Enable. Default is 0.
15:9	reserved. Default is 0x0.

Status Register

The **Status** register is used to monitor the status of PCI bus-related events. This register is read only and is hardwired to the value 0x0.

Bit	Description
3:0	Reserved. Default is 0x0.
4	New Capabilities (AGP/ACPI). Default is 1 for AGP/ACPI (Strapped)
5	66 Mhz Capable. Default is 0 for PCI 33 Mhz 1 for AGP (Strapped)
6	UDF supported. Default is 0.
7	Fast Back-toBack capable. Default is 0. (Strapped)
8	Data Parity Reported. Default is 0.
10:9	Device Select Timing. Default is 0x0.
11	Signaled Target Abort. Default is 0.
12	Received Target Abort. Default is 0.
13	Received Master Abort. Default is 0.
14	Signaled System Error. Default is 0.
15	Detected Parity Error. Default is 0.

Revision_ID Register

The **Revision_ID** register is used to identify the revision number of the PCI device. This register is read only.

Bit	Description
7:0	Banshee Revision Identification. (0=na, 1=.35u, 2 = .25u)

Class_code Register

The **Class_code** register is used to identify the generic functionality of the PCI device. See the PCI specification for more information. This register is read only.

Bit	Description
23:0	Class Code. Default is 0x3.

Cache_line_size Register

The **Cache_line_size** register specifies the system cache line size in doubleword increments. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0.

Bit	Description
7:0	Cache Line Size. Default is 0x0.

Latency_timer Register

The **Latency_timer** register specifies the latency of bus master timeouts. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0.

Bit	Description
7:0	Latency Timer. Default is 0x0.

Header_type Register

The **Header_type** register defines the format of the PCI base address registers (**memBaseAddr** in Banshee). Bits 0:6 are read only and hardwired to 0x0. Bit 7 of **Header_type** specifies Banshee as a single function PCI device.

Bit	Description
6:0	Header Type. Default is 0x0.
7	Multiple-Function PCI device (0=single function, 1=multiple function). Default is 0x0.

BIST Register

The **BIST** register is implemented by those PCI devices that are capable of built-in self-test. Banshee does not provide this capability. This register is read only and is hardwired to 0x0.

Bit	Description
7:0	BIST field and configuration. Default is 0x0.

memBaseAddr0 Register

The **memBaseAddr** register determines the base address for all PCI memory mapped accesses to Banshee. Writing 0xffffffff to this register will reset it to its default state. Once **memBaseAddr** has been reset, it can be probed by software to determine the amount of memory space required for Banshee. A subsequent write to **memBaseAddr** will set the memory base address for all PCI memory accesses. See the PCI specification for more details on memory base address programming. Banshee requires 32 Mbytes of address space for memory mapped accesses. For memory mapped accesses on the 32-bit PCI bus, the contents of **memBaseAddr** are compared with the **pci_ad** bits 31..25 (upper 7 bits) to determine if Banshee is being accessed. This register is R/W.

Bit	Description
31:0	Memory Base Address. Default is 0xf8000000.

memBaseAddr1 Register

The **memBaseAddr** register determines the base address for all PCI memory mapped accesses to Banshee. Writing 0xffffffff to this register will reset it to its default state. Once **memBaseAddr** has been reset, it can be probed by software to determine the amount of memory space required for Banshee. A subsequent write to **memBaseAddr** will set the memory base address for all PCI memory accesses. See the PCI specification for more details on memory base address programming. Banshee requires 32 Mbytes of address space for memory mapped accesses. For memory mapped accesses on the 32-bit PCI bus, the contents of **memBaseAddr** are compared with the **pci_ad** bits 31..25 (upper 7 bits) to determine if Banshee is being accessed. This register is R/W.

Bit	Description
31:0	Memory Base Address. Default is 0xf8000008.

ioBaseAddr Register

The **memBaseAddr** register determines the base address for all PCI IO mapped accesses to Banshee. Writing 0xffffffff to this register will reset it to its default state. Once **ioBaseAddr** has been reset, it can be probed by software to determine the amount of io space required for Banshee. A subsequent write to

ioBaseAddr will set the IO base address for all PCI memory accesses. See the PCI specification for more details on IO base address programming. Banshee requires 256 Bytes of address space for IO mapped accesses. For IO mapped accesses on the 32-bit PCI bus, the contents of **ioBaseAddr** are compared with the **pci_ad** bits 31..8 (upper 24 bits) to determine if Banshee is being accessed. This register is R/W.

Bit	Description
31:0	IO Base Address. Default is 0xffffff01.

subVendorID Register

The **subVendorID** register defines the board manufacturer ID. During system initialization the expansion code located at **romBaseAddr** will set this register to the appropriate value. This register is read during plug and play initialization. See the PC97 specification for more details on **subVendorID** and plug and play requirements. The default value for this register is automaticaly loaded after reset from the ROM. Bits 7:0 are stored in ROM location 0x7ff8, while bits 15:8 are stored in 0x7ff9 for a 32K ROM. Bits 7:0 are stored in ROM location 0xfff8, while bits 15:8 are stored in 0xfff9 for a 64K ROM.

Bit	Description
15:0	Subsystem Vendor ID register, Initialized by expansion prom, default is read by ROM.

subSystemID Register

The **subSystemID** register defines the board type. During system initialization, the expansion code located at **romBaseAddr** will set this register to the appropriate value. This register is read during plug and play initialization. See the PC97 specification for more details on **subSystemID** and plug and play requirements. The default value for this register is automaticaly loaded after reset from the ROM. Bits 7:0 are stored in ROM location 0x7ffa, while bits 15:8 are stored in 0x7ffb for a 32K ROM. Bits 7:0 are stored in ROM location 0xfffa, while bits 15:8 are stored in 0xfffb for a 64K ROM.

Bit	Description
15:0	Subsystem ID register, Initialized by expansion prom, default is read by ROM.

romBaseAddr Register

The **romBaseAddr** register determines the base address for all PCI ROM accesses to Banshee. Writing 0xfffffffe to this register will reset it to its default state. Once **romBaseAddr** has been reset, it can be probed by software to determine the amount of ROM space required for Banshee. A subsequent write to **romBaseAddr** will set the ROM base address for all PCI memory accesses. See the PCI specification for more details on memory base address programming. Banshee requires 32 to 64 Kbytes of address space for ROM accesses and is configured by strapping bit 2. For ROM accesses on the 32-bit PCI bus, the contents of **romBaseAddr** are compared with the **pci_ad** bits 31..16 (upper 16 bits) to determine if Banshee is being accessed. This register is R/W.

Bit	Description
31:0	Expansion Rom Base Address. Default is 0xffff8000 or 0xffff0000.

Capabilities Pointer

The Capabilities pointer register contains the offset in configuration space of beginning of the capability link list structure. This register is read only.

Bit	Description
31:0	Capabilities Pointer offset. Default is 0x00000054 if AGP is enabled via the strapping bits, otherwise it is 0x60.

Interrupt_line Register

The **Interrupt_line** register is used to map PCI interrupts to system interrupts. In a PC environment, for example, the values of 0 to 15 in this register correspond to IRQ0-IRQ15 on the system board. The value 0xff indicates no connection. This register is R/W.

Bit	Description
0:7	Interrupt Line. Default is 0x5 (IRQ5)

Interrupt_pin Register

The **Interrupt_pin** register defines which of the four PCI interrupt request lines, INTA* - INTRD*, the PCI device is connected to. This register is read only and is hardwired to 0x1.

Bit	Description
0:7	Interrupt Pin. Default is 0x1 (INTA*)

Min_gnt Register

The **Min_gnt** register specifies the burst period a PCI bus master requires. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0 since Banshee does not support bus mastering.

Bit	Description
7:0	Minimum Grant. Default is 0x0.

Max_lat Register

The **Max_lat** register specifies the maximum request frequency a PCI bus master requires. It must be implemented by devices capable of bus mastering. This register is read only and is hardwired to 0x0 since Banshee does not support bus mastering.

Bit	Description
7:0	Maximum Latency. Default is 0x0.

fabID Register

Identification code of the manufacturing plant.

Bit	Description
3:0	Manufacturing fab identification. Read only. (1 = TSMC)
31:4	Scratch pad. (read / write)

cfgStatus Register

The **cfgStatus** register is an alias to the normal memory-mapped status register. See section x.x for a description of the status registers. Reading the configuration space **cfgStatus** register returns the same data as if reading from the memory-mapped status register.

cfgScratch Register

The **cfgScratch** register can be used as scratch pad storage space by software. The values of **cfgScratch** are not used internally to alter functionality, so any value can be stored to and read from **cfgScratch**.

Bit	Description
31:0	Scratchpad register. Default is 0x0.

New capabilities (AGP and ACPI)

AGP and ACPI Use PCI 's new capabilities mechanism. The New Capabilities structure is implemented as a linked list of registers containing information for each function supported by Banshee. The list contains both AGP status and command registers. AGP registers read back '0' if AGP is disabled via the strapping pins.

Capability Identifier Register

The capability register resides at offset (CAP_OFFSET). This register identifies AGP revision compliance

Bit	Description
7:0	Capability ID. Always == 2 for AGP
15:8	Next Capability ID Pointer. Default is 0x60.
19:16	Minor AGP revision, the interface conforms to
23:20	Major AGP revision, the interface conforms to
31:24	Reserved. Defined as 0.

AGP Status

AGP status register documents maximum number of requests that Banshee can manage, AGP sideband capable, and transfer rate

Bit	Description
1:0	Data rates that Banshee can deliver/receive. Bit[0] = 1x, bit[1] = 2x. Default is 1.
4:2	Reserved. Default is 0
5	AGP_4G. AGP supports above 4 Giga bytes of memory. Default is 1.
8:6	Reserved. Default is 0.
9	SBA. Device supports side band addressing
23:10	Reserved. Default is 0
31:24	RQ_DEPTH. Max # of requests that Banshee can manage. Default is 7.

AGP Command

AGP status register documents maximum number of requests that Banshee can manage, AGP sideband capable, and transfer rate

Bit	Description
2:0	Data Rate bit[0] = 1x, bit[1] = 2x. Only 1 bit must be set (Read/Write)
4:3	Reserved. Default is 0
5	AGP_4G_ENABLE. AGP supports above 4 Giga bytes of memory. Default is 0.
7:6	Reserved. Default is 0
8	AGP enable. Enables AGP function. AGP_RESET sets this bit to 0. (R/W)
9	SBA_ENABLE. Enable side band addressing mechanism. (R/W)
23:10	Reserved. Default is 0
31:24	RQ_DEPTH. Max # of requests System can handle. (R/W)

ACPI Cap ID

The ACPI Cap ID register identifies what Banshee supports in ACPI.

Bit	Description
7:0	Capability ID. Always == 1 for ACPI
15:8	Next Capability ID Pointer. Default is 0
18:16	Version. Default is 0x1.
19	PME Clock. Default is 0.
20	Aux Power Source. Default is 0.
21	DSI. Default is 1. Indicates additional software initialization must take place.
24:22	Reserved. Default is 0
25	D1 Support. Default is 0.
26	D2 Support. Default is 0.
31:27	PME Support. Default is 0.

ACPI Ctrl/Status

ACPI status register allows transition from the D3 to D0 state.

Bit	Description
1:0	Power State. Defaults to 0x0. Banshee only accepts writes of 0x0 or 0x3 to these bits. (R/W)
7:2	Reserved. Default is 0
8	Sticky bit. Default is 0.
12:9	Data Select. Default is 0.
14:13	Data Scale. Default is 0.
15	Sticky bit. Default is 0.
21:16	Reserved. Default is 0
22	B2 B3 support. Default is 0.
23	BPCC_En. Default is 0.
31:24	Data for data select. Default is 0.

12. Init Registers

Register Name	I/O Address	Bits	R/ W	Description
status	00-03	31:0	R	Banshee status register
pciInit0	04-07	31:0	R/W	PCI initialization register
lfbMemoryConfig	0c-0f	31:0	R/W	Should remain 0xa2200
miscInit0	10-13	31:0	R/W	Misc. initialization register
miscInit1	14-17	31:0	R/W	Misc. initialization register
dramInit0	18-1b	31:0	R/W	Dram initialization register0
dramInit1	1c-1f	31:0	R/W	Dram initialization register1
agpInit0	20-23	31:0	R/W	AGP initialization register
textureGRXBackend	24-27	31:0	R/W	Texture Cache initialization register
vgaInit0	28-2b	31:0	R/W	VGA initialization register
vgaInit1	2c-2f	31:0	R/W	VGA initialization register
2d_command	30-33	31:0	W	2d command register (to be used to write SGRAM mode and special mode registers)
2d_srcBaseAddr	34-37	31:0	W	2d srcBaseAddr register (to be used to write to SGRAM mode and special mode registers)
strapInfo	38-3b	11:0	R	Strap bits after power up.
reserved	3c-3f	na		

status Register (0x0)

The **status** register provides a way for the CPU to interrogate the graphics processor about its current state and FIFO availability. The **status** register is read only, but writing to **status** clears any Banshee generated PCI interrupts.

Bit	Description
4:0	PCI FIFO freespace (0x1f=FIFO empty). Default is 0x1f.
5	PCI FIFO busy. Default is 0x0.
6	Vertical retrace (0=Vertical retrace active, 1=Vertical retrace inactive). Default is 1.
7	FBI graphics engine busy (0=engine idle, 1=engine busy). Default is 0.
8	TREX busy (0=engine idle, 1=engine busy). Default is 0.
9	Banshee busy (0=idle, 1=busy). Default is 0.
10	2D busy (0=idle, 1=busy). Default is 0.
11	Cmd fifo 0 busy. Default is 0x0.
12	Cmd fifo 1 busy. Default is 0x0.
27:13	reserved
30:28	Swap Buffers Pending. Default is 0x0.
31	PCI Interrupt Generated. Default is 0x0.

Bits(4:0) show the number of entries available in the internal host FIFO. The internal host FIFO is 32 entries deep. The FIFO is empty when bits(4:0)=0x1f. Bit(6) is the state of the monitor vertical retrace signal, and is used to determine when the monitor is being refreshed. Bit(7) of **status** is used to determine if the graphics engine of FBI is active. Note that bit(7) only determines if the graphics engine of FBI is busy – it does not include information as to the status of the internal PCI FIFOs. Bit(8) of **status** is used

to determine if TREX is busy. Note that bit(8) of **status** is set if any unit in TREX is not idle – this includes the graphics engine and all internal TREX FIFOs. Bit(9) of **status** determines if all units in the Banshee system (including graphics engines, FIFOs, etc.) are idle. Bit(9) is set when any internal unit in Banshee is active (e.g. graphics is being rendered or any FIFO is not empty). Bit(10) of **status** is used to determine if the 2D graphics engine is active. Bits(11:10) of **status** is used to determine if either command fifo 0 or command fifo 1 are active. When a SWAPBUFFER command is received from the host cpu, bits (30:28) are incremented – when a SWAPBUFFER command completes, bits (30:28) are decremented. Bit(31) of status is used to monitor the status of the PCI interrupt signal. If Banshee generates a vertical retrace interrupt (as defined in **pciInterrupt**), bit(31) is set and the PCI interrupt signal line is activated to generate a hardware interrupt.

pciInit0 Register (0x4)

pciInit0 register contains the control information on how PCI should behave. Bits 15:0 are the output of the counter clocked by GRX clock. Bits 19:18 control Interrupts. Bits 17:13 allow the retry interval to be increased, while bits 12 and 11 allow retries to be disabled. Biys 9 and 8 determint the bus performance. Bits 6:2 determine the PCI fifo Low water mark. This value should never be 0 (no overflow checking is done) and should be set greater than 2 for any fast device operations. Bits 25:20 control how many non modal LFB accesses are grouped together before being pushed to memory. This register is read write and defaults to 0x01800040.

Bit	Description
1:0	Reserved
6:2	PCI FIFO Empty Entries Low Water Mark. Valid values are 0-31. Default is 0x10.
7	Reserved. Default is 0x1.
8	Wait state cycles for PCI read accesses (0=1 ws, 1=2 ws). Default is 0x0.
9	Wait state cycles for PCI write accesses (0=no ws, 1=one ws). Default is 0x0.
10	Reserved. Default is 0x0.
11	Disable PCI IO access retries. . Default is 0x0.
12	Disable PCI Memory access retries. Default is 0x0.
17:13	Retry interval, less 8 clocks. Default is 0.
18	PCI Interrupt Enable Default is 0.
19	PCI Interrupt Time Out Enable. Default is 0.
25:20	PCI Fifo read threshold. Default is 0x18.
26	Force PCI/CMD Frame buffer accesses to high priority. (1= high, 0 = low, except for PCI frame buffer reads). . Default is 0x0.

lfbMemoryConfig Register (0xC)

This register defaults to 0xa2200.

miscInit0 Register (0x10)

miscInit0 contains resets to all subsystems, pixel swizzling, and Y origin subtraction. Bits [1:0] reset the 3D graphics subsytem. Bits [3:2] enable byte/word swizzling during register accesses to 2D or 3D. Bits [6:4] define resets for video, 2D, and memory subsytems. Bits[29:18] define the Y origin subtraction

value used during address calculation when Y flip is enabled in fbzMode. Bits [31:30] enable byte/word swizzling during non modal LFB reads and writes

Bit	Description
	Miscellaneous Control
0	FBI Graphics Reset (0=run, 1=reset). Default is 0.
1	FBI FIFO Reset (0=run, 1=reset). Default is 0. [resets PCI FIFO and the PCI data packer]
2	Byte swizzle incoming register writes (1=enable). [Register byte data is swizzled if **miscInit0**[2]==1 and pci_address[20]==1] for 3D registers, and pci_address[19] ==1 for 2D registers. Default is 0.
3	Word swizzle incoming register writes (1=enable). [Register word data is swizzled if **miscInit0**[2] == 1 and pci_address[20] ==1] for 3D registers, and pci_address[19] ==1 for 2D registers. Default is 0.
4	Video Timing Reset (0=run, 1=reset). Default is 0.
5	2D Graphics Reset (0=run, 1=reset). Default is 0.
6	Memory Timing Reset (0=run, 1=reset). Default is 0.
7	VGA Video Timing Reset (0=run, 1=reset). Default is 0.
10:8	Programmable delay to be added to the blank signal before it outputs to the TV out interface. This is in terms of number of flops clocked by the 2x clock. The objective is to synchronize the blank signal with the data output by matching the CLUT delay. Default is 0x0. 000 = 2 flops; 001 = 3 flops; …… 111 = 9 flops
13:11	Programmable delay to be added to the vsync and hsync signals before they are output to the TV out interface. This is in terms of number of flops clocked by the 2x clock. The objective is to synchronize the sync signals with the data output by matching the CLUT delay. Default is 0x0. 000 = 2 flops; 001 = 3 flops; …… 111 = 9 flops
16:14	Programmable delay to be added to the vsync and hsync signals before they are output to the monitor. This is in terms of number of flops clocked by the 2x clock. The objective is to synchronize the sync signals with the data output by matching the delay through the CLUT and DAC. Default is 0x0. 000 = 2 flops; 001 = 3 flops; …… 111 = 9 flops
17	Reserved
	Y Origin Definition bits
29:18	Y Origin Swap subtraction value (12 bits). Default is 0x0.
30	Byte swizzle incoming non modal LFB writes (1=enable). Default is 0.
31	Word swizzle incoming non modal LFB writes (1=enable). Default is 0.

miscInit1 Register (0x14)

miscInit1 register controls miscellaneous operations of Banshee available in real mode. Bit 0 is used to correct for CLUT addresses being inverted during host accesses. This bit should be set to 1 for proper operation. Bit 3 enables and disables writes to the PCI **subVendorID** and **subSystemID** registers. Bit 4 enables writes to the ROM through **romBaseAddr**. Bit 5 enables the new triangle address aliasing allowing better address compaction. Bit 6 disables texture mapping.

Power down of H3 is controlled by bits 11:7, where bit 7 powers down the color lookup tables, bit 8 powers down the DAC itself, bits 9, 10, and 11 power down the three PLL's.

Bits 17 and 18 disable stalling on the opposite pipe (either 2D or 3D) when a command is sent down. These bits are used for testing, and should not be set during normal operation.

Bit 19 is used to terminate command fifo activity. Setting this bit to '1' halts the command fifo and resets all of the registers in the command register space to their default values. In order for Banshee to be shut down gracefully, this bit should only be set when Banshee is idle. Be sure to restore this bit to 0 when finished.

Bits 28 through 24 indicate the value of the strapping registers at boot up. Note that altering these bit effect the read back information of PCI and AGP resource reporting. For more information on the strapping registers, see the section on Power on Strapping

Bit	Description
	Miscellaneous Control
0	invert_clut_address. Default = 0.
2:1	tri_mode - triangle iterator mode. Default is 0x0.
3	Enable Sub Vendor/ Subsystem ID writes. (0=disable, 1=enable). Default is 0x0.
4	Enable ROM writes. (0=disable, 1=enable). Default is 0x0.
5	Alternate triangle addressing map (0=disable, 1=enable). Default is 0x0.
6	Disable texture mapping (0=enable, 1=disable). Default is 0x0.
	Power Down Control
7	Power Down CLUT. Default is 0x0.
8	Power Down DAC. Default is 0x0.
9	Power Down Video PLL. Default is 0x0.
10	Power Down Graphics PLL. Default is 0x0.
11	Power Down Memory PLL. Default is 0x0.
	2D Block Write Control
14:12	Block write threshold. Default is 0x0.
15	Disable 2D Block write. Default is 0x0.
16	Disable 2D stall on 3D synchronous dispatch. When set to 1, 2D will not wait on pending 3D operations to complete before being issued. Default is 0x0.
17	Disable 3D stall on 2D synchronous dispatch. When set to 1, 3D will not wait on pending 2D operations to complete before being issued. Default is 0x0.
18	Reserved.
19	Command Stream Reset (1=reset command streams, 0 = normal operation). Default is 0x0.
23:20	Reserved
24	PCI Fast device. Default strapped on VMI_DATA 0
25	PCI BIOS Size. Default strapped on VMI_DATA 1
26	PCI 66Mhz. Default strapped on VMI_DATA 2.
27	AGP Enabled. Default strapped on VMI_DATA 3.
28	PCI Device Type. Default strapped on VMI_DATA 4.
	TV Out clock delay adjust
29	tv_out_clk_inv - invert delayed clock. Default = 0.
31:30	tv_out_clk_del_adj - choose between 4 values of delay. Default = 0.

dramInit0 Register (0x18)

dramInit0 controls the sgram interface timing of specific timing parameters. The default value of this register is 0x00579d29.

Bit	Description
	Sgram access timing
1:0	tRRD - row active to row active (1-4 clks). Default is 0x1 (2 clks)
3:2	tRCD - RAS to CAS delay (1-4 clks). Default is 0x2 (3 clks).
5:4	tRP - row precharge (1-4 clks). Default is 0x2 (3 clks).
9:6	tRAS - minimum active time (1-16 clks). Default is 0x4 (5 clks).
13:10	tRC - minimum row cycle time (1-16 clks). Default is 0x7 (8 clks).
15:14	tCAS latency (1-4 clks). Default is 0x2 (3 clks).
16	tMRS mode and special mode register cycle time (1-2 clks).. Default is 0x1 (2 clks)
17	tDQR Rd to DQM assertion delay (0-1 clks). Default is 0x1 (1 clk)
18	tBWC Block write cycle time (1-2 clks). Default is 0x1 (2 clks)
19	tWL WR to pre (1-2 clks). Default is 0x0 (1 clk)
21:20	tBWL BKWR to Pre (1-4 clks). Default is 0x1 (2 clks)
22	tRL RD to PRE (1-2 clks). Default is 0x1 (2 clks)
23	dont allow WR/BKWR to terminate RD, use BST. Default is 0x0 (allow wr-term)
24	Disable the dead bus cycle btw. RD and WR (0=enable, 1=disable). Default is 0x0
25	SGRAM write per bit enable (0=disable, 1=enable). Default is 0x0
26	Number of Sgram chipsets (0=1, 1=2). Default is 0x0. (power on strap = VMI_DATA_5)
27	Sgram type (0=8Mbit, 1=16Mbit). Default is 0x0. (power on strap = VMI_DATA_6)

dramInit1 Register (0x1C)

Bit	Description
	SGRAM Refresh Control
0	Refresh Enable (0=disable, 1=enable). Default is 0.
9:1	Refresh_load Value. (Internal 14-bit counter 5 LSBs are 0x0) Default is 0x100.
	Video Refresh Control
10	Video arbitration priority. (0=normal, 1=aggressive) Default is 0
	Miscellaneous video Control
11	Triple buffer enable (0=double buffering, 1=triple buffering). Default = 0.
12	Dither passthrough (0=dithering, 1=bypass dithering). Default = 0.
	SGRAM read data sampling control
13	sg_clk_nodelay - bypass the delay element. Default = 1.
14	sg_use_inv_sample - resample the flopped sgram data with another negative-edge flop before flopping data with mclk. Default = 0.
15	sg_del_clk_invert - invert delayed clock before using it. Default = 0.
19:16	sg_clk_adj - delay value for sgram read data sample clock. Default = 0x0. (2-62 NAND gates of delay, in steps of 4)
	SGRAM frame buffer output delay control (control + data bits)
23:20	sg_oclk_del_adj - Delay amount for clock out to SGRAMs. Default = 0xf.
24	sg_oclk_nodelay - forces clock out to SGRAMs to have minimum delay. (0=delayed, 1=nodelay). Default = 0.
	Memory Controller configuration bits
25	mctl_short_power_on. Power on in 128 cycles. Default = 0. VMI_ADDR_1
26	mctl_no_aggressive - turn off mem_ctrl's aggressive row activation. Default = 0.
27	mctl_pagebreak - force a pagebreak for all accesses. Default = 0.
28	mctl_tristate_outputs - force data outputs to be tristate. Default = 0.
29	mctl_no_vin_locking - prevent vin from locking the bus during requesting. 0=allow locking, 1=prevent locking. Default = 0.
30	mctl_type_sdram - (0=use SGRAMs, 1=use SDRAMs). Default = 0. VMI_ADDR_2

When using SGRAMs, *mctl_type_sdram* should be set to 0. When using SDRAMs, only 16Mbit (16x512K parts) are supported, which result in a 16MB frame buffer. The *sgram_type* and *sgram_chipsets* bits in **dramInit0** are ignored when *mctl_type_sdram*=1.

Note that the **fastfillCMD** behaves differently when *mctl_type_sdram*=1 (**dramInit1[30]).** When fastfilling with SGRAMs (*mctl_type_sdram*=0), if dithering is enabled and **fastfillCMD[0]=**1, no dithering will happen. But when fastfilling with SDRAMs (*mctl_type_sdram*=1), if dithering is enabled and **fastfillCMD[0]**=1, dithering will still happen, since SDRAMs do not support blockwriting.

agpInit0 Register (0x20)

The **agpInit0** register is used to control how AGP behaves when making requests. Bit 0 sets the request priority level. Bits [3:1] control the largest size the request can be. Bits [6:4] determine when the agp

request fifo becomes full (requests that have not yet been issued to the AGP target). Bits [10:7] control when to much data has been returned, and AGP needs to begin stalling.

Bit	Description
0	Force AGP request to be high priority. (0=Low, 1 = High). Default is 0x0.
3:1	Maximum AGP request length. (0=1 octword, 7 = 8 octwords). Default is 0x7.
6:4	AGP request fifo full threshold. Default is 0x1.
10:7	AGP read fifo full threshold. Default is 0x9.

vgaInit0 Register (0x28)

The **vgaInit0** register is used for hardware initialization and configuration of the VGA controller in Banshee. VGA can be disabled by writing bit 0 to a "1". Bit 1 allows external video timing to drive the VGA core video scan out logic. Bit 2 controls how the VGA DAC control logic views the width of the RAM. For VGA compatibility, this bit should be set to 0 (6 bit DAC).

VGA extensions are enabled by bit 6. These extensions are mention in the VGA portion of this spec, in the CRTC register space. Bit 10 enables the ability to read back the PCI configuration when bit 6 of this register is 0.

Bit 8 determines if the chips should wake up as a VGA motherboard or an add in card. Bit 9 disables the VGA to response to legacy address decoding. This bit should be set if Banshee is not the primary display adapter in the system. By default, this bit is set if Banshee is set as a multimedia device with the strapping bits. Setting this bit also disables write access to 0x46e8 and 0x102.

Bit 12 should be set when in an extended (non-VGA) mode. This disables the VGA from fetching memory data during video raster scan out.

Bit 13 is used when an external DAC is supported. This bit should always be set to 0.

Bit 21:14 determine the start page of VGA in board memory. By default, VGA is placed at the beginning of memory. If need be, it can be moved anywhere on a 64K byte boundary within 16M bytes.

Bit 22 disables VGA refresh control of board memory. When VGA is in scan out mode, it prefers memory refresh to happen at horizontal sync time. When this bit is set to 0, three memory refresh cycles happen after HBLANK occurs, and the memory refresh time out counter is deferred. When this bit is set to 1, the memory refresh time out counter explicitly controls memory refresh events.

Bit	Description	Default
	Miscellaneous Control	
0	VGA disable. (0=Enable, 1 = Disable). Setting this bit to 1 shuts off all access to the VGA core.	0
1	Use external video timing. This bit is used to retrieve SYNC information through the normal VGA mechanism when the VGA CRTC is not providing timing control.	0
2	VGA 6/8 bit CLUT. (0= 6 bit, 1 = 8 bit).	0
5:3	Reserved.	1
6	Enable VGA Extensions	0
7	Reserved	0
8	0x46e8/0x3C3 Wake up select (0=use 0x46e8, 1=use 0x3C3 or IO Base + 0xC3). VGA add in cards that use 0x46e8 while mother board VGA uses 0x3C3. When Banshee is a multimedia device, this bit should be set to '1' and the VGA subsystem should be enabled with IO Base + 0xC3.	0
9	Disable VGA Legacy Memory/IO Decode (0=Enable, 1=Disable)	0
10	Use alternate VGA Config read back (0 = Enable, 1=Disable). Setting this bit to 0 allows the VGA to read back configuration through CRTC index 0x1c.	0
11	Enable Fast Blink (test bit) (1=fast blink, 0 = normal blink).	0x0
12	Use extended video shift out. Set this bit to 1 to disable all VGA memory access when video processor is shifting out data.	0
13	Decode 3c6. (test bit)	0
21:14	Vga base offset in 64k quantities	
22	Disable SGRAM refresh requests on HBLANK. When set to 1, the VGA does not produce memory refreshes during horizontal blanking.	0
31:23	reserved	0

vgaInit1 Register (0x2C)

The **vgaInit1** register contains the read and write apertures for VBE. VBE uses address 0xA0000 as an aperture into Banshee memory. See the section on VBE apertures in the VGA portion of this document. Bit 20 enables sequential chain mode, a pseudo packed pixel format Bits 28:21 define lock bits that disable writes to specific sections of the VGA core. See the section on register locking in the VGA portion of this document.

Bit	Description	Default
9:0	VBE write Aperture, in 32K granularity	0
19:10	VBE read Aperture, in 32K granularity	0
20	Enable 0xA0000 Sequential Chain 4 mode.	0
21	Lock Horizontal Timing - 3B4/3D4 index 0,1,2,3,4,5,1a	0
22	Lock Vertical Timing -3B4/3D4 index 6,7 (bit 7,5,3,2 and 0), 9 10, 11 (bits[3:0]), 15,16,1b.	0
23	Lock H2 - 0x3B4/0x3D4 index 17, bit 2	0
24	Lock Vsync - 0x3C2, bit 7.	0
25	Lock Hsync - 0x3C2, bit 6.	0
26	Lock Clock Select - 0x3C2, bits 3 and 2.	0
27	Lock Ram Enable - 0x3C2, bit 1	0
28	Lock Character Clock - 0x3C4, index 1 bit 0.	0

2d_Command_Register (0x30)

Writing to this register is the same as writing to the 2d unit's **command** register. This mapping is intended to provide a way to initialize the SGRAM mode and special mode registers at init time.

2d_srcBaseAddr Register (0x34)

Writing to this register is the same as writing to the 2d unit's **srcBaseAddr** register. This mapping is intended to provide a way to initialize the SGRAM mode and special mode registers at init time.

13. Frame Buffer Access

Frame Buffer Organization

The Banshee linear frame buffer base address is located in a separate memory base address register in PCI config space and occupies 32 megabytes of address space for linear access. It is assumed (but not required) that VGA will use the first 256K of linear memory, and the desktop, and video will use the remaining linear memory.

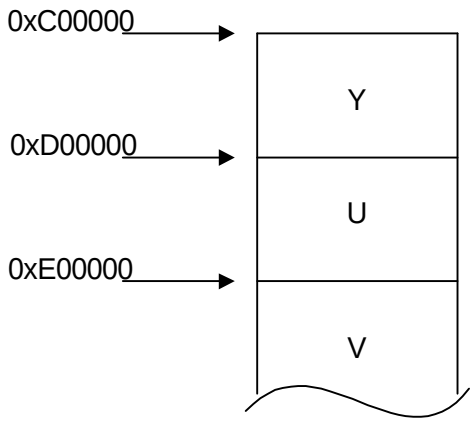
Linear Frame Buffer Access

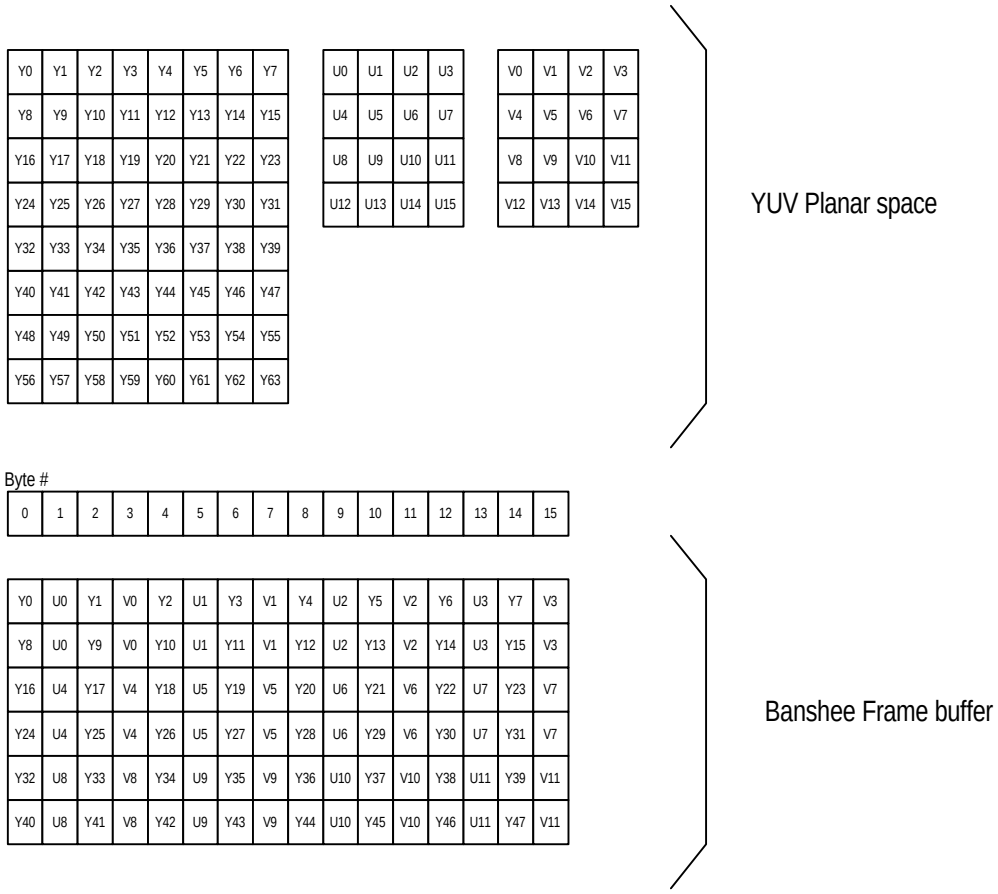
Linear frame buffer access is accessed much like system, and can store the desktop, video, 3D front buffer, 3D back buffer, 3D auxillary buffer, and textures. Memory management is done with a true linear memory manager.

YUV Planar Access

YUV planar memory allows the CPU to write Y, U, and V in separate regions of memory space. As Y, U, and V are written, they are converted into YUYV packed form, and stored in the frame buffer at the correct offset from the YUV base address register. The first megabyte region defines Y, where each 32-bit write, generates a 64-bit write on Banshee, with appropriate byte masks. The second megabyte region of YUV planar memory defines U space, where each 32-bit write generates two 64-bit writes with

appropriate byte enable bits. The third region of YUV planar memory defines the V space, where each 32-bit write generates two 64-bit writes with appropriate byte enable bits. The conversion between planar and packed is described below. YUV planar space has a fixed 1024 byte stride, and a programmable destination stride.





14. Accessing the ROM

ROM Configuration

Banshee supports either 32K or 64K of ROM space. The size of the ROM is determined during at power up by an external strapping pin (see the section of strapping pins for more information).

Directly after reset, PCI subsystem and subvendor information are loaded from the next to last four bytes of ROM memory. The last four bytes are reserved for checksum information.

ROM Reads

Banshee supports reads to the ROM through the normal PCI mechanism. In order to read the ROM, set the **romBaseAddr** register bit 0 to 1. ROM accesses are then possible at the address indicated by the most significant bits of **romBaseAddr**. ROM reads can have any combination of byte enables asserted. Since

the ROM is a byte device however, asserting multilble byte enables at once will cause the transfer of data on the PCI bus to be slow.

It is important to note the ROM shares the bus with VMI and TV out. During ROM accesses, data on these ports will become ROM information providing what may appear to be bad pixels on the display. This is normal; however, if it is know that ROM accesses are to occur, it is recommended that VMI or TV out be disabled prior to ROM access.

ROM Writes

Banshee also supports a mechanism for programming flash ROMs when they are available. The model that Banshee uses is that of a 32K/64K EEPROM that allows programming by polling the EEPROM.

By default, Banshee will not respond to writes pointed at by **romBaseAddr**. By enabling bit 0 of **romBaseAddr** and also setting bit 4 of **miscInit1**, writes pointed at by **romBaseAddr** will be processed.

Typically, programmable ROMs have a sequence of write events that must occur to be placed in the 'Program Mode'. Then either a single or multiple writes occur (depending of the ROM used) to fill in data. Finally, the ROM is polled via ROM reads, to confirm the write is complete. This process is repeated until the ROM is completely written.

For more information on how to program a specific ROM, see its data sheet or application notes.

15. Power on Strapping Pins

During power up, Banshee gets some of its configuration information from strapping pins. This information is used to control how Banshee will behave.

StrapInfo bit	**PIN**	**Description**
11	VMI_ADDR_3	Unused
10	VMI_ADDR_2	mctl_type_sdram (0=SGRAMs, 1=SDRAMs)
9	VMI_ADDR_1	mctl_short_power_on (0-normal power-on, 1=for RTL simulation onl
8	VMI_ADDR_0	re-map IDSEL (0=IDSEL is IDSEL, 1= PCI_AD_16 is IDSEL)
7	VMI_DATA_7	Disable PCI IRQ register (0=Enable, 1 = Disable).
6	VMI_DATA_6	SGRAM chip size (0=8Mb, 1=16Mb)
5	VMI_DATA_5	SGRAM number of chips (0=4 parts, 1=8 parts)
4	VMI_DATA_4	PCI Device Type (0= VGA, 1= Multimedia)
3	VMI_DATA_3	AGP Enable (0=Disabled, 1 = Enabled).
2	VMI_DATA_2	PCI 66Mhz (0 = 33Mhz, 1 =66Mhz)
1	VMI_DATA_1	BIOS Size (0=32K, 1 = 64K)
0	VMI_DATA_0	PCI Fast Device. (0=DEVSEL Medium,1= DEVSEL Fast)

16. Monitor Sense

Banshee Supports the ability to detect a monitor, as well as determine if the monitor is color or monochrome. This is accomplished with an internal MSENSE signal. MSENSE becomes active when a

current is driven through either the RED, GREEN or BLUE DAC outputs. If a monochrome monitor is present, only the GREEN output will cause MSENSE to become active. MSENSE is readable through IO 0x3c2, bit 4.

17. Hardware Initialization

- PCI Configuration
- DRAM Init
- VGA Core Wakeup
- Other Init?

18. Data Formats

Signal	Pixel Sequence for 4 : 1 : 1								Pixel Sequence for 4 : 2 : 2					
P15	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7	Y7
P14	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6	Y6
P13	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5	Y5
P12	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4	Y4
P11	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3	Y3
P10	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2	Y2
P9	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1	Y1
P8	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0	Y0
P7	U7	U5	U3	U1	U7	U5	U3	U1	U7	V7	U7	V7	U7	V7
P6	U6	U4	U2	U0	U6	U4	U2	U0	U6	V6	U6	V6	U6	V6
P5	V7	V5	V3	V1	V7	V5	V3	V1	U5	V5	U5	V5	U5	V5
P4	V6	V4	V2	V0	V6	V4	V2	V0	U4	V4	U4	V4	U4	V4
P3	0	0	0	0	0	0	0	0	U3	V3	U3	V3	U3	V3
P2	0	0	0	0	0	0	0	0	U2	V2	U2	V2	U2	V2
P1	0	0	0	0	0	0	0	0	U1	V1	U1	V1	U1	V1
P0	0	0	0	0	0	0	0	0	U0	V0	U0	V0	U0	V0