V2200

Integrated 3D / 2D / Video Accelerator

TECHNICAL SPECIFICATIONS
(including RISC PROGRAMMING)

Table of Contents

1. Introduction

The Rendition V2200 provides very high performance two-dimensional (2D), three-dimensional (3D) graphics, video decompression and VGA compatibility. By integrating a RISC processor, the chip provides PCs and other platforms with 3D animation and video decompression capabilities previously reserved for high-cost workstations. The internal RISC processor can issue multiple instructions per cycle and has an expanded graphics instruction set. The chip has a separate triangle and pixel drawing engines that perform perspective and sub-pixel correct bilinear-filtered texturing, plus blending, antialiasing and haze calculations on each pixel. The chip block diagram is shown in Figure 1.

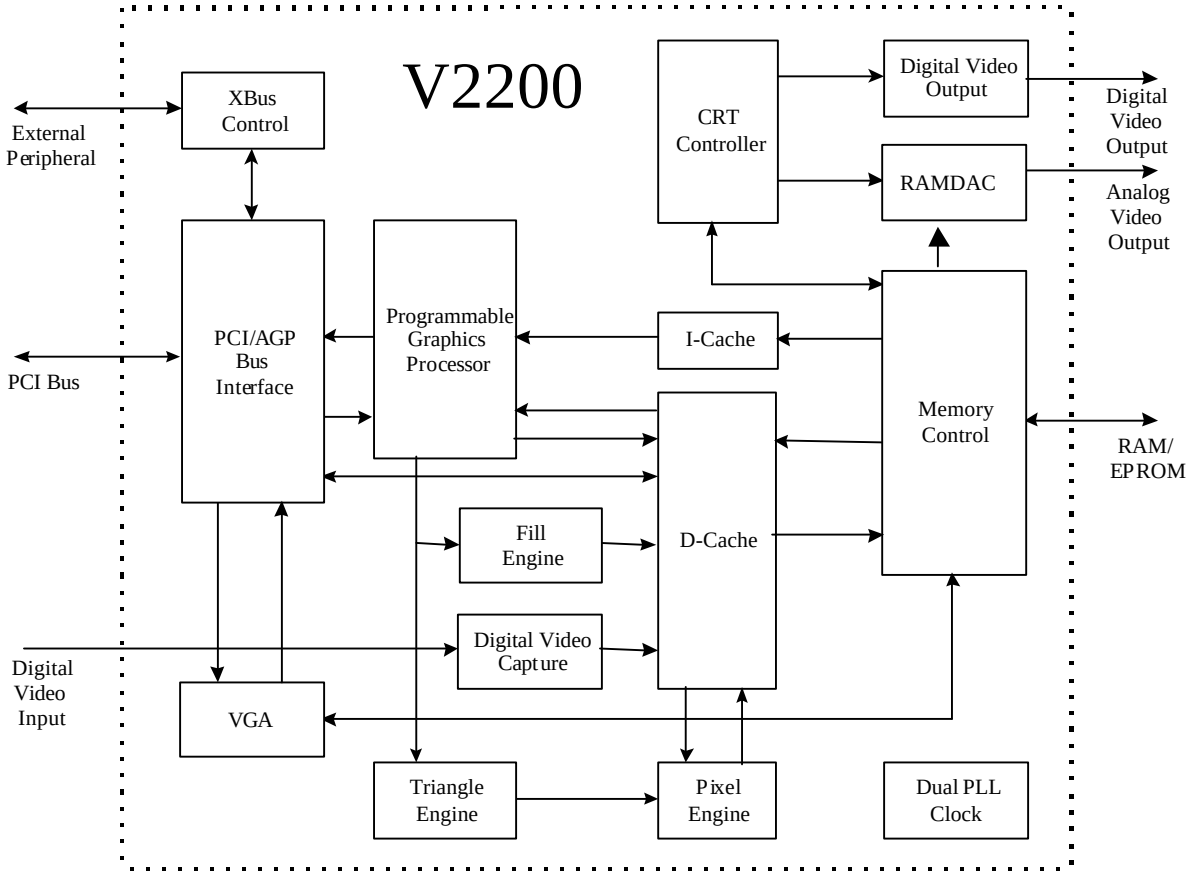


Figure 1 -- V2200 Block Diagram

2. RISC Processor

The V2200 RISC is similar to other Reduced-Instruction-Set-Computer processors and has the following key attributes:

- 32-bit interlocked integer processor.
- Single-word, multiple-operation instructions.
  - One or two integer operations.
  - Integer operation using an 8-bit unsigned-immediate value.
  - A load/store, or a load-immediate (16-bit unsigned value).
  - Branch or jump.
  - A pixel draw or a triangle draw instruction.
- Linear 32-bit address space.
- Combinatorial 32x32 multiply
- Delayed load with large register file.
- The 128 entry register-file can also be used as memory scratch-pad (direct and indirect register specification).
- Delayed non-annulling branches.
- Double-word, word, unsigned half-word and unsigned byte loads and stores (big-endian byte order).

The RISC processor can access up to 16M bytes of local memory. Some hardware control registers are mapped into local memory space. See the *PCI I/O and Memory Interface* section for memory map information.

Note, the RISC does not increment across the boundary between 0x00ffffff and 0x01000000, nor does it increment from 0xffffffff and 0. That is, it does not increment from normal memory to ROM areas, plus relative branches and relative jumps do not cross this boundary. Absolute jumps can access either ROM memory or normal memory from any location. The most significant bit of the absolute jump address specifies ROM memory if set and normal memory if cleared.

2.1 Instruction Formats

The Rendition V2200 uses 32-bit fixed-width multiple-operation instructions. There are eight basic instruction formats. Table 2.1 shows the instruction formats.

8	*8*	*8*	*8*
OP	D	S2	S1

Integer, *draw* or *jmprl* operation

8	*8*	*8*	*8*
OP	offset8	S2	S1

Store operation

8	*8*	*8*	*8*
OP	D	offset8	S1

Load operation

8	*8*	*8*	*8*
OP	S3	S2	S1

Sort operation

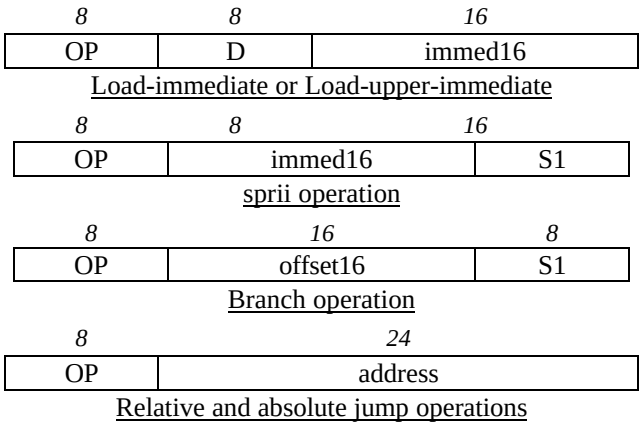


Table 2.1 -- Instruction format definitions

2.2 Register Definitions

Tables 2.2a and 2.2b define the RISC register file map. See the *Special RISC Hardware Registers* and the *Register File Segment Register* sections for additional information. Note, special registers (index < 64) are never forwarded. When writable special registers are used as a destination, they have a non-interlocked delay, as specified in their respective descriptions, before they can be used as source operands. This delay remains the same even if the operation itself has a non-zero delay (e.g. *mul timer,S2,S1* has same delay as *add timer,S2,S1*). It is illegal to use one of these registers while in the shadow of its respective hazard. Note, *%32* to *%46* are legal as a vector source but not destination.

Register	Definition
%255 or fp	Frame pointer
%254 or ra	Program link register
%253 or ira	Interrupt link register
%252 or sp	Stack pointer
%128 .. %251	General purpose registers.
%112 .. %127	Indirect access using *Seg3* to registers %128 to %255 (only). See text.
%96 .. %111	Indirect access using *Seg2* to registers %128 to %255 (only). See text.
%80 .. %95	Indirect access using *Seg1* to registers %128 to %255 (only). See text.
%64 .. %79	Indirect access using *Seg0* to registers %128 to %255 (only). See text.

Table 2.2a -- Integer register definitions

Register	Definition
%57 or msk11	Constant: `0xFFFFF800`. Writes are ignored
%56 or msk12	Constant: `0xFFFFF000`. Writes are ignored
%47 or riscintr	RISC interrupt register. Used for “polling” (not really an async interrupt on V2200). See text.
%46 or winclip	Same as *clip* register, except returns the last 4 clip flags in bits b5:b2. Read only
%41 or npfifo	Read word from *InputFIFO* but do not pop. No byte swap. Read only.
%40 or npswfifo	Read word from *InputFIFO* with byte swap (B3↔B0 and B2↔B1) but do not pop. Read only.
%40 or drawctl	Draw control register. See the *Parameterized Draw* and *Triangle Engine* sections. Write only.
%39 or timer	Free running 32-bit RISC cycle counter. Writes have a hazard of 3 instructions. Read/write.
%38 or clip	3D clip-code register. Flags from last 6 *clip* or last 3 *clipv* instructions in b7:b2, other bits = 0. Updated after *clip* or *clipv* instruction with a hazard of 1 instruction. See text. Read only.
%37 or flag	Flag register. See the *Flag Register* section. There is a hazard of 3 instructions from an instruction that writes *Flag* to an instruction that reads or uses it. Read/write.
%36 or scale	Most significant asserted bit location + 1 of input S2 register (e.g. if *seed* == 16, *Scale* = 5). *scale* has a hazard of 1 cycle after *seed S2*. Invalid (for RISC) after perspective correct *draw*. Read only.
%35 or seg	Segment register (b15:b0) is read/write. Has a 3 cycle hazard after written before it can be read or used as a register file pointer. See the *Register File Segment Register* section. Also, the normalized 13-bit index into the seed ROM can be read in b31:b19, with a hazard of 1 instruction after a *seed*.
%34 or seed	Output of the *seed* ROM (left aligned). Updated after *seed S2* instruction with a hazard of 2 instructions. Invalid (for RISC) after perspective correct *draw*. Read only.
%33 or fifo	Read/write word from/to *InputFIFO*/*OutputFIFO*. No byte swap. Read/write.
%32 or swfifo	Read word from *InputFIFO* with byte swap (B3↔B0 and B2↔B1). Read only.
%31 or f800	Constant: `0x0000F800`. Writes are ignored
%30 or f000	Constant: `0x0000F000`. Writes are ignored
%29 or e800	Constant: `0x0000E800`. Writes are ignored
%28 or e000	Constant: `0x0000E000`. Writes are ignored
%27 or d800	Constant: `0x0000D800`. Writes are ignored
%26 or d000	Constant: `0x0000D000`. Writes are ignored
%25 or c800	Constant: `0x0000C800`. Writes are ignored
%24 or c000	Constant: `0x0000C000`. Writes are ignored
%23 or b800	Constant: `0x0000B800`. Writes are ignored
%22 or b000	Constant: `0x0000B000`. Writes are ignored
%21 or bit17	Constant: `0x00020000`. Writes are ignored
%20 or bit23	Constant: `0x00800000`. Writes are ignored
%19 or bit25	Constant: `0x02000000`. Writes are ignored
%18 or fpfrac	Constant: `0x007FFFFF`. Writes are ignored
%17 or bit16	Constant: `0x00010000`. Writes are ignored
%16 or allfs	Constant: `0xFFFFFFFF`. Writes are ignored
%15 or if1_0	Constant: `0x00010000`. Writes are ignored
%14 or zer3	Constant: `0x00000000`. Writes are ignored
%13 or minus1	Constant: `0xFFFFFFFF`. Writes are ignored
%12 or zer2	Constant: `0x00000000`. Writes are ignored
%11 or b1b3msk	Constant: `0x00FF00FF`. Writes are ignored
%10 or b0b2msk	Constant: `0xFF00FF00`. Writes are ignored
%9 or h1msk	Constant: `0x0000FFFF`. Writes are ignored
%8 or h0msk	Constant: `0xFFFF0000`. Writes are ignored
%7 or b3msk	Constant: `0x000000FF`. Writes are ignored
%6 or b2msk	Constant: `0x0000FF00`. Writes are ignored
%5 or b1msk	Constant: `0x00FF0000`. Writes are ignored
%4 or b0msk	Constant: `0xFF000000`. Writes are ignored
%3 or msb	Constant: `0x80000000`. Writes are ignored
%2 or if0_5	Constant: `0x00008000`. Writes are ignored
%1 or zer1	Constant: `0x00000000`. Writes are ignored
%0 or zero	Constant: `0x00000000`. Writes are ignored

Table 2.2b -- Integer register definitions

2.3 ALU operations

Integer instructions can be scalar (1 operation) or vectors (suffix = v, 2 operations). Vector operand S1 can also be specified as scalar (same value for both operations, vector-scalar suffix = vs). Vector operands must be double-word aligned. Immediate operations use an 8-bit unsigned immediate value in place of S1. All operations are signed unless otherwise specified. There are no instruction exceptions, no status register, and ALU overflows wrap.

Tables 2.3a and 2.3b show ALU instructions.

scalar	Immed	v	vs	Encoding		Description	Dly
10	00	20	30	**add**	D,S2,S1	D = S2 + S1	0
11	01	21	31	**sub**	D,S2,S1	D = S2 – S1	0
12	02	22	32	**andn**	D,S2,S1	D = S2 AND NOT S1	0
13	03	23	33	**subf**	D,S2,S1	D = S1 – S2	0
14	04	24	34	**and**	D,S2,S1	D = S2 AND S1	0
15	05	25	35	**or**	D,S2,S1	D = S2 OR S1	0
16	06	26	36	**nor**	D,S2,S1	D = S2 NOR S1	0
17	07	27	37	**xor**	D,S2,S1	D = S2 XOR S1	0
18	08	28	38	**mul**	D,S2,S1	D = S2 * S1	1*
19	09	29	39	**mulsr8**	D,S2,S1	D = (S2 * S1) >> 8	1*
1A	0A	2A	3A	**mulsr16**	D,S2,S1	D = (S2 * S1) >> 16	1*
1B	0B	2B	3B	**mulsr24**	D,S2,S1	D = (S2 * S1) >> 24	1*
1C	0C	2C	3C	**mulsr32**	D,S2,S1	D = (S2 * S1) >> 32	1*
1D	0D	2D	3D	**mulsr31**	D,S2,S1	D = (S2 * S1) >> 31	1*
1E	0E	2E	3E	**clip**	S2,S1	*Clip* = (*Clip*<<1) & 0xfc | ((S2–S1≤0) ? 0 : 4) (signed compare)	hazard = 1
1F	0F	n/a	n/a	**f2i**	D,S2,S1	D = int(S2 * (1 << S1)). S2 is an IEEE 754 32-bit float. S1 is an integer (0 ≤ S1 ≤ 30). Result is rounded toward –∞ and overflows saturate to most negative or positive integer.	0

Table 2.3a - Vectorizable ALU opcode definitions

Scalar	Immed	Encoding		Description	Dly
n/a	40	**addif**	D,S2,S1	D = S2 + (S1 << 16)	0
n/a	41	**subif**	D,S2,S1	D = S2 – (S1 << 16)	0
52	n/a	**abs**	D,S1	D = abs(S1)	0
53	42	**seedsr**	D,S2,S1	D = *seed*(S2) >> (*scale*(S2) - S1), 0 ≤ *scale*(S2) - S1 ≤ 30. Updates scale, seed and seed index. Can not occur for >3 instructions after a *Fill Engine* instruction. See text.	2
n/a	43	**subfif**	D,S2,S1	D = (S1 << 16) – S2	0
54	44	**rotr**	D,S2,S1	D = S2 rotated right by S1 bits.	0‡
55	45	**sl**	D,S2,S1	D = S2 << (S1 & 0x1F) (logical)	0‡
56	46	**asr**	D,S2,S1	D = S2 >> (S1 & 0x1F) (arithmetic)	0‡
57	47	**sr**	D,S2,S1	D = S2 >> (S1 & 0x1F) (logical)	0‡

* All multiply instructions are signed and generate a 64-bit product. This product is right shifted by 0, 8, 16, 24, 31 or 32 bits. After the shift, the least-significant 32-bits are used as the result. All scalar multiply instructions take one cycle to launch and have a delay of one. All multiply vector instructions take two cycles to launch, the even register multiply has a delay of zero and the odd register multiply has a delay of one.

‡ If the result of a shift operation is used as a branch decision variable, that shift operation has an interlocked delay of one.

58	48	**slt**	D,S2,S1	D = ((S2 < S1) ? 1 **:** 0). (signed)	0
59	49	**sltu**	D,S2,S1	D = ((S2 < S1) ? 1 **:** 0). (unsigned)	0
5A	4A	**seq**	D,S2,S1	D = ((S2 == S1) ? 1 **:** 0)	0
n/a	4B	**addsl8**	D,S2,S1	D = S2 + (S1 << 8)	0
5C	4C	**min**	D,S2,S1	D = min(S2, S1)	0
5D	4D	**max**	D,S2,S1	D = max(S2, S1)	0
5E	n/a	**at**	D,S2,S1	D = (S2 + S1) & 0xFFFF0000	0
n/a	4E	**sprii**	reg,immed16	pixreg = immed16	n/a***
n/a	4F	**spri**	reg,S2	pixreg = S2	n/a***
5F	n/a	**spr**	(S1),S2	PixRegSpecifiedBy(S1) = S2	n/a***
C0	n/a	**getxy**	D,S1	Deven = S1 & 0xffff0000, Dodd = S1 << 16	0
C1	n/a	**getyx**	D,S1	Deven = S1 << 16, Dodd = S1 &0xffff0000	0
C2	n/a	**getra**	D,S1	Unpack unsigned bytes (red, alpha) to two 16.16 words D0 = S1 & 0xff0000, D1 = (S1 >> 8) & 0xff0000	0
C3	n/a	**getgb**	D,S1	Unpack unsigned bytes (green, blue) to two 16.16 words. D0 = (S1 << 8) & 0xff0000, D1 = (S1 << 16) & 0xff0000.	0
C4	n/a	**sort**	S3,S2,S1	Sort three *General Purpose* registers. The upper bits of the smallest value's index are put in *Seg0*, the middle value's index in *Seg1*, and largest value's index in *Seg2*. For ties, S1 is considered smallest, then S2. Also, if ⌈ max(S3,S2,S1) ⌉ – 1 < ⌈ min(S3,S2,S1) ⌉, four 1 bits are shifted into *clip* (and *winclip*) else four 0 bits are shifted in.	2 cycle. Hazard = 3 for *Seg* and = 2 for *clip*

Table 2.3b - Scalar ALU opcode definitions

2.4 Load and Store Instructions

Load and stores are "register + offset" based and are shown in table 2.4. The unsigned 8-bit offset is multiplied by the data-size and then added to the contents of S1 to form the memory (byte) address. Note, memory loads have an interlocked delay of 1 instruction (*pre*, *li*, and *lui* have a delay of 0). The RISC processor is "big-endian."
There is one 8-byte data block in the "*line-buffer-cache*" (production will have one 16 byte data block) that is shared between RISC load data and PCI read data. Note, the *pre* instruction, the *lvra* instruction and PCI reads to odd words launch an 8-byte data prefetch. The *line-buffer-cache* is invalidated on store hits.

*** There is a hazard of two instructions from an instruction that uses a *Triangle Engine* shadowed register as a destination and a *sprii*, *spri* or *spr* instruction.

OP	Encoding		Description
70	**lb**	D, offset8(S1)	D = unsigned_Byte(S1 + offset8)
71	**lh**	D, offset8(S1)	D = unsigned_HalfWord(S1 + offset8 * 2). Address must be half-word-aligned
72	**lw**	D, offset8(S1)	D = unsigned_Word(S1 + offset8 * 4). Address must be word-aligned.
73	**pre**	offset8(S1)	Launch a prefetch of memory location at (S1 + offset8 * 8).
74	**lv**	D, offset8(S1)	Deven = Word(S1 + offset8 * 8), Dodd = Word(S1 + offset8 * 8 + 4). The D index must be even. Address must be double-word-aligned.
75	**lvra**	D, offset8(S1)	Deven = Word(S1 + offset8 * 8), Dodd = Word(S1 + offset8 * 8 + 4). The D index must be even. Address must be double-word-aligned. Prefetch (S1 + offset8 * 8 + 8).
76	**li**	D, #immed16	D = immed16. (load 16-bit unsigned immediate)
77	**lui**	D, #immed16	D = immed16 << 16. (load-upper immediate (low 16-bits = 0))
78	**sb**	offset8(S1), S2	Byte(S1 + offset8) = S2
79	**sh**	offset8(S1), S2	HalfWord(S1 + offset8 * 2) = S2. Address must be half-word-aligned
7A	**sw**	offset8(S1), S2	Word(S1 + offset8 * 4) = S2. Address must be word-aligned
7C	**sv**	offset8(S1), S2	Word(S1 + offset8 * 8) = S2even, Word(S1 + offset8 * 8 + 4) = S2odd. S2 index must be even. Address must be double-word-aligned.
7D	**sy**	offset8(S1), S2	Byte(S1 + offset8 * 8) = S2[31:24], Byte(S1 + offset8 * 8 + 2) = S2[23:16], Byte(S1 + offset8 * 8 + 4) = S2[15:8], Byte(S1 + offset8 * 8 + 6) = S2[7:0]. Address must be double-word-aligned. Other bytes unchanged.
7E	**scr**	offset8(S1), S2	Byte(S1 + offset8 * 8 + 1) = S2[31:24], Byte(S1 + offset8 * 8 + 5) = S2[23:16], Byte(S1 + offset8 * 8 + 9) = S2[15:8], Byte(S1 + offset8 * 8 + 13) = S2[7:0]. Address must be quad-word-aligned. Other bytes unchanged. Two cycles in *DCache*.
7F	**scb**	offset8(S1), S2	Byte(S1 + offset8 * 8 + 3) = S2[31:24], Byte(S1 + offset8 * 8 + 7) = S2[23:16], Byte(S1 + offset8 * 8 + 11) = S2[15:8], Byte(S1 + offset8 * 8 + 15) = S2[7:0]. Address must be quad-word-aligned. Other bytes unchanged. Two cycles in *DCache*.

Table 2.4 - Load/Store opcode definitions

2.5 Branch and Jump Instructions

The processor has non-annulling delayed-branches (i.e. branch/jump delay-slot instructions are always executed. Note, *halt* and *getpc* do not have a delay slot). The delay-slot of a jump or branch operation cannot contain a jump or a branch, but can contain a *halt* or a *getpc*. For branches, *offset16* is multiplied by 4 and sign-extended before being used as the branch offset. The branch offset is relative to the branch delay-slot. For *jmp* and *jmpl*, *addr* * 4 is used as the absolute jump target address. The most-significant bit of an absolute jump address specifies ROM memory if set and normal memory if cleared. Note, only *blz* and *bgez* are available for indexes ≤63. Table 2.5 shows branch and jump instructions.

Opcode	Encoding		Description
60	**bez**	S1, offset16	Branch to dlyslotPC+offset16*4 if S1 == 0
61	**bnez**	S1, offset16	Branch to dlyslotPC+offset16*4 if S1 != 0
62	**bgez**	S1, offset16	Branch to dlyslotPC+offset16*4 if S1 ≥ 0
63	**blz**	S1, offset16	Branch to dlyslotPC+offset16*4 if S1 < 0
64	**bgz**	S1, offset16	Branch to dlyslotPC+offset16*4 if S1 > 0
65	**blez**	S1, offset16	Branch to dlyslotPC+offset16*4 if S1 ≤ 0
68	**rjmp**	addr	Relative jump to dlyslotPC+addr*4
6A	**rjmpl**	addr	Relative jump to dlyslotPC+addr*4, and link (using *ra* reg)
6B	**getpc**	D	Jump-never and link (using D as link register). Does not have a delay slot.
6C	**jmp**	addr	Jump to absolute *addr* << 2 (*addr* is 24-bits).
6D	**halt**	cause	Halt RISC. *cause* is optional. Last executed instruction. Does not have a delay slot.
6E	**jmpl**	addr	Jump to absolute *addr* << 2 (*addr* is 24-bits), and link (using *ra* reg).
6F	**jmprl**	D, (S1)	Jump to address in S1, and link (using D as link register).

Table 2.5 - Branch and jump opcode definitions

2.6 Step Instructions

In V1000 compatibility mode, step instructions are used to update Pixel Engine shadow registers that are not going to be explicitly updated by the next *draw* instruction. Step instructions are illegal in V2000 mode (*drawctl[3:0]* = 0xB). They can be scalar (*step*) or vector (*stepv*). A scalar *step* instruction updates a single *Pixel Engine* register (using S2) and also does a normal scalar *add* of its operands, in the RISC processor. The vector *stepv* instruction updates two *Pixel Engine* registers (using respective S2) and also does a normal vector *addv* using its operand vector pairs. DrawCtl[3:0] cannot be equal to 11 when issuing *step* or *stepv* instructions. When in V2000 mode, there is a hazard of two instructions from an instruction that uses a *Triangle Engine* shadowed register as a destination to a *step* or a *stepv* instruction. There is a hazard of 1 instruction form a *step cnt* instruction to any draw instruction. No other *step* instruction has a hazard. Table 2.6 shows the *step* instructions.

Opcode	Encoding		Description
D0	**step** **step**	x,D,S2,S1 y,D,S2,S1	For *step x*, the sign of S1 is used to update *XDir*. S2 and S1 index must be even for *step x*, which updates *XShadow*. S2 index must be odd for *step y*. Format is 16I.16F.
D1	**step**	q,D,S2,S1	For *step q*, S2 is used for a seed operation. There is no register alignment restriction. Format for q is 0I.24F. Must be >3 instructions after the previous *Fill Engine* instruction.
D2	**step** **step**	r,D,S2,S1 v,D,S2,S1	For *step v*, the sign of S1 is used to update *VDir*. S2 index must be even for *step r*. S2 and S1 index must be odd for *step v*. Format is 16I.16F.
D3	**step** **step**	i,D,S2,S1 u,D,S2,S1	For *step u*, the sign of S1 is used to update *UDir*. S2 index must be even for *step i* (updates *r* = *g* = *b* = *i*) and odd for *step u*. Format is 16I.16F.
D4	**step**	f,D,S2,S1	S2 index must be odd for *step f*. Updates fog factor. Format is 16I.16F
D5	**step** **step**	g,D,S2,S1 b,D,S2,S1	S2 index must be even for *step g* and odd for *step b*. Format is 16I.16F.
D6	**step** **step**	z,D,S2,S1 a,D,S2,S1	S2 index must be even for *step z* and odd for *step a*. Format is 21I.11F for *z* and 16I.16F for *a*.
D7	**step**	cnt,D,S2,S1	S2 index must be even for *step cnt*. Updates *Cnt* register. Format is unsigned 16I.16F
D8	**stepv**	xy,D,S2,S1	The sign of even index S1 is used to update *XDir*. Even index S2 updates *XShadow*, odd index S2 updates *y*. Format is 16I.16F.
D9	**stepv**	qu,D,S2,S1	The sign of odd index S1 is updates *UDir*. The even index S2 is used for a seed operation (*step q*), odd index S2 updates *u*. Format for q is 0I.24F. Format for *Cnt* is 16I.16F.
DA	**stepv**	rv,D,S2,S1	The sign of odd index S1 is used to update *VDir*. Even index S2 updates *r*, odd index S2 updates *v*. Format is 16I.16F.
DB	**stepv**	iv,D,S2,S1	The sign of odd index S1 is used to update *VDir*. Even index S2 updates *i* (updates *r*, *g* and *b*), odd index S2 updates *v*. Format is 16I.16F.
DC	**stepv**	zv,D,S2,S1	The sign of odd index S1 is used to update *VDir*. Even index S2 updates *z*, odd index S2 updates *v*. Format is 16I.16F for *v* and 21I.11F for *z*.
DD	**stepv**	gb,D,S2,S1	Even index S2 updates *g*, odd index S2 updates *b*. Format is 16I.16F.
DE	**stepv**	za,D,S2,S1	Even index S2 updates *z*, odd index S2 updates *a*. Format is 21I.11F for *z* and 16I.16F for *a*.
DF	**stepv**	cnt,D,S2,S1	Same as *step Cnt,D,S2,S1* except has extra add (odd index). Even index S2 updates *Cnt*.

Table 2.6 - *step* instruction definitions

Note, *Cnt* is the length count used by all non *Triangle Engine* drawing instructions. Note, *step Cnt* and *stepv Cnt* have a non-interlocked delay of 1 before and parallel or parameterized draw instructions. Note, *Xshadow* is the *X* value used by non *Triangle Engine* drawing instructions that do not have an explicit *X* coordinate register file parameter. Note, *step x* and *stepv xy* have a hazard of 1 before parallel draws, but not parameterized draws. Note, for *step* operations that update *Red*, *Green*, *Blue*, or *Alpha* the respective field of the *FGColor* register are updated, but other *FGColor* fields remain the same. Note, *Red*, *Green*, *Blue*, *Intensity*, *Fog* and *Alpha* values are saturated to the range of 0 to 255 as they are passed to the Pixel Engine.

2.7 Draw Instructions

There are two types of draw instructions, *Parallel Draw* and *Parameterized Draw*. Either a single draw (e.g. *draw1*) or multiple draw (e.g. *rdraw1*) can be specified in a single instruction. All draw instructions use *Pat*, if it is enabled, and conditionally draw based on the *Cnt* register. The *Cnt* register is a 16-bit down counter that is loaded with b31:b16 of *S2* during *step Cnt* or *stepv Cnt* instructions. *Cnt* is unsigned for parallel draws and signed for parameterized draw operations. Note, there is a 3 instruction hazard after writing the *drawctl* register before any draw instructions. Note, *draw* and *rdraw* cannot occur in V2000 mode (*drawctl[3:0]* = 0xB).

Number formats are the same as for *step* instructions (e.g. X is 16.16, Z is 21.11 and Q is 0.24). See the following sections for more information on each type of draw instruction.

Draw operation operands are implicitly accessed as *base* + *offset*, where *base* is the properly aligned *D*, *S2* and *S1*. Draw instructions using ≤ 2 register file locations must have operands that are double-word aligned. Those with 3 or 4 must be quad-word aligned. Those with 5, 6, 7 or 8 must be eight-word aligned. Those with > 8 must be sixteen-word aligned. Offsets change based on the specified *draw* instruction. Delta values (*S1*) have the same offset as their respective parameters (*S2*).

Explicit drawing parameters are written to the *Pixel Engine* as they are read into *S2* for the *X1* stage of the RISC pipeline. Registers required by a drawing operation, but not explicitly updated by it (that have been changed by the RISC), must be set using a *step* instruction before issuing the related drawing instruction. For example, for many draw operations, *Y* is the current value of *Y* held in the *Pixel Engine* (as set by the previous *step y* or *stepv xy* instruction). Note, set *SwapUV* before setting *U* or *V*. After completion of a draw instruction, destination registers used drawing instructions are as described below.

2.8 Parallel Draw Instructions

For *Parallel Draw* instructions, the *Span Engine* draws 4/sizeof(pixel) pixels per cycle (for non left/right edge destination words), unless the Pixel Engine pipeline is full due to saturating memory bandwidth. Table 2.8 describes *Parallel Draw* instructions. This includes reading any required source data.

X, *XDirection* and *Y* are normally preset using a *stepv xy* instruction before each *Parallel Draw* (*XDirection* is the sign of *S1)*. Parallel draw instructions can draw left to right (*XDirection* = 0, and if a source is being used *dU* = 0) or right to left (*XDirection* = 1, and if a source is being used *dU* = 0xfff80000). There is a hazard of one cycle after setting *Cnt* or *XShadow* before a *Parallel Draw* instruction. The *SwapUV* bit of the *SrcMode* register must be cleared (before setting *V*). The *Umask*, *Vmask* must be set to all 1's and *UClamp*, *VClamp* must be disabled. Patterning is only supported for parallel drawing when drawing from left to right with a pattern-width ≥ 8/sizeof(pixel). *Cnt* is treated as an unsigned 16-bit number. If *Cnt* == 0, no draw occurs and *Cnt* remains = 0. For the *drawp* instruction the destination registers are updated as with an *addv* instruction, if and only if *Cnt* != 0. For the *rdrawp* instruction the destination registers not modified.

The *rdrawpxy* instruction is intended for use in drawing rectangular objects. Its advantage is that a rectangular object can be drawn with an unrolled loop of back-to-back *rdrawpxy* instructions. At the completion of a *drawpxy* or *rdrawpxy*, the *Cnt* and *XShadow* registers are reset to their initial values and the *Register-File* destination pair are updated as with an *addv* exactly once. Note, *Y* is the *Y* scan line value and is used by the *Span Engine* (*H* is not used by the *Span Engine*

Opcode	Encoding		Description	Parameter Offset from Base							
				0	1	2	3	4	5	6	7
E0	**drawp**	D,S2,S1	Single parallel draw. Update *XShadow*, *U*.	--	U						
E1	**rdrawp**	D,S2,S1	Repeated parallel draw. Update *XShadow*, *U*.	--	U						
E4	**drawpxy**	D,S2,S1	Single resetting parallel draw. See text.	H	Y						
E5	**rdrawpxy**	D,S2,S1	Repeated resetting parallel draw. See text.	H	Y						

Table 2.8 - Parallel draw instruction definitions

2.9 Parameterized Draw Instructions

Parameterized Draw instructions (e.g. *rdraw1*) allow most combinations of parameters to be iterated during drawing. Table 2.9a describes these instructions. The parameters to iterate are specified by the *DrawCtl* register as shown in tables 2.9b and 2.9c. Note, DrawCtl[3:0] cannot be equal to 11 when issuing *draw* or *rdraw* instructions.

Cnt is treated as a signed 16-bit number. If *Cnt* ≤ 0, no draw occurs and *Cnt* is unchanged. There is a hazard of one cycle after setting *Cnt* before any draw instruction. There is no hazard , or delay, after setting the *XShadow* register before any *Parameterized Draw* instructions. There is a hazard of 3 cycles after setting *DrawCtl* before a *Parameterized Draw* can be issued. For *draw* operations, the specified parameter destination registers are updated as with an addv instructions, if and only if *Cnt* > 0. *rdraw* operations do not modify destination registers. If a *Parameterized Draw* instruction is reached while the *Fill Engine* is busy, the RISC will stall until it is completed. The RISC is stalled until *rdraw*'s complete. A *draw* instruction takes 3 cycles and an *rdraw* takes (2 + 2 * ParamPairs + 1 * pixels) cycles (e.g. draw3 takes 2 + 2 * 3 = 8 cycles to launch). Note, for R, G, and B, either none, or all, of them must be iterated (e.g. *rdraw* with *QUIV*, *XRGB*, but not *XR*).

All *Parameterized Draw* instructions that include *Q* do two divides per pixel (*UQ*/*Q* and *VQ*/*Q*) for proper perspective correction of texture sources. Perspective correct *Parameterized Draw* instructions must be ≥ 2 pairs in length (e.g. *rdraw1* with *pair1* = 5 is illegal). Before a perspective correct *Parameterized Draw* instruction, a *step q,D,S2,S1* instruction must be issued. As this implies, *Q* is one pixel ahead of other parameters.

Opcode	Encoding		Description	Parameter Offset from Base									
				0	1	2	3	4	5	6	7	8	9
F0	**draw1**	D,S2,S1	1 pair	E1	O1								
F1	**rdraw1**	D,S2,S1	1 pair	E1	O1								
F2	**draw2**	D,S2,S1	2 pairs	E1	O1	E2	O2						
F3	**rdraw2**	D,S2,S1	2 pairs	E1	O1	E2	O2						
F8	**draw3**	D,S2,S1	3 pairs	E1	O1	E2	O2	E3	O3				
F9	**rdraw3**	D,S2,S1	3 pairs	E1	O1	E2	O2	E3	O3				
FA	**draw4**	D,S2,S1	4 pairs	E1	O1	E2	O2	E3	O3	E4	O4		
FB	**rdraw4**	D,S2,S1	4 pairs	E1	O1	E2	O2	E3	O3	E4	O4		
FC	**draw5**	D,S2,S1	5 pairs	E1	O1	E2	O2	E3	O3	E4	O4	G	B
FD	**rdraw5**	D,S2,S1	5 pairs	E1	O1	E2	O2	E3	O3	E4	O4	G	B

Table 2.9a - *Parameterized draw* instruction definition

3	*4*	*5*	*4*
pair4	pair3	pair2	pair1
b15:b13	*b12:b9*	*b8:b4*	*b3:b0*

Table 2.9b -- *DrawCtl* register field definition

pair1	XShadow	E1	O1	pair2	E2	O2	pair3	E3	O3	pair4	E4	O4
0	N/A	X	*	8	X	V	0	G	B	0	G	B
1	X	Q	*	9	Y	V	1	Z	A	1	Z	A
2	N/A	X	Y	10	R	V	2	Z	*	2	Z	*
3	X	Q	Y	11	*	V	3	*	F	3	*	F
4	N/A	X	U	12	I	V	4	Z	F	4	Z	F
5	X	Q	U	13	Z	V	5	N/A	N/A	5	N/A	N/A
6	N/A	X	R	14	A	V	6	R	F	6	R	F
7	X	Q	R	15	Z	U	7	R	A	7	R	A
8	N/A	X	A	16	G	B	8	R	*			
9	X	Q	A	17	Z	A	9	Z	R			
10	N/A	Z	X	18	Z	*	10	*	A			
11	N/A	N/A	N/A	19	*	F	11	F	A			
12	N/A	X	F	20	Z	F	12	I	*			
13	X	Q	F	21	F	V	13	Z	I			
14	N/A	Y	U	22	R	F	14	I	A			
15	Y	Q	U	23	R	A	15	I	F			
				24	R	*						
				25	Z	R						
				26	*	A						
				27	F	A						
				28	I	*						
				29	Z	I						
				30	I	A						
				31	I	F						

Table 2.9c -- *DrawCtl* parameter pairs

Note, * indicates the register file entry is modified, but the field is not used by the *Pixel Engine*.

2.10 Triangle Drawing Instructions

The *Triangle Engine* (*tri* instruction) draws a triangle using a DDA edge walk with sub-pixel correction for Q, U, V and Z. It renders triangles at the rate = (4 + 1 * spans + 1 * pixels) cycles, up to memory bandwidth limits. Note, the chip must be in V2000 mode (*drawctl[3:0]* = 0xB), before setting up or using the *Triangle Engine*. Note, there are no hazards after writing the *drawctl* register to put the chip in V2000 mode. Important features of the Triangle Engine include:

- Iterates X, Z, U, V, Q, R, G, B, A, F, Rs, Gs, Bs along span at one clock per pixel.
- Walks the long vertical edge.
- Subpixel corrects Z, U, V, Q along the edge and along the span.
- Shadows register file locations, if enabled, to automatically load parameters.
- Has an iterate enable bit for each class of parameters.
- Monochrome (gray scale) option reduces setup time.
- Has swapxy option for X Major edge stepping.

Any triangle shape can be rendered by the *Triangle Engine*, up to the limitations of the precision and range of the parameters. Trapezoids with horizontal bottom edges can be drawn as truncated triangles. The bottom *Y* value for the triangle does not have to be located at the intersection of the long edge and the bottom edge. In fact, the long edge and bottom edge don't have to converge. Any convex triangle with a horizontal bottom edge can be rendered top to bottom. The long edge and bottom edge must not intersect before *YBot* or rendering will be incorrect. Refer to the V2200 *step* instructions for the parameter formats.

The *tri* instruction launches the triangle operation from the RISC into the *Triangle Engine*. Since all required parameters are automatically shadowed, the actual launch takes only one cycle. Once launched, the RISC is free to continue normal operations until it tries to write one of the shadowed register locations (*DrawCtl*, *a0 - a31*, *q0 - q15*). If the *Triangle Engine* is busy when one of these registers is written, the RISC stalls until the *Triangle Engine* becomes idle. Note, if a *Triangle Engine* instruction is reached while the *Fill Engine* is busy, the RISC will stall until it is completed. There is a 3 instruction hazard after a *tri* instruction before a *Fill Engine* instruction. A *tri* instruction takes 4 + 1 * scanlines + 1 * pixel cycles, not including memory stalls. Table 2.10a defines the triangle instruction format.

Opcode	Encoding		Description
FE	tri	D,S2,S1	Draw a triangle. A vector add is performed before the triangle is actually launched. The following pairs could be calculated and loaded as part of the tri instruction: xy, zq, uv, ra, gb, r2f, g2b2.

Table 2.10a - *tri* instruction definition

The low 4 bits of *DrawCtl* enable *Triangle Engine* register shadowing when set to 0xB. When enabled, writes to the respective destination register also write to a copy of that register that is physically located in the *Triangle Engine*, independent of whether that parameter is enabled to be iterated. Note, parameters U, V, Z, A, R, G, B and F, if not being iterated, use the current state of the Pixel Engine. So, in this case, they can be the values from the last *spr* or *step* instruction or they can be written to the shadow registers as part of triangle setup. Other parameters must be written to their respective shadowed registers during triangle setup. Note, the *mono* bit of the *DrawCtl* register must be initialized before writing any of the color or color-gradient registers. If the *mono* bit is set, writes to any of the color registers is replicated into all three respective color registers (e.g. writing dR/dx also updates dG/dx and dB/dx). There is no hazard, or delay, between writing *DrawCtl* and updating a register shadowed by the *Triangle Engine*. Table 2.10b specifies the location of *DrawCtl* bits and table 2.10c defines the individual bits.

14	13	12	11	10	9	8	7	6	5	4	3:0
mono	n/a	swapxy	F	S	A	D	V	U	Q	Z	0xB

Table 2.10b - *DrawCtl* register bit locations for *tri* instruction

Bits	Definition	Bits	Definition
0xB	Enable register shadowing. Other values disable it.	A	Alpha iterate Enable
Z	Z iterate enable	S	Specular (Rs, Gs, Bs) iterate enable
Q	Q iterate enable. If enabled use perspective correct texturing, else use not perspective correct	F	Fog iterate enable
U	U texture coordinate iterate enable	swapxy	Swap X and Y parameters. Step in X major edge
V	V texture coordinate iterate enable	mono	Monochrome lighting (last update of R, G or B replicated to all three). Write before setting colors.
D	Diffuse (R, G, B) iterate enable		

Table 2.10c - *DrawCtl* register bit definition for *tri* instruction

State save and restore requirements impose restrictions on the use of the *Triangle Engine*. The *Triangle Engine* uses a fixed set of register file locations for its parameters. These registers are automatically shadowed, when shadowing is enabled. For compatibility with old microcode, register shadowing must be enabled at the beginning of a triangle subroutine that uses the *Triangle Engine* and disabled before exiting that subroutine. By only enabling shadowing when the *Triangle Engine* is in use, V1000 microcode runs without modification. Microcode must leave *Triangle Engine* register shadowing enabled while setting up and using the *Triangle Engine*. The appropriate *Triangle Engine* and *Pixel Engine* state registers are updated by writes to shadowed registers. All start values must be loaded after shadowing is turned on and before the first *tri* instruction. The start values are: *XTop*, *YTop*, *Z*, *Q*, *R*, *A*, *G*, *B*, *R2*, *G2*, *B2*, *U*, *V*. Note, an *spr FGColor* can be used to initialize *A*, *R*, *G* and *B*. Also, *dX*, *dXLong/dy*, *dXTop/dy*, *dXBot/dy* must be loaded before the first tri instruction. All iterated parameters must be initialized before each *tri* instruction, to avoid generating hidden state in the *Triangle Engine*). If iteration is disabled by the *DrawCtl* register for a parameter or both its gradients are =0, that parameter is not iterated (note, *XTop*, *YTop* and *XMid* are always iterated). Gradients are only needed for parameters that are enabled for iteration by the *DrawCtl* register. Note, dX must be written =0 if the long edge is on the left and =0x80000000 if it is on the right. Table 2.10d defines the *Triangle Engine* parameter registers.

Reg	Index	Parameter	Reg	Index	Parameter	Reg	Index	Parameter
a0	%128	XTop	a16	%144	XMid	q0	%160	dX
a1	%129	YTop	a17	%145	YMid	q1	%161	YBot
a2	%130	Z	a18	%146	dZ/dx	q2	%162	dZ/dy
a3	%131	Q	a19	%147	dQ/dx	q3	%163	dQ/dy
a4	%132	R	a20	%148	dR/dx	q4	%164	dR/dy
a5	%133	A	a21	%149	dA/dx	q5	%165	dA/dy
a6	%134	G	a22	%150	dG/dx	q6	%166	dG/dy
a7	%135	B	a23	%151	dB/dx	q7	%167	dB/dy
a8	%136	R2	a24	%152	dR2/dx	q8	%168	dR2/dy
a9	%137	F	a25	%153	dF/dx	q9	%169	dF/dy
a10	%138	G2	a26	%154	dG2/dx	q10	%170	dG2/dy
a11	%139	B2	a27	%155	dB2/dx	q11	%171	dB2/dy
a12	%140	U	a28	%156	dU/dx	q12	%172	dU/dy
a13	%141	V	a29	%157	dV/dx	q13	%173	dV/dy
a14	%142	dXLong/dy	a30	%158	dXTop/dy	q14	%174	dXBot/dy
a15	%143	n/a	a31	%159	n/a	q15	%175	n/a

Table 2.10d - *tri* parameter definition

Note, the dX register (q0) has other functions as well as being used to set the X span drawing direction (b31). Table 2.10e defines the *dX* parameter register.

Bit	Description
31	dX span rendering direction. 0 = increasing, 1 = decreasing. This bit not used when drawing lines.
30:6	Reserved. Write as 0. Reads are unspecified.
5	swapxy. 0 = Normal. Swap X and Y (e.g. triangle spans are rendered in the Y direction). This bit is OR'd with the swapxy bit of the drawctl register.
4	Reserved. Write as 0. Reads are unspecified.
3:1	Primitive Type. 0 = triangle, 1 = line, 2 = span, 3 = dot, 5 = aaedge.
0	Edge walk directions. 0 = increasing, 1 = decreasing. This bit used for both triangles edges and to set the direction for lines.

Table 2.10e - dX register parameter definition

The primitive type determines which parameters are required. Table 2.10f defines which parameters are used for each Primitive Type.

Parameter	triangle	line	span	dot	aaedge
Edge gradient	used	used	not used	not used	used
Span gradient	used	not used	used	not used	not used
Y subpixel correction	used	used	not used	not used	used
X subpixel correction	used	not used	used	not used	not used
dX	used	not used	used	not used	used

Table 2.10f - Parameter usage by *Primitive Type*

To setup a triangle, microcode starts by sorting the vertex data in Y to find the top, bottom and middle vertex. The triangle is rendered by walking the longest edge in the Y dimension. If the longest edge in Y is on the left, the parameters are iterated from left to right along the span, otherwise they are iterated from right to left.

In preparation of walking the triangle edge, the gradient for each parameter must be found for a scanline step down the long edge of the triangle and for a horizontal step along the scanline. The gradient for each parameter "p" is calculated for a unit step in Y along the longest edge (E1):

E1_dpdy = (p2 - p0) / (y2 - y0)

A horizontal line through the middle vertex (P1) can be shown to be the longest span. The Y coordinate of the middle vertex is entered into the line equation for each parameter along the longest edge to find the parameter value at the intercept:

x3 = x0 + E1_dxdy * (y1 - y0)

The gradient for each parameter "p" along the scanline is computed using the endpoints of the longest span:

dpdx = $\Delta p/\Delta x$ = (p1 - p3) / |(x1 - x3)|

If the top vertex does not lie exactly on a pixel center, then a fractional step along the longest edge to the first scanline is necessary. A Y subpixel correction factor (yspc) is applied to each parameter delta for the longest edge:

yspc = (Y_frac) ? ~Y_frac + ε : 0
E1_psi = yspc * dpdy + P0

The triangle engine does the Y subpixel correction for X, X2, Z, U, V and Q. The remaining parameters can be Y subpixel corrected in the microcode if desired.

To begin drawing a scanline, the *Triangle Engine* steps from the edge to the first pixel center on the span. The Z,U,V,Q parameters are subpixel corrected for the distance from the edge to the first pixel center. If the long edge is on the left, it steps right by 1 - xfrac, unless the edge falls exactly on a pixel (xspc = 0):

LEL: xspc = (E1_xsi_frac) ? ~E1_xsi_frac + ε : 0

If the long edge is on the right, the *Triangle Engine* steps left by xfrac. If the edge falls on a pixel center, it steps left by exactly one pixel:

LER: xspc = (E1_xsi_frac) ? E1_xsi_frac : 1.0

The X subpixel correction factor is applied to parameter "p" (p = [Z, U, V, Q]) as follows:

span_p = E1_psi + xspc * dpdx

The *DrawCtl* register has individual enable bits for each parameter class. All enabled parameters are incremented once per clock along the span. Parameters that are not enabled for iterating retain their constant values. Pixels whose X coordinates are >= the left edge and < the right edge are drawn. The equation for any parameter "p" is:

Span_p = span_p + dpdx

After the span is complete, the iterators, that are enabled, step down the long edge:

Pedge = Pedge + E1_dPdy
Y++

The Triangle Engine works in conjunction with the Pixel Engine hardware. The current triangle must finish before the Pixel Engine state can be modified by the RISC. If the current triangle covers the full screen it will take around 6 mS to render (at 640x480). The Triangle idle condition is incorporated into the Pixel Engine Idle bit, so the V1000 context switch procedure can be used. Direct accesses to the frame buffer bypass the rendering engines to guarantee low latency.

Setting the swapxy bit allows trapezoids with a vertical edge The swapxy bit applies to all primitives rendered by the triangle engine. Setting it for a trapezoid causes the triangle engine to step the long edge in X from left to right. The interior is rendered with top to bottom or bottom to top spans for long edge top and long edge bottom cases respectively.

The triangle engine is split between the RISC and Pixel Engine. The perspective correction of U and V texture coordinates use the reciprocal unit and multiplier in the *Triangle Engine*. The color, alpha, fog and Z parameters are generated just before they are needed in the pixel engine.

2.11 Memory Fill Engine Instructions

The V2200 *Fill Engine* accelerates area fills. Both SGRAM and SDRAM memory are supported. When SGRAM memory is used as local memory, block writes are used for maximum performance. Each fill instruction writes one scan line of data to the frame buffer. *Fill Engine* instructions are launched from the RISC, but execute in the *Fill Engine*. After launching a fill instruction, the RISC can continue operation until a shared resource is needed, as described below. The *Fill Engine* is separate from the *Pixel Engine* and *Triangle Engine*. However, if the *Triangle Engine* or *Pixel Engine* are busy when a *Fill Engine* instruction is reached, the RISC will stall until they are both idle.

There are nine fill instructions: *fill*, *mbrush*, *mblt*, *mbltf*, *mbltfo*, *cbrush8*, *cbrush16*, *cbrush32* and *cbltf*. The *Fill Engine* supports 8, 16 and 32 bits per pixel. Several parameters are needed by the fill instructions: pixel *format*, *writemask*, *offset*, *color*, and fill data. These must be preloaded by microcode into general purpose registers, as appropriate, before the fill instruction is executed. The starting byte address of the memory write and the write length, in pixels, are specified by the *S2* operand vector register pair. The starting address is in the even register and the length is in the odd register. The starting address must be aligned to the pixel size. In addition to invoking the *Fill Engine*, fill instructions compute the starting address for the next fill instruction by adding the starting address to address delta value, which is in the scalar *S1* register operand. The result is written to the destination register. Typically, *D* and *S2* are the same register index. The *FIFO* register can not be used a either a source or destination for *Fill Engine* instructions.

Solid color fills are done by *fill*. Solid fill is the simplest of the fill instructions, needing only the pixel *format*, *writemask* and *color*. Typically, rectangles use an unrolled loop so scanlines can be launched at the rate of one cycle per scanline.

The *mbrush* instruction is a transparent monochrome brush expansion using a 32-bit monochrome brush. The brush is expanded to the pixel depth and repeated as necessary for the width of the fill. A brush bit that is a “1” specifies a foreground pixel. The *color* value is written to these pixels. Background pixels are not written. If a brush is less than 32 bits (pixels) wide, it must be replicated to fill 32 bits. Bit 31 of the brush is the left pixel. The *offset* is used to align the brush, as specified below.

The *mblt* instruction is a transparent monochrome expansion BitBlt. A source bit that is a “1” specifies a foreground pixel. The *color* value is written to these pixels. Background pixels are not written. Bit 31 of the first source word is the left source pixel. The *mbltf* instruction is the same as an *mblt*, except it uses the *InputFIFO* as the source. The *InputFIFO* source is read as needed by the Fill Engine. The *mbltfo* instruction is the same as an *mbltf*, except it is an opaque monochrome expansion BitBlt. For *mbltfo*, background pixels are written using the *bgcolor* value. For all three types of monochrome BitBlt the *offset* is used to align the source, as specified below.

The *cbrush* instructions perform opaque color brush writes. Color brushes are 8, 16, or 32 pixels wide, for *cbrush8*, *cbrush16*, and *cbrush32*, respectively. Note, *cbrush8* can only be used with 8 and 16 bpp, and *cbrush32* can only be used with 8 bpp. The color brush is repeated as necessary for the width of the fill. The colors of the brush are preloaded into some or all of registers *a8* through *a15*, depending on type. The left pixel is in the most significant byte or bytes.

Color BitBlts are done using the *cbltf* instruction. For *cbltf*, BitBlt data always comes from the input FIFO (if a memory to memory BitBlt is needed, the *Pixel Engine* should be used). BitBlt data is the same size as destination pixels. The first pixel is in the most-significant portion of the first input word. The *InputFIFO* is read as needed by the Fill Engine. If bit 7 of the *format* is set, the *cbltf* becomes chroma-key BitBlt. All pixels that match the *key* (a2) are scissored (memory is not modified). Note, *key* must be replicated to fill all 32 bits.

All Fill Engine instructions reference the *writemask* that enables the write on a “per-bit” basis in SGRAM systems using the SGRAM write mask feature. In SDRAM systems the *writemask* register is redefined, since SDRAM does not have the “per-bit” write mask feature. For SDRAM, the least significant bit of each byte of the writemask is used as the byte enable of the respective byte (writemask[0] enables byte 3 writes, etc.). Other bits of the writemask are ignored in SDRAM systems.

For 8 and 16 bit/pixel formats, the *color* and *writemask* must be replicated in their respective registers to fill all 32 bits. For SGRAM, monochrome fills copy the *color* and *writemask* to the memory’s color and mask registers, respectively, before starting the fill.

The alignment of the first pixel data is specified by the offset register. The offset indicates the number of pixels that it is shifted left. For example, if the second piece of fill data is to be written at the starting address, the offset register would contain a “1”, and the first datum would be thrown away. A value of "0" in the offset register causes the first datum to be written at the starting address.

If a color brush repeats every 8 bytes, hardware uses a single pass, even if at 16 or 32 bpp. If a color brush is >8 bytes wide, the *Fill Engine* makes multiple passes.

Table 2.11a describes *Fill Engine* instructions. Table 2.11b and 2.11c describe *Fill Engine* instruction parameters. Registers shadowed by the *Fill Engine* have a hazard after being written, before a *Fill Engine* instruction that uses them can be issued, as specified by these tables. There is also a three instruction hazard after a *Fill Engine* instruction before an instruction that generates a seed (e.g. *seedsr* or *step q*).

For the *Fill Engine*, programming considerations for maximum performance include:

- Loading the *color* register.
- Loading the *writemask* register.
- Reducing startup overhead.
- *InputFIFO* reads.
- Using vector loads to preload *mblt* data.

When the *color* register is written, a memory cycle is needed for the next monochrome fill to load the SGRAMs color register. To avoid this memory cycle, the color register should only be loaded if it has changed from the value used by the previous monochrome fill. If a monochrome fill is done after certain color fills (cbrush8 with 8 or 16 bits/pixel or cbrush16 with 8 bits/pixel), the memory cycle can not be avoided, even if the color register is unchanged. Similarly, when the *writemask* is written, the next fill does a memory cycle to write the SGRAMs write mask. This memory cycle can be avoided by not writing the writemask between fills. Note, as a performance optimization only, Windows microcode should avoid using *a0* through *a15*, to prevent unnecessary startup overhead.

The *fill*, *mbrush*, *mbltf*, *mbltfo*, and *cbltf* instructions always launch in one cycle. However, *cbrush8*, *cbrush16*, *cbrush32*, and *mblt* all potentially incur an extra startup overhead. Startup overhead is avoided on the *cbrush* instructions if the color brush data (registers *a8* to *a15*), *format* and *offset* have not been written since the last *cbrush* instruction, and the previous *Fill Engine* instruction was also a *cbrush*. For example, to optimally fill an area with a color brush, every 8th scan line, assuming an 8x8 brush, is filled using *cbrush8* over the entire area without changing the brush. The brush is then changed, and starting with the second scan line every 8th line is again filled. The brush is then changed again and this is repeated for lines starting at line 3, 4, 5, 6, 7, and 8. Start overhead is listed in table 2.11d.

Due to shared resources, there must be an instruction separating any fill instruction from a subsequent draw or rdraw instruction.

Opcode	Encoding		Max Offset (@bpp) 8	16	32	Length	Description
0xE7	**mbltfo**	D,S2,S1	31	31	31	1 to 1984	Monochrome opaque BitBlt using *InputFIFO*.
0xE8	**fill**	D,S2,S1	n/a	n/a	n/a	1 to 1984	Solid fill.
0xE9	**mbrush**	D,S2,S1	31	31	31	1 to 1984	Monochrome brush expansion.
0xEA	**mblt**	D,S2,S1	63	63	63	1 to 255	Monochrome BitBlt using registers as source.
0xEB	**mbltf**	D,S2,S1	31	31	31	1 to 1984	Monochrome transparent BitBlt using *InputFIFO*.
0xEC	**cbrush8**	D,S2,S1	7	7	7	1 to 1984	8 pixel wide color brush fill. For all depths.
0xED	**cbrush16**	D,S2,S1	15	15	15	1 to 1984	16 pixel wide color brush fill. For 8 and 16 bpp only.
0xEE	**cbrush32**	D,S2,S1	31	31	31	1 to 1984	32 pixel wide color brush fill. For 8 bpp only.
0xEF	**cbltf**	D,S2,S1	3	1	0	1 to 1984	Color BitBlt (optional chromakey) from *InputFIFO*.

Table 2.11a - *Fill Engine* instruction definitions

Reg	Index	fill	mbrush	mblt	mbltf	mbltfo	Hzrd	Comment
a0	%128	format	format	format	format	format	3	b2:b0 (1, 2, 4 for 8, 16 and 32 bpp).
a1	%129	wmask	wmask	wmask	wmask	wmask	1	Replicated to 32 bits.
a2	%130	color	color	color	color	color	2	Replicated to 32 bits.
a3	%131	n/a	offset	offset	offset	offset	3	Pixel offset.
a8	%136	n/a	mbrush[31:0]	msrc0	n/a	bgcolor	3	brush, Blt data, or background color.
a9	%137	n/a	n/a	msrc1	n/a	n/a	3	BitBlt data.
a10	%138	n/a	n/a	msrc2	n/a	n/a	3	BitBlt data.
a11	%139	n/a	n/a	msrc3	n/a	n/a	3	BitBlt data.
a12	%140	n/a	n/a	msrc4	n/a	n/a	3	BitBlt data.
a13	%141	n/a	n/a	msrc5	n/a	n/a	3	BitBlt data.
a14	%142	n/a	n/a	msrc6	n/a	n/a	3	BitBlt data.
a15	%143	n/a	n/a	msrc7	n/a	n/a	3	BitBlt data.

Table 2.11b - *Fill Engine* parameter definition for solid and monochrome instructions

Reg	Index	cbrush8 8 bpp	cbrush8 16 bpp	cbrush8 32 bpp	cbrush16 8 bpp	cbrush16 16 bpp	cbrush32 8 bpp	cbltf	Hzrd	Comment
a0	%128	format	format	format	format	format	format	format	3	bits[2:0] (1, 2, 4 for 8, 16 and 32 bpp). Bit 7 enables chroma-key.
a1	%129	wmask	wmask	wmask	wmask	wmask	wmask	wmask	1	Replicated to 32 bits.
a2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	key	2	Chroma color
a3	%131	offset	offset	offset	offset	offset	offset	offset	3	Pixel offset.
a8	%136	cbrush0	cbrush0	cbrush0	cbrush0	cbrush0	cbrush0	n/a	3	brush.
a9	%137	cbrush1	cbrush1	cbrush1	cbrush1	cbrush1	cbrush1	n/a	3	brush.
a10	%138	n/a	cbrush2	cbrush2	cbrush2	cbrush2	cbrush2	n/a	3	brush.
a11	%139	n/a	cbrush3	cbrush3	cbrush3	cbrush3	cbrush3	n/a	3	brush.
a12	%140	n/a	n/a	cbrush4	n/a	cbrush4	cbrush4	n/a	3	brush.
a13	%141	n/a	n/a	cbrush5	n/a	cbrush5	cbrush5	n/a	3	brush.
a14	%142	n/a	n/a	cbrush6	n/a	cbrush6	cbrush6	n/a	3	brush.
a15	%143	n/a	n/a	cbrush7	n/a	cbrush7	cbrush7	n/a	3	brush.

Table 2.11c - *Fill Engine* parameter location definition for color instructions

Instruction	Startup Overhead (cycles)	Avoidable
cbrush8 (8bpp)	1	yes
cbrush8 (16bpp)	3	yes
cbrush8 (32bpp)	7	yes
cbrush16 (8bpp)	3	yes
cbrush16 (16bpp)	7	yes
cbrush32 (8bpp)	7	yes
mblt	ceiling((length + offset)/32)	no

Table 2.11d - *Fill Engine* launch overhead

After any issue and startup overhead, the *Fill Engine* instructions run in the background and the RISC can continue on with the instruction stream until it encounters:

- Another *Fill Engine* instruction.
- An instruction that reads from the *InputFIFO* if an *mbltf*, *mbltfo*, or *cbltf* is in progress.
- An instruction that uses the shifter (e.g. *rotr*, *f2I*), if an mbltf, *mbltfo*, or *cbltf* is in progress.
- An instruction that writes register *a8* if an *mbltfo* is in progress.

If any of these situations arises, the RISC stalls until the Fill Engine is finished.

All registers that are used by the Fill Engine are either fully shadowed or only used at *Fill Engine* startup. This allows the RISC to continue on and potentially write *these registers*, without stalling to wait for the *Fill Engine* to finish (except *a8* during *mbltfo*). Register data used by *mblt* is typically loaded from memory. By using the *offset* register, this *mblt* data can be loaded into the RISC with vector loads that are more efficient than single word loads.

2.12 Pipeline Architecture

The RISC processor has five pipeline stages:

F - Fetch the next instruction from the I-Cache
D - Decode the instruction while reading source operands from the register-file
X1 - eXecute the instruction or calculate load/store address.
X2 - Second eXecute cycle for two-cycle instructions. For load, read data. For load or store, read tag.
W - Write result to register file.

The pipeline stages are illustrated in figure 2.12.

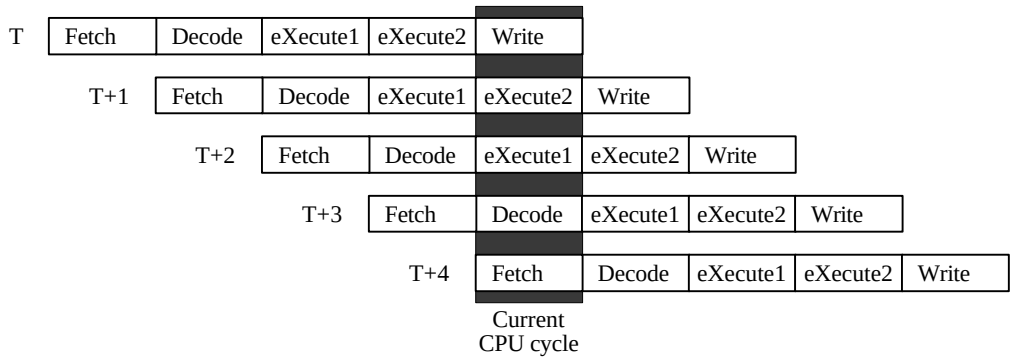


Figure 2.12 -- RISC execution pipeline

3. RISC Register-File Special Hardware Registers

There are a number of special hardware registers. Most of these registers appear to the RISC as part of the standard register-file map. The rest appear as memory locations at the top of the 32-bit RISC address space. For all special hardware registers, unused bits read as 0 and must be written as 0.

3.1 FIFO Registers

There is one input FIFO (32 entry) and one output FIFO (8 entry). Each entry is 32-bits and is read or written as a 32-bit word. These FIFOs are jointly referred to as *FIFO*. *FIFO* can be the source or destination register like any other register-file location. Note, using the *swfifo* as a vector source produces the swapped fifo data in the even source and the non-swapped fifo data in the odd source, and only one entry is popped from the fifo.

When used as a source register, a 32-bit value from the input FIFO is returned (e.g. When *FIFO* is used as the scalar S1 of a vector-scalar instruction, pop exactly one entry and use it for both of the vector operations). The RISC can read the input FIFO data without byte swap (*FIFO*) or with byte swap (*swfifo*, B3↔B0 and B2↔B1). The input FIFO can be checked for empty by testing the *FIFOInRdy* bit of *Flag*. When *FIFO* is used as a destination register, a 32-bit value is written to the output FIFO. The output FIFO can be checked for full by testing the *FIFOOutRdy* bit of *Flag*.

The processor stalls if (input) *FIFO* is empty when it is read, or if (output) *FIFO* is full when it is written. Note, in the V2200 the output FIFO is checked for full in the decode stage to make context switch code simpler. The input and output *FIFO* appear as registers to the system CPU as described in the *PCI IO Register Map* section.

To eliminate a potential deadlock case that occurred in the V1000, the RISC will stall in the decode stage, if the *OutputFIFO* is full. Therefore, the threshold of the *OutputFIFO* being full, as detected by the RISC, is when six entries are full. In addition a new bit has been added to the *Flag* register (bit[30] = *FIFOOutForce*). This bit is intended for state restore code only. When it is set, the RISC will not stall when the *OutputFIFO* is “full”.

3.2 Flag Register

The Flag Register (*Flag*) is defined as shown in table 3.2. All bits are read/write unless specified otherwise.

bit	Name	Description
0	**SignalSys**	When written as a one, sets and holds an interrupt to the system CPU. The System CPU resets this bit by setting the *RISCIntr* bit in the *Comm* register (write = 0 ignored). Writes to this bit have a delay of 3 before a change can be observed by the host.
1	**PixelBusy**	*Triangle* Engine, Pixel *Engine* or *Fill Engine* is busy (=1 if busy). Read only.
2	**FIFOOutRdy**	Output-FIFO ready (=1 if not full). Read only.
3	**ICacheOff**	Force *ICache* to always miss. Set by *Reset*.
7:4	**Status**	RISC updated status visible by host in *Comm* Register, as *RISCStatus*. Writes to these bits have a delay of 3 before a change can be observed by the host. Read/write.
11:8	**SysStatus**	Four-bit field set by system processor to send status information to the RISC. Read only.
16	**PickValid**	*PickValid* bit from last *spr Pick*. Wait for *PixelBusy* = 0 before reading. See text. Read only.
17	**PickHit**	Set if there has been a draw hit. Wait for *PixelBusy* = 0 before reading. See text. Read only.
30	**FIFOOutForce**	If set, RISC does not stall on a full *OutputFIFO*. Read/write. Cleared by *Reset*.
31	**FIFOInRdy**	Input-FIFO ready (=1 if not empty). Read only.

Table 3.2 -- *Flag* register bit definition

3.3 Register File Segment Register

Source and destination registers can be specified directly or indirectly. Indirect accesses allow the register-file to be used as a scratch-pad memory. For direct accesses, the operand index field is used directly as the 8-bit index.

Register file locations from %128 to %255 can be indirectly accessed using the *Seg* register. *Seg* contains four 3-bit base pointers: *Seg*(0) in b2:b0, *Seg*(1) in b6:b4, *Seg*(2) in b11:b8, and *Seg*(3) in b14:b12. For indirect accesses the two most significant bits of the operand index field (b7:b6) are set to 2'b01. The next two operand index field bits (b5:b4) are used to specify which segment pointer to use. The register file location to access is specified as:

Index = 128 | (((*Seg* >> (*OperandIndex*[b5:b4] * 4)) & 7) * 16) | *OperandIndex*[b3:b0].

After *Seg* is updated, there is a hazard of 3 cycles before it can be used as a segment base. Note, *Seg* register b3, b7, b11 and b15 always read as 1.

3.4 Clip register

The *Clip* register contains the flag results from the last six *clip*, last three *clipv* or last two *sort* instructions in b7:b2 (non-interlocked delay of 1). For 2D operations, the last four clip flag results can be tested using the *winclip* register. Unused bits read as 0. A vector *clip* operation generates two new flag bits, with the even-address registers being compared first. The *sort* instruction generates four new (identical) flag bits. New flag bits are shifted into b2, with old flag bits being shifted left as the new flag bits are generated.

3.5 Seed and Scale Registers

These special registers implement the inverse seed lookup. The *seed S2* (a *step q* in hardware) instruction updates these registers. The location of the most significant 1 bit of the *seed S2* input can be found by reading the *Scale* register. The *Scale* register returns 0 if input == 0, otherwise it returns the leading 1 bit location + 1 (e.g. if input == 8, Scale = 4). The left-aligned output of the seed-ROM can be read using the *Seed* register. The normalized input 13-bit index into the seed ROM can be read in b31:b19 of *Seg*. These registers are not interlocked. See the corresponding entries in the *Register Definition* section and the *seedsr* instruction for instruction hazard delays for these registers.

3.6 RISC Interrupt Register

The RISC interrupt register's name (*riscintr*) is a slight misnomer. It is not a true asynchronous interrupt. Instead it is used to test the interrupt status in the dispatch loop, and then branch to the interrupt service routine, if there is a pending interrupt. Since the dispatch loop is executed before every primitive, this operates as an asynchronous interrupt with a maximum latency of one primitive, hence the name. The RISC interrupt register definition is shown in table 3.6.

bit	Name	Description
0	**VertIntr**	Vertical retrace interrupt. Bit set, and held, at start of vertical-retrace, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
1	**XIntr**	Externally initiated interrupt. Edge sensitive pass-through from *Xintr* input. Polarity of active edge is programmed by the *XBusCtl* register. Bit asserted even if this interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
2	**VInEvenIntr**	Video input even field interrupt. If video input is enabled, bit set and held at end of even field, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
3	**VInOddIntr**	Video input odd field interrupt. If video input is enabled, bit set and held at end of even field, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
4	**WriteIntr0**	Interrupt on write to memory byte specified by WriteIntr0Adr register (see *Video Input* section). Set even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
5	**WriteIntr1**	Interrupt on write to memory byte specified by WriteIntr1Adr register (see *Video Input* section). Set even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
14	**VInOddIntrLast**	Hardware sets this bit if the last video input interrupt was *VInOddIntr*. Read only.
15	**WriteIntr1Last**	Hardware sets this bit if the last memory write interrupt was *WriteIntr1*. Read only.
16	**VertIntrEn**	Enable the vertical retrace interrupt. Read/write.
17	**XIntrEn**	Enable the external interrupt. Read/write.
18	**VInEvenIntrEn**	Enable the video input even field interrupt. Read/write.
19	**VInOddIntrEn**	Enable the video input odd field interrupt. Read/write.
20	**WriteIntr0En**	Enable the memory write 0 interrupt. Read/write.
21	**WriteIntr1En**	Enable the memory write 1 interrupt. Read/write.
31	**RISCIntr**	This bit is asserted if any enabled RISC interrupts are true.

Table 3.6 – RISC Interrupt Register bit definition

4. Pixel Engine

The *Pixel Engine* is controlled by the RISC processor, or the *Triangle Engine*, on a per-pixel (or 32-bit) basis. The RISC has four *Pixel Engine* instruction types. For example, *draw1* (r*draw1*) which draws one (many) pixels using the *DrawCtl* register to specify which registers to iterate, *drawp* (*rdrawp*) which draws one (many) blocks of pixels (a block of pixels is all or part of a 32-bit word), *step* which sends data to the pixel engine and then updates the specified register(s), and *spr* which sets one of the *Pixel Engine* state registers. The drawing instructions are described in the *Drawing Instructions* section. The *step* instructions are specified in the *Step Instructions* section.

For *spr*, *S2* is used as data. *S1* specifies which *Pixel Engine* register, or register field to update. Register indexes are shown in table 4. Register field *spr* index values are shown in their respective register description tables. Note a field's bit position does not change between *spr register* and *spr field*. These mode registers are pipelined in the *Pixel Engine* (i.e. no need to wait for the *Pixel Engine* to be idle).

spr	*Pixel Engine* Register Description
0	**SrcBase.** Set byte address of source. Byte aligned, except word aligned if a YUYV source.
1	**SrcMode.** See text
2	**SrcMask.** See text
3	**Stride.** See text
4	**DstBase.** Set byte address of destination. Must be word aligned.
5	**DstMode.** See text
6	**DstFmt.** See text
7	**PMask.** See text
8	**Pattern.** Load 32-bit pattern register. See text
10	**IPat.** 32-bit pattern register = WordAtMemByteAdr(S2). The address is byte aligned. See text
12	**PatMode.** Set pattern length minus 1, and pattern enable and opaque bits. See text
13	**PatOffset.** (0 ≤ S2 ≤ 31). The *PatOffset* is also reset to 0 by *spr* writes of *Pat* and *IPat*. See text
14	**ScissorX.** ScissorL = b27:b16, ScissorR = b11:b0. Pixels are drawn if ScissorL ≤ X < ScissorR and 0 ≤ X < 2048
15	**ScissorY.** ScissorT = b27:b16, ScissorB = b11:b0. Pixels are drawn if ScissorT ≤ Y < ScissorB and 0 ≤ Y < 2048
16	**ZBase.** Set byte address of Z buffer. Must be half-word aligned
17	**DitherOffset.** See text
19	**FGColor.** Foreground color. See text
20	**BGColor.** Background color. See text
21	**FogColor.** See text
22	**DstColor.** Used as current *DstColor* if *DstRdDisable* is set. Must be explicitly set after entering *DstRdDisable* mode. Data destroyed by any *Pixel Engine* destination read. See text.
23	**ChromaColor.** 32-bit Chroma key index. See text.
24	**YUVContrast.** YUV2RGB contrast control in b7:b0. See text.
26	**ChromaMask.** 32-bit Chroma key index mask. See text.
27	**FGColorRGB.** Same as *spr FGColor*, except the *Alpha* value is not modified.
28	**Pick.** *PickValid* = b16 and *PickHit* = b17. Current values appear in *Flag* register. See text.
29	**AlphaThres**. If *TranspReject* is enabled, any destination pixel with *Alpha* ≤ *AlphaThres* (b7:b0) is not written.
30	**SpecColor**. Set the specular color register. Red in b[23:16], green in b[15:8] and blue in b[7:0].
31	**Sync.** Can be unconditional (b16 == 0) or conditional (b16 = 1). An unconditional *Sync* causes the *Pixel Engine* to wait until previous drawing commands are completed (i.e. *Pixel Engine* idle) before continuing. A conditional *Sync* waits only if the Destination is being read, or the Z-buffer is being read.
32 - 39	**Tex4Pal.** Eight color pairs for 4-bit indexed textures. Even index colors are in b31:b16, and odd index colors are in b15:b0 (*spr 32* is *Tex4Pal_01, … , spr 39* is *Tex4Pal_ef*). The 16-bit color can be 565, 1555 or 4444 as specified in the *SrcFmt* section of the *SrcMode* register description.
48 - 55, 61 - 63, 83	Bit field *spr*'s for the *SrcMode* register.
56 - 57	Bit field *spr*'s for the *SrcMask* register.
58 - 60	Bit field *spr*'s for the *Stride* register.
64 - 76	Bit field *spr*'s for the *DstMode* register.
80 - 82	Bit field *spr*'s for the *PatMode* register.

Table 4 -- *Pixel Engine spr* commands

4.1 SrcMode

The *spr SrcMode* command uses S2 to set source pixel mode (*SrcMode*) as shown in table 4.1. Note, *Alpha* is considered opaque (= 255) for source formats that do not explicitly have an *alpha* field.

bits	spr	Name	Description
3:0	48	**SrcFmt**	Source texture-format is shown below (Set = 0 for BitBlt and other 2D operations). Note, if *YUV2RGB* mode is enabled fields map to: C_r = r, Y = g, C_b = b, except *SrcFmt* = 0xD has two Y values (Y_0 for even U and Y_1 for odd). **Value Definition** — **Value Definition** 0 = DstFmt — 6 16-bit 1555 a, r, g, b 1 8-bit 332 r, g, b — 8 4-bit indexed 565 r, g, b 2 8-bit intensity — 9 4-bit indexed 4444 a, r, g, b 3 8-bit alpha (r, g, b = 255) — A 4-bit indexed 1555 a, r, g, b 4 16-bit 565 r, g, b — C 32-bit 8888 a, r, g, b 5 16-bit 4444 a, r, g, b — D 32-bit 8888 4:2:2 (Y_0, C_r, Y_1, C_b)
6:4	49	**SrcFunc**	Source function is shown below (*Color* is foreground or background value): **Value Definition** 0 No texture. *SrcColor* = *Color*. *SrcAlpha* = *ColorAlpha* 1 Replace. *SrcColor* = *TexColor*. *SrcAlpha* = *TexAlpha* (use for BitBlt) 2 Decal. *SrcColor* = (1 - *TexAlpha*) * *Color* + *TexAlpha* * *TexColor*. *SrcAlpha* = *ColorAlpha* 3 Modulate. *SrcColor* = *Color* * *TexColor*. *SrcAlpha* = *TexAlpha* * *ColorAlpha*
7	50	**SrcFilter**	Source texture-filtering function. (=0: BitBlt and point sample; =1: bilinear filtering).
9	51	**SrcColorNoPad**	If set, low bits of colors = 0 when converting to 8888 color in the *Pixel Engine*. If cleared, correct low bits during conversion (e.g. 8-bit-color = floor(5-bit-color*255/31 + 0.5)).
10	52	**SwapUV**	Swap meaning of U and V. Must be written before *UMask* or *VMask* are used.
11	53	**ChromaKey**	Enable chroma-key. See text.
12	54	**ZScissorEn**	If set, scissor pixels that overflow in Z (*ZOvf*), even if Z buffer is off. *ZOvf* is calculated as *Z* is written, so a *Z* value must be written after *ZScissorEn* is cleared to clear *ZOvf*: ZOvf = ((Z & 0xf8000000) != 0) && ZScissorEn, applied as *Z* written to Pixel Engine.
13	55	**SrcBGR**	If set, *Red* is in LSBs of source. If cleared, *Blue* is in LSBs of source. Also see *SrcFmt*.
14	61	**ChromaBlackYUV**	If set and chroma key matches, texture data *(a, r, g, b)* = (0, 128, 16, 128). If clear and chroma key matches, texture data *(a, r, g, b)* = (0, 0, 0, 0).
15	62	**UClamp**	If set, clamp the texture *U* value such that $0 \le U \le UMask$.
16	63	**VClamp**	If set, clamp the texture *V* value such that $0 \le V \le VMask$.
17	83	**SpecularEn**	If set, enable specular color add in *Pixel Engine* for *Triangle Engine* primitives (but not *draw* or *rdraw*). Cleared by *Reset*.

Table 4.1 -- *SrcMode* register bit definition

4.2 SrcMask

SrcMask specifies the U and V offset mask to allow "wrapping" textures as specified in Table 4.2. Each mask is set to one less than the texture height or width in pixels/texels (e.g. for a height of 128 and a width of 64, S2=0x7F003F). The maximum texture width or height is 2048. Note, the UMask, VMask, and SwapUV values are applied as U and V are received by the pixel engine, as opposed to when a pixel is drawn, so they must be set before sending *U* or *V* to the *Pixel Engine*. For example, the code sequence below on the left uses the *SrcMask* data in *r0*, and the right one uses *r1*:

```
spr  SrcMask,r0                spr  SrcMask,r0
step U,D,S2,S1                 step U,D,S2,S1
spr  SrcMask,r1                spr  SrcMask,r1
draw     xy,r4,r6      ; SrcMask=r0      draw     xquiv,r4,r6   ; SrcMask=r1
```

bits	spr	Name	Description
10:0	56	**UMask**	Mask offset into texture map in U direction.
26:16	57	**VMask**	Mask offset into texture map in V direction.

Table 4.2 -- *SrcMask* register bit definition

4.3 Stride

The Stride register is used for pixel address calculations (source, destination and Z). Shift distance translation from value to distances are shown in table 4.3a and resulting widths are in table 4.3b. The pixel address calculation is:

Address = Base + ((Stride0) ? (Y << Stride0) : 0) + ((Stride1) ? (Y << Stride1) : 0) + (X * XDepthInBytes).

bits	spr	Name	Description
2:0	58	**SrcStride0**	Frame-width shift-left distance 0 (For pixel/texel address calculation, in power-of-two bytes) **Value Distance** — **Value Definition** 0 zero all bits — 4 2 1 8 — 5 8 and 9. For 800 by 600 by 8 (SrcStride1 = 2). 2 9 — 6 9 and 6. For 800 by 600 by 16. (SrcStride1 = 5). 3 10 — 7 10 and 7. For 800 by 600 by 32. (SrcStride1 = 6).
6:4		**SrcStride1**	Frame-width shift-left distance 1 (For pixel/texel address calculation, in power-of-two bytes) **Value Distance** — **Value Definition** 0 zero all bits — 4 7 1 4 — 5 10 2 5 — 6 11 3 6 — 7 12
10:8	59	**DstStride0**	Destination frame-width shift-left distance 0. Conversion same as for SrcStride0.
14:12		**DstStride1**	Destination frame-width shift-left distance 1. Conversion same as for SrcStride1.
18:16	60	**ZStride0**	Z-buffer frame-width shift-left distance 0. Conversion same as for SrcStride0.
22:20		**ZStride1**	Z-buffer frame-width shift-left distance 1. Conversion same as for SrcStride1.

Table 4.3a -- *Stride* register bit definition

Stride field		Width in pixels			Stride field		Width in pixels		
1	**0**	**8 bpp**	**16 bpp**	**32 bpp**	**1**	**0**	**8 bpp**	**16 bpp**	**32 bpp**
0	0	0	0	0	5	4	1028	514	257
0	4	4	2	1	1	3	1040	520	260
1	0	16	8	4	2	3	1056	528	264
1	4	20	10	5	3	3	1088	544	272
2	0	32	16	8	4	3	1152	576	288
2	4	36	18	9	1	7	1168	584	292
3	0	64	32	16	2	7	1184	592	296
3	4	68	34	17	3	7	1216	608	304

4	0	128	64	32	5	1	1280	640	320
4	4	132	66	33	5	2	1536	768	384
0	1	256	128	64	5	6	1600	800	400
1	1	272	136	68	5	5	1792	896	448
2	1	288	144	72	6	0	2048	1024	512
3	1	320	160	80	6	4	2052	1026	513
4	1	384	192	96	5	7	2176	1088	544
0	2	512	256	128	6	1	2304	1152	576
1	2	528	264	132	6	2	2560	1280	640
2	2	544	272	136	6	6	2624	1312	656
3	2	576	288	144	6	5	2816	1408	704
1	6	592	296	148	6	3	3072	1536	768
2	6	608	304	152	6	7	3200	1600	800
4	2	640	320	160	7	0	4096	2048	1024
4	6	704	352	176	7	4	4100	2050	1025
0	5	768	384	192	7	1	4352	2176	1088
1	5	784	392	196	7	2	4608	2304	1152
2	5	800	400	200	7	6	4672	2336	1168
3	5	832	416	208	7	5	4864	2432	1216
4	5	896	448	224	7	3	5120	2560	1280
0	3	1024	512	256	7	7	5248	2624	1312

Table 4.3b – Available stride widths

4.4 DstMode

The *spr DstMode* command uses *S2* to set Destination drawing mode (*DstMode*), as shown in table 4.4. For 2D drawing operations this register is typically set = *ALUMode* (other bits = 0). The Pixel engine compares the destination result with the destination read value and if they match, no write is performed (if destination wasn't read, the destination result is always written).

The following equation is used to determine if the destination will be read by the pixel engine:

```
ReadDst = !DstRdDisable && (
  ((PMask != 0xffffffff) && (PMask != 0)) || (ALUMode == nor(s,d)) || (ALUMode == and(~s,d))
  || (ALUMode == and(s,~d)) || (ALUMode == (~d)) || (ALUMode == XOR(s,d)) || (ALUMode == nand(s,d))
  || (ALUMode == and(s,d)) || (ALUMode == xnor(s,d)) || (ALUMode == (d)) || (ALUMode == or(~s,d))
  || (ALUMode == or(s,~d)) || (ALUMode == or(s,d))
  || BlendEnable &&
       ( (BlendSrcFunc == BlendSrcAlphaSat) || (BlendSrcFunc == BlendSrcAlphaSatInv )
         || (BlendSrcFunc == BlendDstColor ) || (BlendSrcFunc == BlendDstColorInv)
         || (BlendSrcFunc == BlendDstAlpha ) || (BlendSrcFunc == BlendDstAlphaInv)
         || (BlendDstFunc != Blend0) )      )
```

bit(s)	spr	Name	Description
3:0	64	**ALUMode**	Pixel destination ALU update modes are as shown below (S = source, D = Destination): **Value Definition** — **Value Definition** 0 (AllBits0) — 8 AND(S,D) 1 NOR(S,D) — 9 XNOR(S,D) 2 AND(~S,D) — a (D) 3 (~S) — b OR(~S,D) 4 AND(S,~D) — c (S) (Replace mode) 5 (~D) — d OR(S,~D) 6 XOR(S,D) — e OR(S,D) 7 NAND(S,D) — f (AllBits1)
7:4	65	**BlendSrcFunc**	Source blending functions are as shown below: **Value Definition** — **Value Definition** 0 BlendDstColor — 7 Blend1 1 BlendDstColorInv — 8 BlendSrcAlphaSat 2 BlendSrcAlpha — 9 BlendSrcAlphaSatInv 3 BlendSrcAlphaInv — 10 BlendSrcColor 4 BlendDstAlpha — 11 BlendSrcColorInv 5 BlendDstAlphaInv — 15 BlendAddSignedSrc 6 Blend0
11:8	66	**BlendDstFunc**	Destination blending functions are as shown below: **Value Definition** — **Value Definition** 0 BlendSrcColor — 7 Blend1 1 BlendSrcColorInv — 8 BlendSrcAlphaSat 2 BlendSrcAlpha — 9 BlendSrcAlphaSatInv 3 BlendSrcAlphaInv — 10 BlendDstColor 4 BlendDstAlpha — 11 BlendDstColorInv 5 BlendDstAlphaInv — 15 BlendAddSignedDst 6 Blend0
14:12	67	**ZBufMode**	Z-buffer modes are shown below. To turn Z-buffer read off, set ZBufMode = 0. **Value Definition** — **Value Definition** 0 Replace always (No Z read) — 4 Replace if NewZ > OldZ 1 Replace if NewZ < OldZ — 5 Replace if NewZ != OldZ 2 Replace if NewZ = OldZ — 6 Replace if NewZ ≥ OldZ 3 Replace if NewZ ≤ OldZ — 7 Replace never (No Z read).
16	68	**ZBufWrMode**	The Pixel engine compares the NewZ with the OldZ value. If they match, no write is performed (except if *ZBufWrMode* is set and Z wasn't read, the Z result is written). **Value Definition** — **Value Definition** 0 Off (no Z write) — 1 Write Z if Z test passes
18	69	**YUV2RGB**	If set, treat *SrcColor* as YUV value and convert to RGB in the blend unit. Other blend functions are disabled when this bit is set.
19	70	**BlendEnable**	If set, enable blending.
20	71	**DitherEnable**	If set, enable pixel dithering. See the *Dither* section for description.
21	72	**FogEnable**	If set, enable fog mode.
22	73	**DstColorNoPad**	If set, low bits of colors = 0 when converting to 8888 RGBA in *Pixel Engine*. If reset, correct low bits during conversion (e.g. 8-bit-color = floor(5-bit-color*255/31 + 0.5)).
23	74	**DstRdDisable**	If set, disable the destination read and use the *DstColor* register as destination data. When = 0 the Pixel Engine uses the ALUMode, etc. to decide if it needs to read the destination.
24	75	**DstBGR**	If set, *Red* is in LSBs of destination. If cleared, *Blue* is in LSBs of destination. (In *DstRdDisable* mode, the *DstColor* register is never swapped).
25	76	**TranspReject**	If set, pixels with texture unit output of *Alpha* ≤ *AlphaThres* will not be written, independent of whether the destination is being read.

Table 4.4 -- *DstMode* register bit definition

4.5 DstFmt

The *spr DstFmt* command uses *S2* to set destination format (*DstFmt*) as shown in table 4.5.

bit(s)	Name	Description
3:0	**DstFmt**	Drawing engine destination pixel format is as shown below: **Value** **Definition** — **Value** **Definition** 0 Reserved — 4 16-bit 565 r, g, b 1 8-bit 332 r, g, b — 5 16-bit 4444 a, r, g, b 2 8-bit intensity, 8 r — 6 16-bit 1555 alpha, red, green, blue 3 8-bit alpha — C 32-bit 8888 alpha, red, green, blue

Table 4.5 -- *DstFmt* register bit definition

4.6 PMask

The *spr PMask* command uses the data in the *S2* to set all 32-bits of the plane write mask. For 8-bit and 16-bit pixels, the plane mask must be replicated to fill all 32-bits. If a bit is set, it enables *Pixel Engine* writes to its respective plane (bit 0 enables plane 0, etc.). If a bit is zero, it prevents writes by the *Pixel Engine* to that plane.

4.7 Pattern Registers

The 32-bit pattern register can be set directly (e.g. *spr Pattern*), or by loading a word from memory (e.g. *spr IPat*). Note, *spr IPat* loads a word from an arbitrary byte aligned memory location. *PatOffset* is the current pattern-bit (bit = ~*PatOffset* & 0x1F). The bit 31 of *Pat* is the first or left-most, and the bit 0 is the last or right-most pattern bit. *PatOffset* is updated after each pixel draw (even if scissored), if the pattern is enabled, as follows:

PatOffset = (*PatOffset* != *PatLengthM1*) ? *PatOffset* + 1 : 0;

If the pattern is enabled, the current pattern-bit determines if the current-pixel is a foreground-pixel (bit = 1) or background-pixel. If the pattern is transparent, background pixels are not modified. The pattern is independent of, and can be applied to, BitBlt and textures. Note, patterning works for parallel draw operations (e.g. *rdrawp*), but only if drawing from left to right. Table 4.7 defines the *PatMode* register.

bit(s)	spr	Name	Description
4:0	80	PatLengthM1	Pattern width - 1. The pattern must be at least 8/sizeof(pixel) wide for parallel draws.
16	81	PatEnable	Enable pattern using 32-bit pattern register.
17	82	PatOpaque	Pattern register specifies an opaque pattern if set. If reset, background pixels not drawn.

Table 4.7 -- Pixel Engine *PatMode* register definition

4.8 Dither

The dither function uses a 4 by 4 perfect-magic-square dither. Use of the *DitherOffset* allows dithered images to be bit identical, independent of the window alignment. The *DitherOffset* register specifies the two-bit dither offsets for X (b1:b0) and Y (b5:b4). These bits are added to the low two bits of the integer portion of X and Y and the results (modulo 4) are used as the dither matrix index as specified below. The dither matrix is shown in table 4.8.

		X			
		0	**1**	**2**	**3**
Y	**0**	0	12	3	15
	1	7	11	4	8
	2	13	1	14	2
	3	10	6	9	5

Table 4.8 -- Dither matrix

The dither functions are:

Color6 = min(Color8 + (dither[X[1:0] + *DitherOffset*[1:0], Y[1:0] + *DitherOffset*[5:4]] >> 2), 255) >> 2.
Color5 = min(Color8 + (dither[X[1:0] + *DitherOffset*[1:0], Y[1:0] + *DitherOffset*[5:4]] >> 1), 255) >> 3.
Color4 = min(Color8 + (dither[X[1:0] + *DitherOffset*[1:0], Y[1:0] + *DitherOffset*[5:4]]), 255) >> 4.
Color3 = min(Color8 + (dither[X[1:0] + *DitherOffset*[1:0], Y[1:0] + *DitherOffset*[5:4]] << 1), 255) >> 5.
Color2 = min(Color8 + (dither[X[1:0] + *DitherOffset*[1:0], Y[1:0] + *DitherOffset*[5:4]] << 2), 255) >> 6.

4.9 FGColor

The *spr FGColor* command uses S2 to set the *FGColor*. For single pixel drawing, *FGColor* is defined as: *Alpha* in b31:b24, *Red* in b23:b16, *Green* in b15:b8 and *Blue* in b7:b0 (8888 ARGB). When used as an 8-bit or a 16-bit pixel, the data must be replicated to 32-bits. For parallel drawing, *FGColor* contains 4/sizeof(pixel) pixels of data (i.e. four 8-bit pixels, two 16-bit pixels or one 32-bit pixel). The left pixel is in the most significant bits of the word and always has the same alignment to screen pixels: 0, 4, 8, etc. for 8-bit pixels and 0, 2, 4, etc. for 16-bit pixels. Operations that update *Red*, *Green*, *Blue*, or *Alpha* overwrite their respective fields of the *FGColor* register, but other *FGColor* fields remain unchanged. Note, *spr FGColorRGB* functions the same as *spr FGColor*, except the *Alpha* value is not modified. Note, an *Alpha* value of 255 is considered opaque and a value of 0 is considered transparent.

4.10 BGColor

The *spr BGColor* command uses *S2* to set the *BGColor*. The *BGColor* is used if the *Pattern* is enabled, the current pixel is a background pixel and the pattern is opaque. If the pattern is transparent, background pixels are not modified. The format of *BGColor* is the same as for the *FGColor* register.

4.11 FogColor

The *spr FogColor* command uses S2 to set the *FogColor*. Red is in b23:b16, Green is in b15:b8 and Blue is in b7:b0. Fog blending is enabled by setting the *FogEnable* bit of the *DstMode* register. The fog factor (*f*) is used to blend the *SrcColor* with the *FogColor*. The fog function is: *NewColor* = *f* * (*SrcColor* – *FogColor*) + *FogColor*, (where an *f* value of 0 implies full fog and 255 implies no fog).

4.12 DstColor

If *DstRdDisable* bit of the *DstMode* register is set, this color register is used as *DstColor* in all calculations. Alpha is in b31:b24, Red is in b23:b16, Green is in b15:b8 and Blue is in b7:b0 (8888 ARGB). Setting *DstRdDisable* disables all reads of the destination pixel. After setting *DstRdDisable*, a *spr DstColor* is required before *DstColor* is valid. Any frame-buffer destination read will destroy the value in this register.

4.13 Pick

Pick helps collision detection. Current data can be read in the Flag register. Table 4.13 shows the *spr Pick* description. These two bits are unspecified after *Reset*.

bit(s)	Name	Description
16	PickValid	During *spr Pick*, set to the value of b16 (only *spr Pick* affects value).
17	PickHit	Set and held when a pixel draw passes visibility tests (i.e. *Scissor*, *Pattern*, *Z* and *Alpha* test) but before *PMask* is applied. During *spr Pick*, set to the value of b17.

Table 4.13 -- Pixel Engine *Pick* register definition

A typical pick sequence of operations is:

```
while(PickValid) ;          /* Wait for PickValid FALSE from last primitive. */
spr(Pick, if1_0);           /* Set PickValid = TRUE. Pipelined in Pixel Engine. */
for(each primitive)
   {
      draw(primitive);      /* Draw object to be tested. */
      if(PickHit)
         flushRestOfPrimitives();  /* If a hit, flush rest of primitives. */
   }
while(PixelBusy) ;          /* Wait until done. */
result = PickHit;
spr(Pick, zero);            /* Set PickValid = FALSE for next time. */
return result;              /* Return with result. */
```

4.14 ChromaKey

If *ChromaKey* bit of the *SrcMode* register is set, the *ChromaColor* is used as the chroma-key reference color. When bits enabled by the *ChromaMask* are the same in the texel data and the *ChromaColor*, the texel is treated as transparent and black (see the *ChromaBlackYUV* bit of the *SrcMode* register for more information). The chroma match equation is:

match = ((LeftJustifiedRawTexelData & *ChromaMask*) == (*ChromaColor* & *ChromaMask*))

ChromaColor and *ChromaMask* are 32-bit values as shown in table 4.14. The mask and color values are interpreted differently depending on mode. Also, 8-bit and 16-bit pixels must be replicated as shown. *ChromaKey* works in parallel draw mode by clearing byte enables for pixels that have a *ChromaKey* match. Note, *ChromaKey* is not available with 4-bit indexed texture sources. The *TranspReject* mode with *SrcFunc* = 0xA (4-bit indexed to 1555) can be used to achieve this functionality. Note, it is illegal to enable *ChromaKey* when *SrcFunc* = 0 (i.e. no texture).

SrcBGR	Src Type	**ChromaMask** (M = Mask) 31:24	23:16	15:8	7:0	**ChromaColor** (C = Key Color) 31:24	23:16	15:8	7:0
n/a	**8-bit parallel**	M	M	M	M	C	C	C	C
	8-bit index	M	M	0	0	C	C	n/a	
	16-bit parallel	M		M		C		C	
	32-bit parallel	M				C			
0	**8-bit rgb**	M_{rgb}	M_{rgb}	0	0	C_{rgb}	C_{rgb}	n/a	
	16-bit rgb	M_{rgb}		M_{rgb}		C_{rgb}		C_{rgb}	
	24-bit rgb	0	M_R	M_G	M_B	n/a	C_R	C_G	C_B
	32-bit argb	M_A	M_R	M_G	M_B	C_A	C_R	C_G	C_B
	$Y_0C_rY_1C_b$	M_Y	M_{Cr}	M_Y	M_{Cb}	C_Y	C_{Cr}	C_Y	C_{Cb}
1	**8-bit bgr**	M_{bgr}	M_{bgr}	0	0	C_{bgr}	C_{bgr}	n/a	
	16-bit bgr	M_{bgr}		M_{bgr}		C_{bgr}		C_{bgr}	
	24-bit bgr	0	M_B	M_G	M_R	n/a	C_B	C_G	C_R
	32-bit abgr	M_A	M_B	M_G	M_R	C_A	C_B	C_G	C_R
	$Y_0C_bY_1C_r$	M_Y	M_{Cb}	M_Y	M_{Cr}	C_Y	C_{Cb}	C_Y	C_{Cr}

Table 4.14 -- *ChromaKey* mask and color

4.15 Color Space Conversion

Color space conversion transforms YCbCr (also referred to as YUV) to RGB when the *YUV2RGB* bit is set in the *DstMode* register.. YCrCb (and YCbCr) are supported as *Pixel Engine* texture source formats. YCrCb source data is generated in the same way as all other source formats. This source data is converted to RGB using the blend hardware, disallowing use of blend or fog functions during color space conversion. During color space conversion, *FogColor* must be set as (r, g, b) = (128, *YUVBlack*, 128). The *FogColor* green value (*FogColor[15:8]*) is used as a unsigned 8-bit value which, for reference only, is called *YUVBlack*. *YUVBlack* should nominally be set to 16, to match MPEG. Larger values darken all pixels. In addition, *Pixel Engine* register 24 (*YUVContrast[7:0]*) controls contrast. *YUVContrast* is an unsigned 8-bit value which defaults to 41 after *Reset*, to match MPEG. Larger values increase the contrast ratio, which helps compensate for the linear color ramps often used in Windows applications and desktops.

The color space conversion equations are:

sY = (*YUVContrast* + 256) * (Y - *YUVBlack*) + 128

R = floor((sY + 407 * (Cr - 128))/255)

G = floor((sY - 207 * (Cr - 128) - 100 * (Cb -128))/255)

B = floor((sY + 515 * (Cb-128))/255)

5. PCI Configuration Space, Control Registers and Memory Interface

The V2200 requests 16M bytes of physical memory space, 128K bytes of BIOS ROM memory space from *PCI Power-Up-Software*. In addition, the IO interface is mapped into both a 256 bytes of IO space and a 256K byte memory-mapped-IO (MMIO) space by *PCI Power-Up-Software*. See the *Native IO Register* section for register information. A 16M byte block for memory is requested independent of how much local memory is actually installed. Access to all of local memory is available at any time. However, to assure coherency with *Pixel Engine* data, PCI accesses must be synchronized by waiting for the pixel engine to be idle, unless software determines that PCI and *Pixel Engine* accesses are not to the same addresses. Also, for best performance, the RISC processor and *Pixel Engine* should be idle during PCI memory accesses. The PCI and RISC view of memory and memory mapped I/O is shown in table 5.

RISC Address	PCI Memory Address	Description
`0x00000000` to `0x00ffffff`	`0xSS000000` to `0xSSffffff`	Up to 16M bytes of local memory.
`0xfffe0000` to `0xffffffff`	`0xRRRR0000` to `0xRRRR0000 + 0x1ffff`	BIOS ROM. 128K byte local ROM address range. PCI address on 128K byte boundary.
`0xffe00400` to `0xffe004f8`	n/a (See I/O map)	CRTC registers. See text.
`0xffe00500`	n/a (See I/O map)	Memory control register. See text.
`0xffe00580` to `0xffe005f8`	n/a (See I/O map)	Color palette. See text.
`0xffe00600` to `0xffe00608`	n/a (See I/O map)	PLL control registers. See text.
`0xffe00680` to `0xffe00688`	n/a (See I/O map)	Video input control register. See text.
`0xffe00700`	n/a	Sub-vendor identification field. 32-bit. Write only. Can be read at PCI configuration address 0x2c.
`0xffe00704`	n/a	If set, PCI configuration space specifies a 66 MHz bus, else 33 MHz. Bit cleared by *Reset*. Write only.
`0xffe00800` to `0xffe00ffc`	n/a (See I/O map)	Pixel Engine state. Word wide registers on 8 byte boundaries. See *Context Switching* section.

Table 5 -- Memory map

5.1 PCI Configuration Header

All required configuration registers are supported for a type 0 PCI Configuration Space Header. In addition, *Base Address Registers* for the memory space, I/O mapped register space and memory mapped register space are implemented. The PCI Configuration Space Header definition is shown in table 5.1.

Address	31:24	23:16	15:8	7:0
0x00	Device ID (0x2000)		Vendor ID (0x1163)	
0x04	Status		Command	
0x08	Class Code (0x030000, 0x038000 or 0x048000)			Revision ID
0x0c	BIST (n/a)	Header Type (=0)	Latency Timer	Cache Line Size
0x10	Memory Base Address (16M bytes)			
0x14	I/O Mapped Register Base Address (256 bytes)			
0x18	Memory Mapped Register Base Address (256K bytes)			
0x1c	n/a			
0x20	n/a			
0x24	n/a			
0x28	Cardbus CIS Pointer (n/a)			
0x2c	Subsystem ID (from ROM)		Subsystem Vendor ID (from ROM)	
0x30	Expansion ROM Address (128K bytes)			
0x34	Reserved			
0x38	Reserved			
0x3c	Max_Lat (=0)	Min_Gnt (=0)	Interrupt Pin (=1)	Interrupt Line

Table 5.1 – PCI Configuration Header Definition

5.2 Native IO Registers

The Native IO registers are shown in table 5.2. Addresses are specified as the byte *offset* from the allocated base IO port or the byte *offset* from the MMIO base address. The specified byte address offset for control registers assumes a little-endian (e.g. x86) processor. IO registers have the same byte order independent of memory byte swap. For MMIO, each of the four FIFO byte swap locations are repeated throughout an entire 32K byte section, so a “repeat move string” instruction can be used to improve performance.

IO Offset	Mem Offset	Name	Description	
0x00	0 to 0x7ffc	**FIFO**	No byte swap	Writes 32-bit command FIFO. Reads 32-bit data output FIFO. Four address locations are provided to allow correction of byte order. Only word accesses are allowed. Writes while input FIFO is full or reads while output FIFO is empty set the *fifoerror* interrupt bit and generate an interrupt if enabled.
0x04	0x8000 to 0xfffc		swap B3↔B0 and B2↔B1	
0x08	0x10000 to 0x17ffc		swap B3↔B2 and B1↔B0	
0x0c	0x18000 to 0x1fffc		swap B3↔B1 and B2↔B0	
0x40	0x20040	**FIFOInFree**	Free entries in RISC input FIFO (b4:b0). Byte register. Read only.	
0x41	0x20041	**FIFOOutValid**	Valid entries (b2:b0) for RISC output FIFO. Byte register. Read only.	
0x42	0x20042	**Comm**	Communication register between host and RISC. Byte register. Read/write.	
0x43	0x20043	**MemEndian**	Select byte swapping on memory accesses. Byte register. Read/write.	
0x44	0x20044	**Intr**	Interrupt register. 16-bit register. Read/write.	
0x46	0x20046	**IntrEn**	Interrupt enable register. 16-bit register. Read/write.	
0x48	0x20048	**Debug**	Soft resets, RISC hold, and RISC single-step. Byte register. Read/write.	
0x49	0x20049	**LowWaterMark**	Input FIFO low water mark. Set ≥ 3 for DMA. Byte register. Read/ write.	
0x4a	0x2004a	**Status**	Status register. Specifies which blocks of the V2200 are busy.	
0x4b	0x2004b	**XBusCtl**	*XBus* control register. Read/write.	
0x4c	0x2004c	**PCITest**	PCI test register. Word register. Write only.	
0x50	0x20050	**DMACmdPtr**	DMA command list pointer. See *PCI Bus Master* and *Context Switching* sections. b31:b2 are read/write and b1:b0 are read-only (b0 is *DMABusy*).	
0x54	0x20054	**DMA_Address**	DMA data address. See *PCI Bus Master Section.* 32-bit read only.	
0x58	0x20058	**DMA_Length**	DMA remaining transfer count (b23:b2) and byte swap (b1:b0). See *PCI Bus Master* section. Read only.	
0x5c	0x2005c	**VGA_Extend**	VGA compatibility modes. Byte register. Read/write.	
0x5e	0x2005e	**MemBase**	Most significant byte of the PCI memory base address. Read only.	
0x60	0x20060	**StateIndex**	Select *StateData* to read. See *Context Switching* section. Byte read/write.	
0x64	0x20064	**StateData**	Read state from RISC or *Pixel Engine*. Writes update decode instruction register. RISC must be held during accesses. See *Context Switching* section.	
0x68	0x20068	**SClkPLL**	System Clock PLL control register. Read/write.	
0x70	0x20070	**Scratch**	16-bit BIOS scratch register. Read/write.	
0x72	0x20072	**Mode**	VGA and Vesa mode control. Read/write.	
0x73	0x20073	**Scratch1**	8-bit BIOS scratch register. Read/write.	
0x74	0x20074	**BankSelect**	Local memory to map to the PCI 64K byte memory block at 0xA0000 in bits {b31,b23:b13}. If bit 31 set, access BIOS ROM, else access memory. b23:b13 is address of 8K byte bank. 32-bit read/write. Other bits = 0.	
0x80	0x20080	**CRTCTest**	CRTC Test register. b4:b0 are read/write.	
0x84	0x20084	**CRTCCtl**	CRTC mode.	
0x88	0x20088	**CRTCHorz**	CRTC horizontal timing.	
0x8c	0x2008c	**CRTCVert**	CRTC vertical timing.	
0x90	0x20090	**FrameBaseB**	Stereoscopic frame B base address.	
0x94	0x20094	**FrameBaseA**	Frame A base address.	
0x98	0x20098	**CRTCOffset**	CRTC *StrideOffset*.	
0x9c	0x2009c	**CRTCStatus**	CRTC video scan position.	
0xa0 to 0xa8	0x200a0 ….	**MemCtl**	Memory timing and control registers.	
0xac	0x200ac	**CursorBase**	Cursor base address in *CursorBase[23:10]* (1024 byte aligned).	
0xb0 to 0xbf	0x200b0 ….	**Palette**	Access color palette. All accesses must be bytes.	
0xc0	0x200c0	**PClkPLL**	Pixel Clock PLL control register. Read/write.	
0xd0 to 0xdc	0x200d0 ….	**VideoInput**	Video Input control. See *Video Input* section.	
0xf0 to 0xff	0x200f0 ….	**Device1**	Access the device on the external byte-wide interface bus.	

Table 5.2 – Native IO registers

5.3 Communication Register

The Communication register (*Comm*) allows status exchange between host and the V2200 chip as shown in table 5.3.

bit	Name	Description
3:0	**SysStatus**	Four-bit field set by system processor to send status information to the RISC. Read/write.
7:4	**RISCStatus**	Four-bit field set by V2200 RISC processor by writing its *Flag* register. Read only.

Table 5.3 – Communication register bit definition

5.4 MemEndian Register

The *MemEndian* register controls memory byte swapping. Bit definitions are shown in table 5.4. Read/write.

Bit	Name	Description
1:0	**MemEndian**	Sets the endian-ness of direct PCI to local memory accesses (not IO). Cleared by *Reset*. **b1:b0 Endian Description** 00 No Swap (default) Use for word sized data accesses 01 B3↔B0, B2↔B1 Keeps byte data address same on x86 and V2200 10 B3↔B2, B1↔B0 11 B3↔B1, B2↔B0 Keeps half-word data address same on x86 and V2200
7	**DMABusy**	DMA Busy. Read only.

Table 5.4 – *MemEndian* register bit definition

5.5 Interrupt Register

The interrupt register (*Intr*) specifies status of interrupts from the V2200 chip as shown in table 5.5.

bit	Name	Description
0	**VertIntr**	Vertical retrace interrupt from V2200. Bit set and held at start of vertical-retrace, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
1	**FIFOLowIntr**	Interrupt host as the number of free entries in *InputFIFO* rises above the *LowWaterMark*. Bit asserted even if this interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
2	**RISCIntr**	RISC firmware initiated interrupt. Bit can be asserted by RISC even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
3	**HaltIntr**	RISC halt interrupt. Bit asserted when a halt is executed, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored)
4	**FIFOErrorIntr**	If enabled, a FIFO error interrupt occurs if the input FIFO is written when full, or the output FIFO is read when empty. Bit asserted on FIFO error even if interrupt is disabled.
5	**DMAErrorIntr**	If enabled, a DMA error interrupt occurs on a DMA error (e.g. PCI error). Bit asserted on DMA error even if interrupt is disabled.
6	**DMAIntr**	If enabled, a DMA complete interrupt occurs when the DMA is done. Asserted on completion even if interrupt is disabled.
7	**XIntr**	Externally initiated interrupt. Edge sensitive pass-through from *Xintr* input. Polarity of active edge is programmed by the *XBusCtl* register. Bit asserted even if this interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
8	**VideoInEvenIntr**	Video input even field interrupt. If video input is enabled, bit set and held at end of even field, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
9	**VideoInOddIntr**	Video input odd field interrupt. If video input is enabled, bit set and held at end of even field, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
10	**IdleIntr**	When enabled, interrupts host when chip becomes idle, even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).

11	**WriteIntr0**	Interrupt on write to memory byte specified by WriteIntr0Adr register (see *Video Input* section). Set even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).
12	**WriteIntr1**	Interrupt on write to memory byte specified by WriteIntr1Adr register (see *Video Input* section). Set even if interrupt is disabled. Reset by writing as a 1 (write =0 ignored).

Table 5.5 – Interrupt Register bit definition

5.6 Interrupt Enable Register

The Interrupt Enable register (*IntrEn*) enables interrupts from the V2200 chip as shown in table 5.6.

bit	Name	Description
0	**VertIntrEn**	Enable Vertical interrupts to host
1	**FIFOLowIntrEn**	Enable the input FIFO *LowWaterMark* interrupt to host
2	**RISCIntrEn**	Enable RISC processor interrupts to host
3	**HaltIntrEn**	Enable RISC halt interrupt to host
4	**FIFOErrorIntrEn**	Enable FIFO error interrupt to host
5	**DMAErrorIntrEn**	Enable DMA error interrupt to host
6	**DMAIntrEn**	Enable DMA complete interrupt to host
7	**XIntrEn**	Enable externally initiated interrupt from *XIntr* to host
8	**VideoInEvenIntrEn**	Enable video input even field interrupt to host
9	**VideoInOddIntrEn**	Enable video input odd field interrupt to host
10	**Idle**	Enable chip idle interrupt to host
11	**WriteIntr0En**	Enable the memory write 0 interrupt. Read/write.
12	**WriteIntr1En**	Enable the memory write 1 interrupt. Read/write.

Table 5.6 – Interrupt Enable (*IntrEn*) register bit definition

5.7 Debug Register

The *Debug* register is used to hold and single-step the RISC processor. This register can also be used to initialize the V2200 chip. All bits are read/write. Bit definitions are shown in table 5.7.

When the *HoldRISC* bit is written in the *Debug* Register, the hold does not take affect until the *Pixel Engine*, the *Triangle Engine*, and the *Fill Engine* are all idle (*PixelBusy*=0 in the *Flags* Register). At this time, the *RISCHeld* bit is set to 1, and an *Idle Interrupt*. After HoldRISC is asserted, no new instruction are allowed past the decode stage in the RISC pipeline (no-ops are inserted into the pipeline until *RISCHeld* = 1). This prevents the *PixelBusy* bit from staying on because new instructions start before the *HoldRISC* takes affect. A good side affect of this is that when *RISCHeld* is asserted and an *Idle Interrupt* is generated, it is guaranteed that no fills using the input FIFO are in progress. Therefore, it is safe to turn the DMA off, and continue with the context switch.

bit	Name	Description
0	**SoftReset**	If set, chip goes into soft reset. This bit is cleared by *Reset*. Read/write.
1	**HoldRISC**	Stop RISC processor when set. Acts as a hold signal only, does not alter any state. Note, the RISC will not stop until any potential hazards are cleared. This bit is set by *Reset*. Read/write.
2	**StepRISC**	Single-step RISC when written as a 1 (ignored if written as 0). Bit is cleared by RISC when step completes (e.g. on a cache miss, it completes when the miss is cleared). Cleared by *Reset*. Read/write.
3	**DebugScratch**	Bit for backward compatibility. Has no effect. Read/write.
4	**SoftVGAReset**	If set, assert VGA reset. Cleared by *Reset*. Read/write.
5	**SoftXReset**	If set, assert *XReset* chip output to reset external devices. Cleared by *Reset*. Read/write.

Table 5.7 – *Debug* register bit definition

5.8 Low Water Mark Register

The Low Water Mark register (*LowWaterMark*) specifies status of interrupts from the V2200 chip as shown in table 5.8.

bit	Name	Description
4:0	**LowWaterMark**	If the *FIFOLowIntrEn* bit is set, the host is interrupted as the number of free entries in the *InputFIFO* rises above this 5-bit value. Also specifies when the DMA starts the request for the next burst of data. For DMA must be ≥ 3.

Table 5.8 – Low Water Mark register bit definition

5.9 Status

The Status register (*Status*) specifies the V2200 busy status, as shown in table 5.9. *Idle* means that the V2200 is in a state that allows the host to "interrupt" the RISC (with *Debug*, *StateIndex* and *StateData* registers). Read only register.

bit	Name	Description
0	**RISCHalted**	RISCHalted
1	**RISCHeld**	RISCHeld. RISC has stopped due to assertion of *RISCHold* bit set in Debug register.
2	**PixelIdle**	*Pixel Engine* and *Triangle Engine* are idle.
3	**FillIdle**	*Fill Engine* is idle.
4	**InputFIFOStall**	RISC stalled for *InputFIFO*.
5	**OutputFIFOStall**	RISC stalled for *OutputFIFO*.
7	**Idle**	(*RISCHalted* || *RISCHeld* || *InputFIFOStall* || *OutputFIFOStall*) && *PixelIdle* && *FillIdle*

Table 5.9 – *Status* register bit definition

5.10 XBusCtl

The XBusCtl register controls the XBus mode as shown in table 5.10.

bit	Name	Description
0	**XBusWaitBlank**	If 0 set, do not start *XBus* accesses except during horizontal front porch or horizontal sync. Cleared by *Reset*. Read/write.
1	**XBusIntrPolarity**	If set, *XBusIntr* polarity is positive edge triggered, else it is negative edge triggered. Cleared by *Reset*. See the *Interrupt Register* section for more information. Read/write.
5:2	**DACTestCtl**	DAC test control bits. Write as 0 for normal operation. Initialized by *XA[3:0]* during *Reset*.
6	**XWaitEn**	1 = use *XWait* normally. 0 = use *XWait* as *VGASpeed* output. Cleared by *Reset*. Read/write.
7	**TestMode**	Test mode enable. Clear for normal operation. Note, *TestMode* must be off when using *XBus* or if *VOut* is enabled. Initialized by *~XD[4]* during *Reset*. Read/write.

Table 5.10 – *XBusCtl* register bit definition

5.11 VGA_Extend Register

The *VGA_Extend* register controls VGA compatibility modes as shown in table 5.11.

bit	Name	Description
2:0	**Compatibility**	These bits came with our VGA core and we weren't absolutely certain we could eliminate them, so here they are. Set to 4 by *Reset*. Do not modify. Read/write.
3	**Bandwidth**	Another historic VGA bit. Cleared by *Reset*. Do not modify. Read/write.
4	**FastMode**	VGA *FastMode*." Set this bit in VGA graphics modes for higher memory bandwidth. Must be cleared for VGA text modes. Cleared by Reset. Read/write.

Table 5.11 – *Mode* register bit definition

5.12 Mode Register

The *Mode* register controls VGA mode, VESA 2.0 mode and enables DMA as shown in table 5.12.

bit	Name	Description
0	**VESA_Mode**	If set, and in native mode, enable the 64K byte memory address space at 0xA0000. Cleared by *Reset*. Read/write
1	**VGA_Mode**	Set for VGA mode. Clear for native mode. Set by *Reset*. Read/write
2	**VGA_32**	Enable 32 bit write mode for VGA mode 13.
3	**DMA_En**	Enable DMA accesses. Cleared by *Reset*. Read/write.

Table 5.12 – *Mode* register bit definition

5.13 Memory Control Registers

Memory control (*MemCtl*) register is mapped to the V2200's PCI IO space and is accessed as a word as described in table 5.13a. The memory configuration specified by the *MemCtl* register is described in table 5.13b. The memory controller and external memory are clocked by a different clock (*MClk*) then other logic (*SClk*).

PCI offset	RISC Addr	Name	Description
0xA0	0xffe00500	MemCtl	Memory timing. Not affected by *SoftReset*. Read/write. Write unused bits as 0. **Bits Description** 0 *MemType* (1 = SGRAM, 0 = SDRAM). Initialized by *MD[33]* during *Reset*. 1 *MemLatency* (1 = 3 cycles). Initialized by *MD[35]* during *Reset*. 4:2 *User defined*. Initialized by *MD[40:38]* during *Reset*. 7:5 *MemSizeCS1*. Memory Size for chips controlled by *CS[1]*. Initialized by *MD[43:41]* during *Reset*. If CS0 is set=0, then 2nd Bank is never refreshed. See table 5.11b for legal values. 8 *MemBanksPerChip* (1 = 4, 0 = 2) . Initialized by *MD[44]* during *Reset*. 10:9 *MemSpeed* (3 = 125MHz, 2 = 100MHz, 1 = 83.33MHz, 0 = 66.67MHz). Initialized by *MD[30:29]* during *Reset*. 12:11 *MemAdrSwizzle*. Initialized by *MD[62:61]* during *Reset*. For native mode, set =0 (swizzle). For VGA mode set = 3 (linear) . 13 Hold memory refresh cycles until video is blanked. Set by *Reset*. 14 Hold all non-video accesses until video is blanked. Cleared by *Reset*. 15 If set, disable memory refresh cycles in MClk cycles. Cleared by *Reset*. 23:16 Write refresh period ([23:16] * 32 + 31). Reads current counter value.
0xA4	0xffe00520	MemDiag	Memory diagnostic register (includes some control bits). All bits cleared by *Reset*. **Bits Description** 0 Reserved. Write as 0. 1 If set, disable CRTC accesses. 2 If set, disable VGA accesses. 3 If set, disable RISC load accesses. 4 If set, disable Load FIFO accesses. 5 If set, disable Store Queue accesses. 6 If set, disable Instruction Cache accesses. 7 If set, disable Memory accesses. 8 If set, block fills take 1 cycle, else 2 cycles. 9 SlowRAS. If set, minimum RAS is 7 cycles, else it is 6 cycles. 10 FastPrecharge. If set, memory precharge is 2 cycles, else it is 3 cycles. 11 StopRefresh. If set, disable DRAM refresh, but still accumulate refresh requests (up to 32). 12 Positive Edge Data Read. Allows an extra ½ cycle read time. Needed at speed above 100MHz.
0xA8	0xffe00540	MemCmd	Memory diagnostic register.
0xAC	0xffe00560	CursorBase	Cursor base address. *CursorBase[23:10]* (aligned to 1024 byte boundary).

Table 5.13a – Memory control register

MemSizeCS1	*MemBanksPerChip*	Total Memory	Memory type	#Chips
0	0 (= 2 banks)	2 MB	128K x 2 x n	64 / n
0	0 (= 2 banks)	4 MB	256K x 2 x n	64 / n
0	0 (= 2 banks)	8 MB	512K x 2 x n	64 / n
0	0 (= 2 banks)	16 MB	1024K x 2 x n	64 / n
2	0 (= 2 banks)	4 MB	128K x 2 x n	128 / n
3	0 (= 2 banks)	8 MB	256K x 2 x n	128 / n
4	0 (= 2 banks)	16 MB	512K x 2 x n	128 / n

Table 5.13b – Memory Configuration

5.14 CRTC Registers

CRTC control registers are mapped to the V2200's PCI IO space. These registers are accessed as words and are described in table 5.14a, and 5.14b.

PCI offset	RISC Addr	Name	Description

0x84	0xffe00420	CRTCCtl	CRTC control register. Read/write. **Bits Description** 3:0 ScrnFmt (see *DstFmt*, table 4.5). Valid values are 1, 2, 4, 5, 6 and 12. Note, CRTC does logical right shift on *R*, *G* and *B* until LSB is in b0. 4 VideoFifoSize. If set, *VideoFifo* is 128 bytes, else it is 64 bytes. Set for best 2D performance. Clear for 3D at 640x480x16. Also see bit 23. 5 Enable DDC data output. Cleared by *Reset*. 6 DDC clock on the *DDCClk* pin. Writing a 0 drives *DDCClk* pin low. Writing a one drives *DDCClk* pin to high-Z. The value on the *DDCClk* pin is returned on reads. Cleared by *Reset*. 7 DDC data. If b5 = 1, *DDCData* pin = b7, else b7 reads *DDCData* pin. 8 VSyncHi. If set, *VSync* is active high. Note, *VSyncHi* is used in DDC mode to drive DDC1 clock output on *VSync*. Set by *Reset*. 9 HSyncHi. If set, *HSync* is active high. Set by *Reset*. 10 VSyncEnable. If set, *VSync* is enabled, else *VSync* is de-asserted. Cleared by *Reset*. 11 HSyncEnable. If set, *HSync* is enabled, else *HSync* is de-asserted. Cleared by *Reset*. 12 VideoEnable. If set, *Blank* is enabled, else Blank is asserted. Note, in VGA mode, is VideoEnable is asserted, native video is output. In this mode, the VGA CRT video must be disabled. 13 Stereoscopic video output frame-toggle enable. 14 FrameDisplayed. Cleared by FrameBaseA write. Set at end of vertical blank. Read only. 15 FrameBufferBGR. If set, Red is in LSBs of video, else Blue is. 16 EvenFrame. Read as set when even stereoscopic frame is displayed. 17 LineDouble. Set for low-res. (e.g. 320x240 with Vertical active = 239). 18 FrameSwitched. Cleared by write to FrameBaseA, except during vertical blank. Set during vertical blank. Read only. 19 Interlaced video mode. *FrameBaseA* is the frame base for even fields and *FrameBaseB* is used for odd fields. Note, the even field VSync is delayed as specified by the *EvenFieldVSyncOffset* (in *CRTCTest*) and the even field is lengthened by one scanline (e.g. for NTSC, each field is effectively 262.5 scanlines). 22:20 HBlankPad. Increase horizontal blank by this number of pixels. 23 If set, *VideoFifo* size is 256 bytes. Bit 23 over rides bit 4.
0x88	0xffe00440	CRTCHorz	CRTC horizontal timing. Writes take effect when *CRTCVert* is written. Until then, the old *CRTCHorz* is returned on reads. Not affected by *Reset*. Read/write. **Bits Description** 7:0 Horizontal active. b7:b0 = WidthInPixels / 8 – 1. 14:9 Horizontal back porch = WidthInPixels / 8 – 1. 20:16 Horizontal sync width = WidthInPixels / 8 – 1. 23:21 Horizontal front porch = WidthInPixels / 8 – 1.
0x8C	0xffe00460	CRTCVert	CRTC vertical timing parameters. Not affected by *Reset*. Read/write. **Bits Description** 10:0 Vertical active = HeightInScanlines – 1. 16:11 Vertical back porch = HeightInScanlines – 1. 19:17 Vertical sync width = HeightInScanlines – 1. 25:20 Vertical front porch = HeightInScanlines – 1.

Table 5.14a -- CRTC control registers

PCI offset	RISC Addr	Name	Description
0x90	0xffe00480	FrameBase B	Frame base B. Pixel aligned top left byte address of screen odd stereoscopic frame. Writes take effect in the CRTC when *FrameBaseA* is written. Until then, the old *FrameBaseB* is returned on read. In stereoscopic mode, *FrameBaseB*[2:0] must match *FrameBaseA*[2:0]. Not affected by *Reset*. Read/write.

0x94	0xffe004a0	FrameBase A	Frame base A. Pixel aligned, top left byte address of screen in non-stereoscopic mode or even stereoscopic frame. Not affected by *Reset*. Read/write.
0x98	0xffe004C0	CRTCOffset	CRTC *StrideOffset* (b15:b0). Byte offset between last *VideoFifo* memory request on one scanline to the first one on the next scanline. Video logical frame buffer width must be a multiple of 8 bytes (see *DstStride*). Width and size are in bytes. *StrideOffset = DstStride – ScreenWidth* + (*ScreenWidth* % *VideoFifoSize*) + ((*FrameBaseA[2:0]* || (*ScreenWidth* % *VideoFifoSize*)) ? 0 : *VideoFifoSize*);
0x9C	0xffe004e0	CRTCStatus	Current CRTC status information. Read/write. **bits Description** 7:0 "Character" clocks (8 pixels) left in current horizontal section. 10:9 Horizontal status (1 =front porch, 3 =sync, 2 =back porch, 0 =active). 21:11 Scan lines left in current vertical section. 23:22 Vertical status (1 =front porch, 3 =sync, 2 =back porch, 0 =active).
0x80	0xffe00400	CRTCTest	CRTC test register. Read/write. **Bits Description** 4:0 VideoLatency. Clocks between video FIFO requests. For testing only. Software should not modify these bits. Read/write. 15:5 EvenFieldVSyncOffset. Interlaced video even field *VSync* leading and trailing edge position before the end of active video on appropriate scanlines. Note, the even field is lengthened by one scanline (e.g. for NTSC, each field is effectively 262.5 scanlines). 16 not(Display Enable), (= horizontal or vertical blank). Read only. 18 Vertical blank. Read only.

Table 5.14b -- CRTC control registers

5.15 PLL Control Registers

There are two internal Phase-Locked-Loop (*PLL*) circuits. The first sets the pixel clock (*PClk*) frequency. The second sets both the system logic clock (*SClk*) and the memory controller clock (*MClk*). Both *SClk* and the *MClk* are generated by the same PLL, with the only difference being the output divider ratio. The V2200 requires a clock input frequency of 14.318 MHz. Following is the PLL frequency programming equation:

$$\text{Clock-out} = ((\ ClkIn\ /\ N\ )\ *\ M\ )\ /\ P$$

The VGA circuitry uses the PCLk when in VGA or test mode. In native mode, the VGA uses SClk/3 so that accesses to its registers are still honored.

When the digital video output (*VOut*) is enabled, an divide by 2 (*extraDiv2* enabled) is inserted after the *PClkPLL* "P" divider and before *PClk*. Note, the PClk PLL oscillator M, N, and P values must be modified accordingly. The output of the "P" divider is used as the 2X clock output (*VOutClk*) that shifts byte serial 16 bits/pixels digital video out on *XBus[7:0]*. The output from the extra divide by 2 is then used as the normal pixel clock. For example, when *VOut* is enabled and a 31.5 MHz pixel rate is needed, set the oscillator at 126 MHz, the "P" divide to 2, and set *extraDiv2*.

The *MClk* frequency must always be greater than or equal to the *SClk* frequency, so the synchronizers between the clock domains work properly. The phase compare frequency (ClkIn/N) must be in the range of 1.0 to 3.0 MHz. The voltage controlled oscillator range is 125 to 250 MHz. After changing either the M or N parameter, software must wait 200 *u*S for the PLL to stabilize before counting on the clock.

The PLL register definitions are shown in table 5.15a. Frequencies set for *SClkOut* and *MClkOut* after *Reset* are described in table 5.15b (see the *Reset* section for more information).

PCI offset	RISC Addr	Name	Description
0xC0	0xffe00600	PClkPLL	Video clock PLL control. At (hard) *Reset*, programming registers set to output 25.165 MHz. Not modified by *SoftReset*. Read/write. **Bits Description** 8:0 PClkM. Ratio of VCO frequency to phase compare frequency. 12:9 PClkP. Ratio of VCO frequency to output frequency. 18:13 PClkN. Ratio of *Clk* input frequency to phase compare frequency. 19 VOutEn. Enable VOut if set. See *Video Output* section. Asserted by *Reset*. 20 VGAStdClk. If set VGA uses normal clock, else uses PLL. Asserted by *Reset*. 21 ExtraDiv2. Set for Digital Video Out. An extra /2 is inserted before PClk, so byte serial output is twice the PClk frequency. Set by *Reset*. 22 DirectPClk. Use *DirectPClk* as input to "P" divider if set, else use PLL. During *Reset*, initialized to the value of *~XD[5]*. 23 PClkStartEn. If set, send a start pulse to *PClkPLL* when writing *PClkPLL*. 24 YUVEnable. If set VOut data is in CbYCrY video format. 25 VOutLSBFirst. If set, VOut data bits[7:0] precede bits[15:8]. 26 DoubleClock. If set, enables clock double circuit. Cleared by *Reset*.
0x68	0xffe00620	SClkPLL	System and memory clock PLL control. At (hard) *Reset*, initialized as specified by *XBus* and *Memory Data* pins (see *Reset* section). Not modified by *SoftReset*. For PCI, read/write. For RISC, read only. **Bits Description** 8:0 SClkM. Ratio of VCO frequency to phase compare frequency. 12:9 SClkP. Ratio of VCO frequency to *SClk* output frequency. 16:13 MClkP. Ratio of VCO frequency to *MClk* output frequency. 22:17 SClkN. Ratio of *Clk* input frequency to phase compare frequency. 23 DirectMClk. If set, use *DirectMClk* as input for both *MClk* and *SClk* "P" dividers, else use PLL. Takes precedence over *TestClkMode* for "P" divider. During *Reset*, initialized to the value of *~XD[5]*. 24 TestClkMode. If set, *SClk* uses *Xtal1* input and *MClk* uses *DirectMClk*. Set *MaxMClk*=66.67 and *MaxSClk*=90. See table 5.13a and *Reset* section. 25 SClkStart. Cycle 0 to 1 to 0 to send a start pulse to *SClkPLL*.

Table 5.15a – PLL control registers

Max Mclk	Max SClk	PLL M	PLL N	OSC freq	Phase freq	MClk P	SClk P	MClk freq	SClk freq
66.7	50.0	70	5	200.45	2.86	3	4	66.8	50.1
66.7	55.6	70	5	200.45	2.86	3	4	66.8	50.1
66.7	60.0	70	5	200.45	2.86	3	4	66.8	50.1
66.7	66.7	70	5	200.45	2.86	3	3	66.8	66.8
66.7	75.0	70	5	200.45	2.86	3	3	66.8	66.8
66.7	83.3	70	5	200.45	2.86	3	3	66.8	66.8
66.7	90.0	n/a	n/a	n/a	n/a	1	1	for test	for test
83.3	50.0	87	5	249.13	2.86	3	5	83.0	49.8
83.3	55.6	58	5	166.09	2.86	2	3	83.0	55.4
83.3	60.0	84	5	240.54	2.86	3	4	80.2	60.1
83.3	66.7	87	5	249.13	2.86	3	4	83.0	62.3
83.3	75.0	52	5	148.91	2.86	2	2	74.5	74.5
83.3	83.3	58	5	166.09	2.86	2	2	83.0	83.0
83.3	90.0	58	5	166.09	2.86	2	2	83.0	83.0
100.0	50.0	70	5	200.45	2.86	2	4	100.2	50.1
100.0	55.6	70	5	200.45	2.86	2	4	100.2	50.1
100.0	60.0	63	5	180.41	2.86	2	3	90.2	60.1
100.0	66.7	70	5	200.45	2.86	2	3	100.2	66.8
100.0	75.0	70	5	200.45	2.86	2	3	100.2	66.8
100.0	83.3	70	5	200.45	2.86	2	3	100.2	66.8
100.0	90.0	63	5	180.41	2.86	2	2	90.2	90.2
125.0	50.0	87	5	249.13	2.86	2	5	124.6	49.8
125.0	55.6	77	5	220.50	2.86	2	4	110.2	55.1
125.0	60.0	84	5	240.54	2.86	2	4	120.3	60.1
125.0	66.7	87	5	249.13	2.86	2	4	124.6	62.3
125.0	75.0	78	5	223.36	2.86	2	3	111.7	74.5
125.0	83.3	87	5	249.13	2.86	2	3	124.6	83.0
125.0	90.0	87	5	249.13	2.86	2	3	124.6	83.0

Table 5.15b – Default *SClkOut* and *MClkOut* frequencies after *Reset* (see *Reset* section)

6. PCI Bus Master

The DMA controller is a PCI bus master that executes DMA operations as specified by a chained DMA command list (*DMACmdList*). A DMA operation is initiated by writing the address of the first *DMACmdList* to the DMA command pointer (*DMACmdPtr*) register. The *DMACmdPtr* holds the physical byte address of current DMA command pair (*DMACmd*). A *DMACmd* must be double-word (32-bit words) aligned and consists of a DMA Length (*DMALength*), in the odd word, and a DMA Address (*DMAAddress*), in the even word. The *DMALength* specifies the size of the DMA transfer or that the *DMAAddress* is a link to the next *DMACmdList*. The *DMAAddress* is the word-aligned physical byte address of the data array to DMA or a link address pointing to the next *DMACmdList*. A null *DMACmdList* pointer is used to terminate DMA.

Each *DMACmdList* must fit in a 16K byte aligned 16K byte block of memory and must be double-word aligned. The *DMABusy* bit is set immediately when the *DMACmdList* pointer is written with a non-null pointer and remains asserted until the DMA has completed or stopped. DMA can be stopped by writing *DMA_En* = 0 in *Mode* register. When the DMA has completed, the *DMACmdPtr* will point to the address of the last *DMACmd*. The host can determine when DMA has completed by testing the *DMABusy* bit in the *DMACmdPtr* register or the *MemEndian* register, or by using the *DMAIntr*. Note, for DMA operations, the *LowWaterMark* must be set ≥ 3. The *DMALength* definition is shown in table 6 and the DMA flow diagram is shown in figure 6.

Name	Description
Length	DMA length. Maximum DMA length is 16 M bytes (b30:b24 is ignored) **bits Description** 31 If set, then *DMAAddress* points to a new command list. 23:2 Length of DMA transfer in words (Ignored if b31 is set). 1:0 Byte swap (Ignored if b31 is set). **Value Endian Swap Description** 0 No Swap Use for word data. 1 B3↔B0, B2↔B1 Use for byte data. 2 B3↔B2, B1↔B0. 3 B3↔B1, B2↔B0 Use for half-word data.

Table 6 -- *DMALength* description

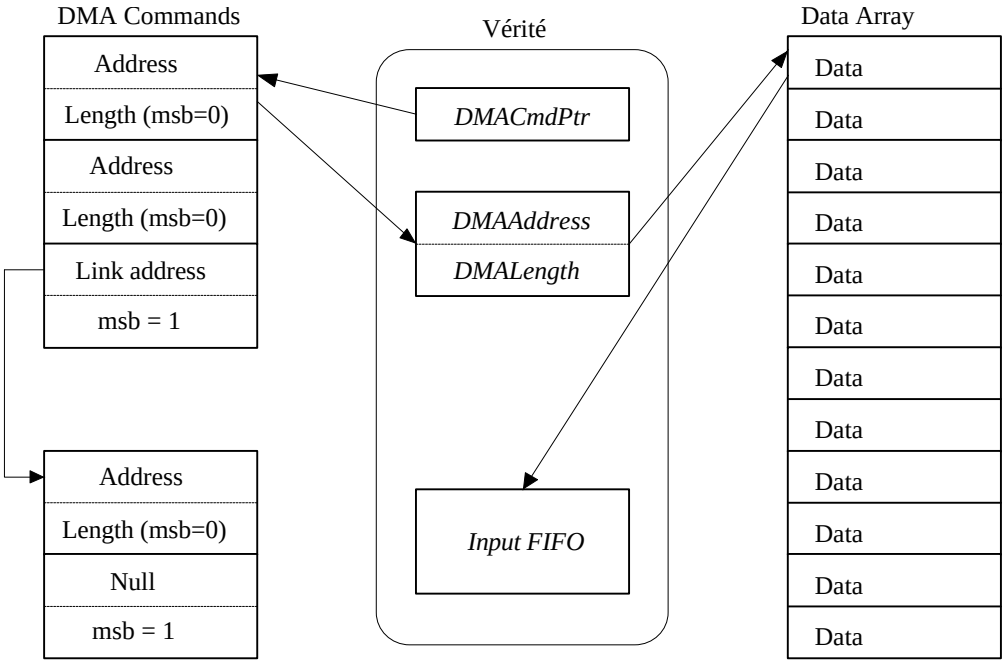


Figure 6 -- DMA flow diagram

7. Palette and Cursor

The palette and cursor are integrated into the V2200. Register definitions are shown in table 7a and table 7b.

VGA	Offset	RISC	Name	Description
0x3C8	0xb0	0xffe00580	Palette Write Index	Current color palette index to write.
0x3C9	0xb1	0xffe00588	Palette Data	Palette read/write data.
0x3C6	0xb2	0xffe00590	Palette Mask	Pixel mask register.
0x3C7	n/a	n/a	(Read) DACState	Last palette operation (3=read, 0 = write).
0x3C7	n/a	n/a	(Write) Palette Read Index	Current color palette index to read.
n/a	0xb3	0xffe00598	Palette Read Index	Current color palette index to read.
n/a	0xb4	0xffe005a0	Cursor Color Write Index	Current cursor color index to write.
n/a	0xb5	0xffe005a8	Cursor Color Data	Cursor palette read/write data.
n/a	0xb6	0xffe005b0	Command Register 0	Palette resolution and power-down.
n/a	0xb7	0xffe005b8	Cursor Color Read Index	Current cursor color index to read.
n/a	0xb8	0xffe005c0	Command Register 1	Palette bypass control.
n/a	0xb9	0xffe005c8	Command Register 2	Cursor, indexing and interlace control.
n/a	0xba	0xffe005d0	Command Register 4	Signature analysis register, when enabled.
n/a	0xbc	0xffe005e0	Cursor X-Low Register (CXLR)	Low 8 bits of cursor X position.
n/a	0xbd	0xffe005e8	Cursor X-High Register (CXHR)	Upper 4 bits of cursor X position.
n/a	0xbe	0xffe005f0	Cursor Y-Low Register (CYLR)	Low 8 bits of cursor Y position.
n/a	0xbf	0xffe005f8	Cursor Y-High Register (CYHR)	Upper 4 bits of cursor Y position.

Table 7a – Palette register definition

VGA IO	IO offset	Name	Description
n/a	0xb6	Command0	Palette command register 0. Cleared by *Reset*. Read/write. **Bits Description** 0 If set, power down mode, else normal operation. 1 Palette output resolution. If set, 8 bits/color, else 6 bits/color. 7 Enable access to extended registers.
n/a	0xb8	Command1	Palette command register 1. Cleared by *Reset*. Read/write. **Bits Description** 6:5 Pixel mode. 0 = 32 bpp, 1 = (2*) 16 bpp, 2 = (4*) 8 bpp 4 Bypass palette. If set, bypass palette, else use palette.
n/a	0xb9	Command2	Palette command register 2. Cleared by *Reset*. Read/write. **Bits Description** 1:0 Cursor mode. 0 = disable cursor, 1 = 3 color mode, 2 = 2 color highlight mode, 3 = 2 color X windows mode. 2 Palette indexing. If set use contiguous indexing, else sparse.
n/a	0xba	Status	Palette Status. Cleared by *Reset*. Read/write. **Bits Description** 7:6 Identification bits. Reads as 0. Read only. 5:4 Revision bits. 2 = External, 3 = Internal. 3 Monitor sense bit. Not implemented. 2 Read/write status. 0 = Write. 1 = Read. 1:0 Addr[a, b]. 0 = Red, 1 = Green, 2 = Blue, color component.
n/a	0xbb	VGAStatus	**Bits Description** 1:0 RdWrOp. Reads = 0 if previous palette access was a write. Reads as = 1 if previous palette access was a read. Read only.
n/a	0xbc	CXLR	Hardware cursor position in X, low 8 bits (CX[7:0]) in CXLR[7:0].
n/a	0xbd	CXHR	Hardware cursor position in X, upper 4 bits (CX[11:8]) in CXHR[3:0].
n/a	0xbe	CYLR	Hardware cursor position in Y, low 8 bits (CY[7:0]) in CYLR[7:0].
n/a	0xbf	CYHR	Hardware cursor position in Y, upper 4 bits (CY[11:8]) in CYHR[3:0].

Table 7b– Palette register bit definition

For compatibility with the V1000, most registers for controlling the V2200's built-in Palette are the same as the V1000 reference design. Register bits that have no function in the V2200 have been eliminated. Palette and cursor registers must be accessed as bytes. The cursor is kept in local memory and is fixed in size at 64 x 64. The 64 scan-lines for the cursor are in 128 contiguous double-word locations starting at *CursorBase* (aligned to 1024 byte boundary) register in the *MemCtl* block. Each double word defines one bit plane for one scan line of the cursor. The cursor planes are interleaved. The bottom scan line of the cursor has plane 0 in the double word at byte offset 0 and plane 1 at byte offset 8. The top scan line of the cursor has plane 0 in the double word at byte offset 1008 and plane 1 at byte offset 1016. Bit 63 of each double-word is the left most bit of its respective plane. The cursor position registers can be written at any time but do not take effect as a set until 0xbf (CYHR) is written. Once a frame has started, the cursor position at the beginning of the frame is used. A cursor position of (CX, CY) = (0, 0) places the cursor off-screen. A position of (1, 1) places the bottom right-most pixel of the cursor in the top left corner pixel of the display.

Palette write performance can be increased by using the word write feature. The starting palette index is written as usual for byte writes to the palette. Then, the Red, Green and Blue components are written as a word. The data format is Red[23:16], Green[15:8], Blue[7:0]. The palette index is auto-incremented, to the next color triple, after each write. This allows the palette to be written in 256 write operations. Note, reads from the palette must still be done a byte at a time.

8. Video Input

The Video input port (Vin) is used to write a byte serial video stream into memory. Video input port data will generally be in a byte serial 4:2:2 CbYCrY format, but other formats can be supported. The host can also write video streams directly to local memory.

The video input port can be treated as a single frame, or each field can be processed independently. The even and odd fields can both be continuously updated, or just the even fields can be captured. Even and odd fields have independent base addresses, although both can point to the same address if desired. Both fields share a single horizontal stride.

An interrupt can be independently generated at the end of each field. The interrupt can be sent to the host CPU or to the RISC processor. If it is sent to the host, the host interrupt service routine must cooperate with the device driver to launch the appropriate off-screen to screen scaled YUV to RGB conversion. If it is sent to the RISC, the RISC must have the appropriate information such as the video position and clip list. See the *Interrupt Register* and the *RISC Interrupt Register* sections for more information on interrupts.

The leading edge of *VInVSync* occurs while *VInHSync* is active during odd frames and while *VInHSync* is inactive during even frames. *VInQSize* should be set to two-entries for NTSC and PAL video input (approximately 28 to 35 MHz). Note that store queue entries reserved for video reduce the number available for normal operations and therefore reduce performance. Video input is scissored to *VInStride* bytes per scan line and *VInMaxVert* scan lines per field, to prevent corruption of memory outside the allocated range.

The Vertical Blanking Interval data standard (VBI) can be supported by the V2200. For this mode, *VInEnable* must be set appropriately, both *VInVBlank* and *VInActive* must be asserted during the VBI data periods, and *VInMaxVert* must be set large enough to hold both VBI and video data. The hardware automatically sets the stride for VBI data scan lines to twice the programmed stride, to account for the extra data bytes in each VBI scan line. This doubles the bytes between the base pointer and the first active video pixel. Note, since video data is usually sent as CbY0CrY1 order, it is simpler to treat the video as a YCbYCr surface (*VInOrder* = 1 with the *SrcMode* register *SrcBGR* set while doing YUV to RGB conversion). While this is not required, some video encoder chips can send *VBI* data as byte 1, byte 0, byte 3, byte 2. This places the *VBI* bytes in the correct order in memory.

The host can also input video by writing to local memory. In order to determine when this host video frame is completed, there are two memory write interrupts. These can be used for two independent video input streams or to specify each field of a single video input stream. Generally, the write-interrupt byte-address is set to the last byte of the appropriate field of video. When the RISC or host writes to the respective byte location, *riscintr* is asserted, if enabled. The RISC interrupt service routine then deals with getting the video onto the active screen.

Table 8a specifies the video input control register addresses and tables 8b and 8c specify bit definitions.

PCI offset	RISC Addr	Name	Description
0xd0	0xffe00680	VInEvenBase	Video input even field base address, double word aligned, top left memory byte address. Also controls the video input stride. Read/write.
0xd4	0xffe006a0	VInOddBase	Video input odd field base address, double word aligned, top left memory byte address. Also controls the video input mode. Read/write.
0xd8	0xffe006c0	WriteIntr0Adr	Memory write interrupt 0 address[23:3]. A write to local memory at this double-word address generates a *WriteIntr0* interrupt, if enabled.
0xdc	0xffe006e0	WriteIntr1Adr	Memory write interrupt 1 address[23:3]. A write to local memory at this double-word address generates a *WriteIntr1* interrupt, if enabled.

Table 8a – Video input control registers

bits	Name	Description
20:0	**VInEvenBase**	Video input even field base address, double word aligned, top left memory byte address / 8.
26:24	**VInMaxVert**	Video Input maximum scan-lines copied to memory per field. **Value MaxVert Value MaxVert** 0 240 4 540 1 263 5 590 2 287 6 720 3 313 7 788
29:27	**VInStride**	Video Input frame-width stride in bytes. **Value Distance Value Distance** 0 1280 bytes 4 2560 bytes 1 1536 bytes 5 3072 bytes 2 1600 bytes 6 3200 bytes 3 2048 bytes 7 4096 bytes
31:30	**VInQSize**	Entries to reserve in store queue for Video Input. Entries reserved only if *VInEnable* is set. Set to two entries for NTSC and PAL video input. **Value Distance Value Distance** 0 1 entry 2 4 entries 1 2 entries 3 6 entries

Table 8b – Video input *VInEvenBase* register bit definition

bits	Name	Description
20:0	**VInOddBase**	Video input odd field base address, double word aligned, top left memory byte address / 8.
25:24	**VInOrder**	Video Input byte order. Convert input byte order into appropriate byte order for each word. Must be set = 0 for VMI. **Value | First Second Third Forth** ⇐ Bytes in input stream 0 | 3 0 1 2 ⇐ Output bytes 1 | 1 0 3 2 ⇐ Output bytes 2 | 0 1 2 3 ⇐ Output bytes 3 | 0 3 2 1 ⇐ Output bytes
26	**VInHSyncHi**	If set, VInHSync is active high, else it is active low. Must be set = 0 for VMI.
27	**VInVSyncHi**	If set, VInVSync is active high, else it is active low. Must be set = 1 for VMI.
28	**VInActiveHi**	If set, VInActive is active high, else it is active low. Must be set = 1 for VMI.
29	**VInNoOdd**	If set, only even fields are copied to memory.
31:30	**VInEnable**	Video input enable. Cleared by *Reset*. If VBI data is enabled, samples active bytes of every scanline of enabled fields. Field starts when *VInVSync* becomes inactive. Note, the scanline address is updated every scanline, even if there is no active data. In addition, the stride during *VInVBlank* is twice the programmed stride. **Value Description** 0 Video in off. Default after *Reset*. 1 Video in enabled. No VBI data. 2 Video in enabled. VBI data enabled. *VInVBlank* is active low. 3 Video in enabled. VBI data enabled. *VInVBlank* is active high.

Table 8c – Video input *VInOddBase* register bit definition

9. Video Output

Digital Video Out (*VOut*) is supported using the external data bus (*XD[7:0]*) for byte serial RGB video data output using *VOutClk*, *VOutActive*, *VOutHSync*, *VOutVSync* for control. *VOut* is enabled and controlled by the by *VOutEn* bit of the *CRTCCtl* register. *VOut* is byte-serial 16-bit per pixel 565 RGB with most significant byte first. The *PLL Control Registers* section for information on programming the *PClkPLL* in this mode.

Analog video, including *HSync* and *VSync*, work as normal when *Digital Video Out* is active, unless disabled using *VideoEnable*, *HSyncEnable* or *VSyncEnable*, respectively. When enabled, the timing for these signals match the respective signals of *VOut*. *VOutHSync*, *VOutVSync* polarity matches HSync and VSync, respectively. *VOutActive* is active-high. All three signals are always enabled when *VOutEn* is asserted. If *Digital Video Out* is enabled and the *XBusWaitBlank* bit in the *XBusCtl* register is set, PCI accesses to the *XBus* are held until horizontal front porch or horizontal sync is active.

10. Context Switching

Before switching contexts, the following procedure allows the current native mode state to be determined:

- Hold the RISC processor using the *Debug* register.
- Wait for the V2200 to become idle, either by polling or using *IdleIntr*.
- Stop DMA by writing *DMA_En* = 0 in the *Mode* register.
- Save RISC state.
- While DMA is busy or *InputFIFO* not empty, read and save contents of *InputFIFO*.
- Read *DMACmdPtr*, *DMA_Address* and *DMA_Count*.
- Read *OutputFIFO* while not empty.
- Read other state (e.g. *Pixel Engine*, etc).
- If switching to VGA mode, memory that will be destroyed by VGA accesses should be written to memory reserved for the screen in native mode, and the hardware cursor should be turned off.
- Right before writing the *Mode* register to switch to VGA mode, read the *CRTCCtl* register to clear the pipeline.

Table 10 specifies *StateData* read back information. Pixel engine state can also be read back by host or RISC at offsets shown. See the *PCI I/O Registers and Memory Interface* section for RISC address. Note, data format is as specified in the *Pixel Engine* section, except as noted. Unused bits read as 0. Reads from unused indexes return unspecified results.

StateIndex	RISC Offset	StateData	StateIndex	RISC Offset	StateData
0	0	**SrcBase**	24	192	**YUVContrast**
1	8	**SrcMode**	26	208	**ChromaMask**
2	16	**SrcMask**	28	224	**Pick**
3	24	**Stride**	29	232	**AlphaThres**
4	32	**DstBase**	30	240	**SpecColor**
5	40	**DstMode**	32 - 39	256 - 312	**Tex4Pal_01, …, Tex4Pal_ef**
6	48	**DstFmt**	112	896	**X** (b31:b16 = Pixel Engine X)
7	56	**PMask**	113	904	**Y** (b31:b16)
8	64	**Pat**	114	912	**Z** (b27 = ZOvf, b26:b11 = Z)
12	96	**PatMode**	117	936	**F** (b23:b16)
13	104	**PatOffset**	119	952	**U** (b26:16 = U, b15:b8 = UFrac)
14	112	**ScissorX**	121	968	**V** (b26:16 = V, b15:b8 = VFrac)
15	120	**ScissorY**	123	984	**UDir** (b31)
16	128	**ZBase**	125	1000	**VDir** (b31)
17	136	**DitherOffset**	128	n/a	**Decode IR**
19	152	**FGColor**	129	n/a	**Decode PC**
20	160	**BGColor**	130	n/a	**Decode S1**
21	168	**FogColor**	131	1048	**DrawCtl** (b15:b0)
22	176	**DstColor**	132	1056	**XShadow** (b31:16 = X, b0 = XDir)
23	184	**ChromaColor**	133	1064	**Count** (b31:b16)

Table 10 – *StateData* read back description

11. VGA

For compatibility with old applications, the V2200 chip has hardware VGA support for basic VGA text and graphics modes through mode 13 and mode X. The BIOS supports enhanced VESA 2.0 modes using V2200 native mode. At PCI configuration time, the V2200 can respond as a PCI VGA display device, a PCI Multimedia device or as a PCI Other-Display device. The state of *XD[7:6]* during *Reset* determines which (see the *Reset* section). The chip is placed in VGA color mode after *Reset* if it is a VGA device, otherwise it is placed in native mode. Note, at address 0x50 in PCI configuration space, bit 0 specifies *PCI_2.1_Compliant* (cleared by *Reset*) and the other 7 bits are available as scratch. When cleared, the *PCI_2.1_Compliant* bit prevents PCI retries by holding the bus until VGA (only) access is completed.

To switch to native mode, or back to VGA mode, the *VGA_Mode* bit is written as specified in the *Mode* register description. When in VGA mode the V2200 responds to VGA I/O and memory addresses and functions as a VGA. VGA registers retain state, while in native mode. In addition, VGA registers can be accessed while in native mode, if the *VESA_Mode* bit is set in the *Mode* register. In addition, native mode registers and memory accesses function correctly while in VGA mode. Note, in VESA mode, the Video Blank status bits (0x3DA) correctly return the native CRTC blank status information. In native mode, when VESA mode is disabled, the V2200 does not respond to VGA memory or IO accesses. See the *Context Switching* section for information on switching between native and VGA modes. VESA 2.0 compatibility is supported in native mode by enabling local memory bank switching at 0xA0000 by setting the VESA mode bit in the Mode register. 64K bytes of local memory are mapped on 8K byte boundaries using *BankSelect*. Tables 11a through 11g specify VGA register definitions. I/O addresses with a "?" change depending on mode. In mono mode, the "?" becomes a "b" and in color mode it becomes a "d" (e.g. 0x3?a is 0x3ba for mono and 0x3da for color).

Read	Write	Name	bits	General Register Description
0x3cc	0x3c2	Miscellaneous		After *Reset*, this register = 0x01 (color mode).
			7	Vertical Sync Polarity.
			6	Horizontal Sync Polarity.
			5	Page bit for odd/even. If set, selects high page of memory.
			4:3	Unused.
			2	Clock select (0=25.18, 1=28.33 MHz). Only applies if *VGAStdClk* is set (in *PClkPLL* register) and using internal PLL *PClk*.
			1	Video RAM enable. If set, memory enabled.
			0	I/O address select. If set, in color mode (uses 0x3dX ports).
0x3c2	n/a	Input Status 0		Not implemented.
0x3?a	n/a	Input Status 1	5:4	Diagnostic bits. See the color plane enable register description.
			3	Vertical blank status. Set if in vertical blank.
			0	Display Enable. Set if in vertical or horizontal blank.
0x3ca	0x3?a	Feature Control	3	Vertical sync select. Should always be set = 0.

Table 11a – VGA General Registers

Port	Index	Name	bits	Sequencer Register Description
0x3c4	n/a	Sequencer index	2:0	Index selects which Sequencer register is accessed at 0x3c5.
0x3c5	0	Reset	1	Synchronous Reset. If 0, synchronously clear and halt Sequencer.
			0	Asynchronous Reset. If 0, asynchronously clear and halt Sequencer.
0x3c5	1	Clocking mode	5	Screen off. If set, video screen is turned off.
			4	Shift4. If set, internal shift register loads every forth character clock.
			3	Dot clock. If set, use VClk/2, else use VClk directly.
			2	Shift2. If set, internal shift register loads every second character clock.
			0	8/9 dot clock. Is set, use 8 wide characters, else use 9 wide.
0x3c5	2	Map mask	3	Map 3 enable. Enable write to memory map 3.
			2	Map 2 enable. Enable write to memory map 2.
			1	Map 1 enable. Enable write to memory map 1.
			0	Map 0 enable. Enable write to memory map 0.
0x3c5	3	Character map select	5	Character map select high bit A.
			4	Character map select high bit B.
			3:2	Character map select A.
			1:0	Character map select B.
0x3c5	4	Memory mode	3	Chain 4. If set, address 1:0 selects map to access.
			2	Odd/even. If set, access data within a map sequentially.
			1	Extended memory. If set, can access 256K bytes, else 64K bytes.

Table 11b– VGA Sequencer registers

Port	Index	Name	bits	DAC Register Description
0x3c6	n/a	Palette mask	7:0	Enable respective bits of video pixel data to color map.
0x3c7	n/a	Palette read index	7:0	Current index to read from palette.
0x3c8	n/a	Palette write index	7:0	Current index to write to palette.
0x3c9	n/a	Palette data	7:0	Palette read/write data.

Table 11c– VGA DAC registers

Port	Index	Name	bits	CRTC Register Description
0x3?4	n/a	CRTC Index	5	Test bit. Must be 0.
			4:0	Index selects which CRTC register is accessed at 0x3?5.
0x3?5	0x0	H total	7:0	Total horizontal characters – 5.
0x3?5	0x1	H display end	7:0	Total displayed characters – 1.
0x3?5	0x2	H start blanking	7:0	Start horizontal blanking when internal count equals this value.
0x3?5	0x3	H end blanking	7	Test bit.
			6:5	Display enable skew control.
			4:0	End horizontal blanking value. Equals count at end of blank.
0x3?5	0x4	H start retrace	7:0	Start horizontal retrace at this character position.
0x3?5	0x5	H end retrace	7	End horizontal blanking value, bit 5.
			6:5	Horizontal retrace delay by this amount.
			4:0	End horizontal retrace. Equals count at end of retrace.
0x3?5	0x6	V total	7:0	Total vertical scan lines – 2
0x3?5	0x7	Overflow	7	Start vertical retrace, bit 9.
			6	Vertical display enable end, bit 9.
			5	Vertical total, bit 9.
			4	Line compare, bit 8.
			3	Start vertical blank, bit 8.
			2	Start vertical retrace, bit 8.
			1	Vertical display enable end, bit 8.
			0	Vertical total, bit 8.
0x3?5	0x8	Preset row scan	6:5	Byte panning control.
			4:0	Starting row scan count after a vertical retrace.
0x3?5	0x9	Max scan line	7	200 to 400 line conversion. Doubles each scan line.
			6	Line compare bit 9.
			5	Start vertical blank, bit 9.
			4:0	Maximum scan line. Number of scan lines per character row – 1.

0x3?5	0xa	Cursor start	5	Cursor off. If set, cursor is off, else on.
			4:0	Cursor start. Specifies the character scan line to start cursor.
0x3?5	0xb	Cursor end	6:5	Cursor skew control. Delay cursor by this number of clocks.
			4:0	Cursor end. Specifies the character scan line to end cursor.
0x3?5	0xc	Start address high	7:0	First address to display after vertical retrace. High 8 bits.
0x3?5	0xd	Start address low	7:0	First address to display after vertical retrace. Low 8 bits.
0x3?5	0xe	Cursor location high	7:0	Cursor location. High 8 bits.
0x3?5	0xf	Cursor location low	7:0	Cursor location. Low 8 bits.
0x3?5	0x10	V start retrace	7:0	Vertical retrace start position. Low 8 bits.
0x3?5	0x11	V end retrace	7	Protect R0-R7. If set, disable writes to CRTC registers 0 to 7.
			6	Select 5 refresh cycles. If set, only 5 refresh cycles per scan line.
			5:4	Not implemented on V2200.
			3:0	End vertical retrace. Scan line count to end vertical retrace.
0x3?5	0x12	V display end	7:0	Vertical display end – 1. Low 8 bits.
0x3?5	0x13	Offset	7:0	Logical line width of screen.
0x3?5	0x14	Underline location	6	Double word mode. If set, enable double word mode.
			5	Count by 4. If set, memory address counter is character clock / 4.
			4:0	Character scan line to display underline.
0x3?5	0x15	V start blank	7:0	Start vertical blanking – 1. Low 8 bits.
0x3?5	0x16	V end blank	7:0	End vertical blanking.
0x3?5	0x17	CRTC mode	7	Hardware reset. If 0, clear horizontal and vertical retrace.
			6	Word/byte mode. If 0, shift all memory address counts down 1 bit.
			5	Address wrap. Selects memory address count MA13 or MA15.
			3	Count by 2. If set, memory count uses character clock / 2.
			2	Horizontal retrace select. If set, use horizontal retrace / 2 for V timing.
			1	Select row scan counter. If set use MA14 as CRT memory MA14.
			0	CMSO. If set, use MA13 as CRT memory MA13.
0x3?5	0x18	Line compare	7:0	Line compare target. Low 8 bits.

Table 11d– VGA CRTC registers

Port	Index	Name	bits	Graphics Register Description
0x3ce	n/a	Graphics index	3:0	Index selects which Graphics register is accessed at 0x3cf
0x3cf	0	Set/reset	3:0	Value to write to respective maps if set/reset mode is enabled.
0x3cf	1	Enable set/reset	3:0	Set/reset enable for respective maps.
0x3cf	2	Color compare	3:0	Color compare value for respective maps.
0x3cf	3	Data rotate	4:3	ALU function select (0: write directly, 1: AND, 2: OR, 3: XOR).
			2:0	Rotate right distance.
0x3cf	4	Read map select	1:0	Select which map to read from.
0x3cf	5	Graphics mode	6	256 color mode.
			5	Shift register mode.
			4	Odd/even mode.
			3	Read type. If 0, read selected map, else read color compare register.
			1:0	Write mode.
0x3cf	6	Miscellaneous	3:2	CPU address space and size (0: 0xa0000 for 128K, 1: 0xa0000 for 64K, 2: 0xb0000 for 32K, 3: 0xb8000 for 32K).
			1	Memory map 1. If set, host address bit 0 is replaced by another bit. The odd/even maps are selected by host address bit0.
			0	Memory map 0. If set, enable graphics mode.
0x3cf	7	Color compare mask	3:0	Enable color compare for respective memory map.
0x3cf	8	Bit mask	7:0	Enable writes to respective bits.

Table 11e– VGA Graphics registers

Port	Index	Name	bits	Attribute Register Description
0x3c0	n/a	Attribute index	5	Palette address source. =1 for normal operation. =0 for palette access.
			4:0	Index selects Graphics register to read at 0x3c1 and write at 0x3c0.
**	0 - 15	Palette	5:0	Map 4-bit color to 64 entry color map.
**	0x10	Attribute mode	7	P5/P4 select. If set, use Palette register bits 5:4 to send to external color palette bits 5:4, else use bits 1:0 of color select register.
			6	Pixel depth. If set, 8 bits/pixel. Clear bit for other modes.
			5	Pixel panning compatibility. If set, CRTC line compare assumes panning register =0. For panning one window of split screen.
			3	If set, enable blinking, else ninth dot same index as background.
			2	Enable line graphics character code
			1	Mono mode. If set, use mono mode, else use color mode.
			0	Graphics mode. If set, use color mode, else use mono mode.
**	0x11	Overscan color	7:0	Index to use as border color.
**	0x12	Color plane enable	5:4	Video status mux. Which plane bits appear in input status register 1 bits 5:4 (0: P2,P0, 1: P5,P4, 2: P3,P1, 3: P7,P6).
			3:0	Enable respective display memory color plane.
**	0x13	Horizontal panning	3:0	Number of pixels to shift video left.
**	0x14	Color select	3:2	If enabled, used as bits 7:6 going to external color palette.
			1:0	If enabled, used as bits 5:4 going to external color palette.

Table 11f– VGA Attribute registers

VGA port	Name	bits	Extended Register Description
0x3de	MemBase		Native-mode PCI Memory base address[31:24]. Byte register. Read only.
0x3df	IOBase		Native-mode PCI IO base address[15:8]. Byte register. Read only.

Table 11g– VGA Extended registers

** Attribute registers are written at 0x3c0 and read at 0x3c1.

12. Reset

The *Reset* input signal must be held for 100 mS to reset the V2200. The *SoftReset* (*SoftReset* bit in *Debug* register asserted and released), must be held for 1 mS. After *Reset* (or *SoftReset*, unless noted), the following occurs:

- *XBus[7:0]* and *XA[1:0]* are read during *Reset* (but not *SoftReset*) to initialize the V2200.
- Memory controller is initialized and memory refresh is started after *Reset* (but not *SoftReset*).
- Internal state registers are initialized to their default values. The PCI section is not affected by *SoftReset*.
- All FIFOs are cleared to empty status.
- The Data Cache is invalidated and enabled. The ICache is turned off.
- Screen refresh requests are disabled. Video outputs are turned off.
- External reset (*XReset*) is asserted during *Reset* and *SoftXReset* (but not *SoftReset*).
- If enabled as a VGA display device, in VGA mode after *Reset* (but not *SoftReset*), else in native mode.
- The RISC PC is set to 0xfffffff0 and the RISC processor starts executing from EPROM.
- The chip is put in V1000 compatibility mode.

The *Reset* configuration for the V2200 is programmed by *XBus[7:0]* and *MD[62:61, 44, 43:41, 40:38, 35, 33, 30:29]* during *Reset* as specified in table 12. If a pin is tied high through a 2.2K ohm resistor, it is read as = 1. If a pin is tied to ground through a 2.2K ohm resistor, it is read as = 0.

Bit(s)	Initializes at *Reset*	Description (without resistors is first entry)
MD[30:29]	*MemSpeed*	Memory speed. 3 = 125MHz, 2 = 100MHz, 1 = 83.33MHz, 0 = 66.67MHz. Use if *XD[5]* is = 1 at *Reset*. See *PLL Control Registers* section.
MD[33]	*MemType*	Memory Type. See *Memory Controller Registers* section.
MD[35]	*MemLatency*	Memory Latency. See *Memory Controller Registers* section.
MD[40:38]	*MemSizeCS0*	Memory Size for Chip Select 0. See *Memory Controller Registers* section.
MD[43:41]	*MemSizeCS1*	Memory Size for Chip Select 1. See *Memory Controller Registers* section.
MD[44]	*MemBanksPerChip*	Memory Banks Per Chip. See *Memory Controller Registers* section.
MD[62:61]	*MemAdrSwizzle*	Memory address swizzle. See *Memory Controller Registers* section.
XD[3:0]	*RISCSpeed*	Maximum RISC clock. 15 = 50MHz, 14 = 55.6MHz, 13 = 60MHz, 12 = 66.67MHz, 11 = 75MHz, 10 = 83.33MHz, 9 = 90MHz
XD[4]	*TestMode#*	1 = Normal Operation, 0 = Test mode.
XD[5]	*PLLClkEnable*	1 = Running, 0 = Direct clock. See *PLL Control Registers* section
XD[7:6]	*DisplayMode*	3 or 2 = VGA, 1 = Multimedia Device, 0 = Other Display Device.
XA[0]	*DACTest*	For test mode only, if set, use *Vin[7:0]* as input to all DACs, else use internal incrementing test counter as input to palette RAM.
XA[1] %%	*SClkPLLEnable*	1 = Use PLL clock, 0 = Direct clock. See *PLL Control Registers* section
XA[2] %%	*PClkPLLEnable*	1 = Use PLL clock, 0 = Direct clock. See *PLL Control Registers* section
XA[3] %%	*PLLAnalogTest*	1 = Normal Operation, 0 = Analog test mode.

Table 12 – Reset configuration bit definition

%% If *XA[3:1]* = 0 and *XD[5]* = 1 at *Reset*, chip enters *TestClkMode*. See *PLL Control Registers* section.

13. Pin Descriptions / Timing Diagrams

V2200 signals have five basic functional groups: PCI, Memory, ROM, Video and External Data bus. These signals are described in the following sections. The Type of each signal is specified as input (I), output (O) or bi-directional (B). For output and bi-directional signals, the nominal output drive size is also specified (Drive). Note, to save pins, memory data pins are also used as address data and control for the ROM.

PCI Pins

V2200 is a 66 MHz PCI device and acts as both a bus slave and bus master. For more information on PCI, see the *PCI Local Bus Specification*, revision 2.1. Basic PCI signal descriptions are given in the following table:

Name	Type	Drive	Description
AD[31:0]	B	16 mA	**PCI Address/Data bus**. Combined PCI address and data bus. A bus transaction contains one address phase and one, or more in the case of a burst, data phases. Data most significant byte is AD[31:24].
C/BE[3:0]#	B	16 mA	**PCI Bus-Command** and **Byte-Enable**. These four bits specify the transaction command during the address phase and bus byte-enables during the data phase.
Par	O	16 mA	**PCI Parity**. Even parity across AD[31:0], C/BE[3:0]# and Par. Parity is valid one cycle delayed from valid address or data. V2200 asserts proper parity for reads, but ignores parity input for writes.
Frame#	B	16 mA	**PCI Cycle Frame**. Current master drive to indicate beginning and duration of an access. When Frame# is de-asserted, the transaction is in the final data phase.
IRdy#	B	16 mA	**PCI Initiator Ready**. Indicates master is ready to complete current data phase. A data phase is completed on any clock where both IRdy# and TRdy# are asserted.
TRdy#	B	16 mA	**PCI Target Ready**. Indicates target is ready to complete current data phase. A data phase is completed on any clock where both IRdy# and TRdy# are asserted.
Stop#	B	16 mA	**PCI Stop**. Indicates target is requesting that the master stop current transaction.
IDSel	I	-	**PCI Initialization Device Select**. Chip select during PCI configuration reads and writes.
DevSel#	B	16 mA	**PCI Device Select**. Target asserts to indicate it has been selected for current access.
Req#	O	16 mA	**PCI Request**. V2200 asserts this to request PCI bus transaction.
Gnt#	I	-	**PCI Grant**. PCI controller asserts this when it grants the PCI bus to the V2200 chip.
IntA#	O	16 mA	**PCI Interrupt Request A**. V2200 asserts this to request a PCI interrupt.
Rst#	I	-	**PCI Reset**. Resets V2200 and asserts XReset#. Must be held for 3 PCI BClk periods.
BClk	I	-	**PCI Clock**. Provides all timing for PCI transactions. PCI signals, other than Rst# and IntA#, are sampled on the rising edge of BClk. PCI clock rates of up to 33 MHz are supported. BClk is asynchronous to both SClkIn and VClk.

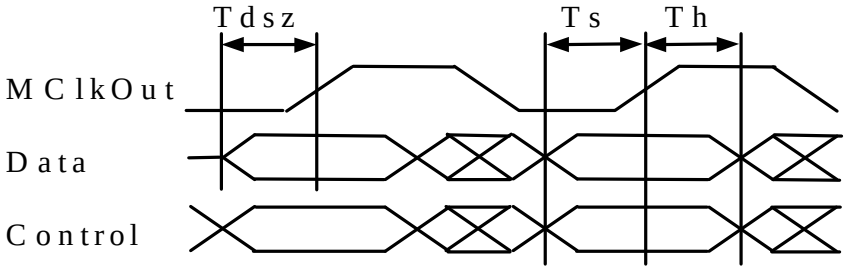
PCI Timing:

Parameter	Symbol	min	max	Units	Comment
PCI Output High Switching Current	-	-44		mA	0<Vout<1.4
		(Vout-1.4)/0.024 -44	11.9*(Vout-5.25)* (Vout+2.45)	mA	1.4<Vout<2.4, max:Vout>3.1
PCI Output Low Switching Current	-	95		mA	Vout>2.2
		Vout/0.023	78.5*Vout* (4.4-Vout)	mA	2.2>Vout>0.55, max: Vout<0.7
PCI Gnt# Setup Time	tGNTs	5		nS	
PCI Setup Time (all others)	tPCIs	3		nS	
PCI Hold Time (all)	tPCIh	0		nS	
PCI Output Delay	tPCIval	2	6	nS	
PCI clock period (BClk)	tBClk	15	∞	nS	
PCI clock, low and high time	-	6		nS	
PCI clock slew rate	-	1.5	4	V/nS	

Memory Pins

The V2200 supports from 2 MB to 16 MB of SGRAM or SDRAM. Note, some memory pins are shared with the BIOS ROM. Also, memory is configured by the memory data bus and external data bus at *Reset*. See the *Reset* section for more information. Memory interface signals are described in the following table:

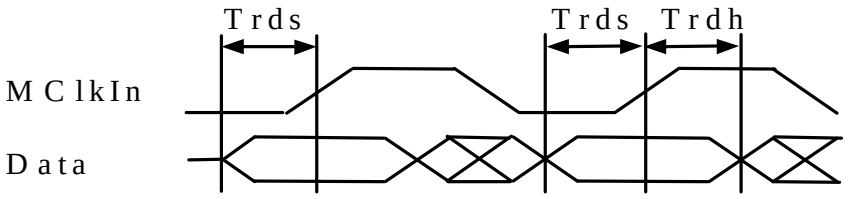
Name	Type	Drive	Description
MD[63:0]	B	8 mA	**Memory Data**. Some memory data pins shared with EPROM.
MRAS#	O	16 mA	**Memory Row-Address-Strobe.**
MCAS#	O	16 mA	**Memory Column-Address-Strobe.**
MWE#	O	16 mA	**Memory Write-Enable.**
MDSF	O	16 mA	**Memory command.**
MDQM[7:0]#	O	8 mA	**Memory byte mask.**
MA[11:0]	O	16 mA	**Memory Address.**
MCS[1:0]#	O	8 mA	**Memory Chip-select.**
MClkOut[1:0]	O	32 mA	**Memory Clock Output.**
MClkIn[1:0]	I	-	**Memory Clock Input.** From *MClkOut* with trace delay matched to memory clock plus memory data trace delays. Used internally for memory read input latches.



Memory write timing diagram

Name	Time (nS)	Description
ts	4.0 (min)	D0-D63. Data bus setup-time before write.
th	2.5 (min)	D0-D63. Data bus hold-time after write.
tdsz	3.5 (min)	D0-D63. Data bus setup-time before write, from tri-state.
tcs	4.0 (min)	All control / address. Setup-time before write.
tch	2.6 (min)	All control / address. Data bus hold-time after write.
ts_{-60}	3.0 (min)	D0-D63. Data bus setup-time before write. (For V2200-60 version.)
th_{-60}	1.5 (min)	D0-D63. Data bus hold-time after write. (For V2200-60 version.)
tdsz	2.5 (min)	D0-D63. Data bus setup-time before write, from tri-state. (For V2200-60 version.)
tcs_{-60}	3.0 (min)	All control / address. Setup-time before write. (For V2200-60 version.)
tch_{-60}	1.6 (min)	All control / address. Hold-time after write. (For V2200-60 version.)

Memory write timing parameters
(Measured at memory chips)



Memory write timing diagram

Name	Time (nS)	Description
trds	2.0 (min)	D0-D63. Data bus setup-time before read.
trdh	2.0 (min)	D0-D63. Data bus hold-time after read.
$trds_{-60}$	1.75 (min)	D0-D63. Data bus setup-time. (For V2200-60 version.)
$trdh_{-60}$	1.75 (min)	D0-D63. Data bus hold-time after read. (For V2200-60 version.)

Memory read timing parameters
(Measured at V2200)

ROM Pins

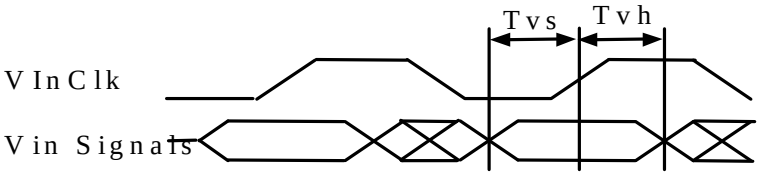
V2200 supports an external (up to) 128K by 8 BIOS ROM, EPROM or EEPROM (EEPROM devices can be programmed in place, contact the factory for programming information). ROM interface signals are described in the following table:

Name	Type	Drive	Description
RD[7:0]	B	8 mA	**BIOS ROM Data.** Shares pins with memory Data (D[7:0]).
RA[16:0]	O	8 mA	**BIOS ROM Address.** Shares pins with memory Data (D[24:8]).
ROMOE#	O	8 mA	**BIOS ROM Output-Enable.** Shares pins with memory Data (D[25]).
ROMWE#	O	8 mA	**BIOS ROM Write-Enable.** Shares pins with memory Data (D[26]).
ROMCS#	O	4 mA	**BIOS ROM Chip-Select.**

Video Pins

V2200 has an internal palette/DAC with hardware cursor. In addition, it has an 8-bit digital video output to enable external NTSC encoding for TV and a byte serial video input bus. Note, *TestMode* must be disabled when using *VIn* or *VOut*. Video interface signals are described in the following table (note, *VOutData* shares pins the *XD[7:0]* bus):

Name	Type	Drive	Description
Red	-	-	**Video red analog output.** Requires external ESD protection.
Green	-	-	**Video green analog output.** Requires external ESD protection.
Blue	-	-	**Video blue analog output.** Requires external ESD protection.
HSync	O	16 mA	**Video Horizontal Sync.** Requires external ESD protection.
VSync	O	16 mA	**Video Vertical Sync.** Requires external ESD protection.
VIn[7:0]	I	-	**Digital Video input data.**
VInActive	I	-	**Digital Video input composite Active** (= not(blank)).
VInVBlank	I	-	**Digital Video input vertical blank.** Used for VBI, if needed by decoder (e.g. Bt829).
VInHSync	I	-	**Digital Video input horizontal sync.**
VInVSync	I	-	**Digital Video input vertical sync.**
VInClk	I	-	**Digital Video input clock.**
VOutActive	O	8 mA	**Digital Video output composite Active video** (not blank).
VOutHSync	O	8 mA	**Digital Video output horizontal sync.** Same polarity as *HSync* output.
VOutVSync	O	8 mA	**Digital Video output vertical sync.** Same polarity as *VSync* output.
VOutClk	O	16 mA	**Digital Video output clock.**
EvenFrame	O	16 mA	**Even Frame.** Even stereoscopic frame displayed.
DDCData	B	16 mA	**VESA Display Data Channel Data.**
DDCClk	B	16 mA	**VESA Display Data Channel Clock output.**



Video in timing diagram

Name	Time (nS)	Description
tvs	5.0 (min)	Set up time before VinClk for Vin[7:0], VInActive, VInVBlank, VinHSync, VinVSync.
tvh	0.0 (min)	Hold up time after VinClk Vin[7:0], VInActive, VInVBlank, VinHSync, VinVSync.

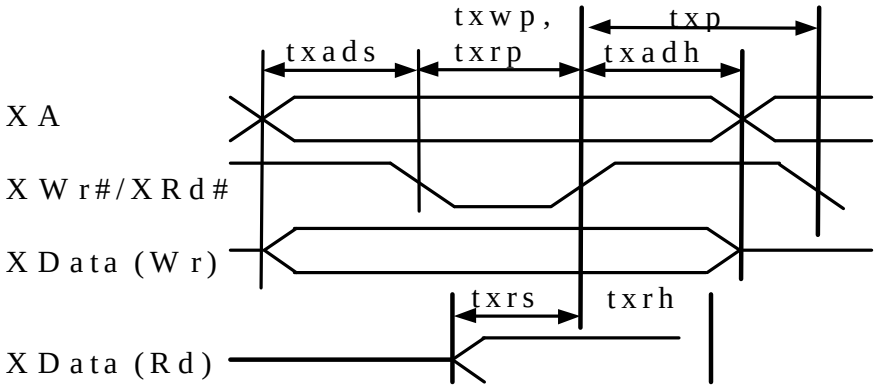
Video in timing parameters

External Data Bus Pins

The external data bus allows V2200 to external byte wide devices mapped into the native mode IO register space. PCI byte, half-word and word accesses are converted into one, two or four accesses respectively, starting with the smallest address. Note, *XBus* accesses are allowed while digital video output is enabled. If the *XBusWaitBlank* bit in the *XBusCtl* register is set, *XBus* accesses only start during horizontal back-porch and sync. Note, *TestMode* must be disabled when using *XBus*. *XBus* interface signals are described in the following table:

Name	Type	Drive	Description
XD[7:0]	B	8 mA	**External Data.** XBus data bus. When digital video output is enabled, used as byte-serial video data output. Also, in test mode, the 8-bits selected by the test select (*Vin[7:0]*) are output. During *Reset*, used for configuration.
XA[3:0]	B	8 mA	**External Address.** Supports 16-byte XBus address space. During *Reset*, used for configuration.
XWr#	O	4 mA	**External Write enable.**
XRd#	O	4 mA	**External Read enable.**
XWait#	I	-	**External access wait.** Allows a slow device to delay completion of access.
XIntr	I	-	**External Interrupt.** External interrupt request to PCI (edge sensitive). Interrupts PCI host only if enabled in the *IntrEn* register. Programmable polarity.
XReset#	O	4 mA	**External Reset.** Reset for local peripherals.

The following figure and table show *XBus* cycle timing diagram and associated timing parameters, respectively:



External bus cycle timing diagram

Name	Time (nS)	Description
txwp	4 * BClk - 3	External bus minimum write pulse width.
txrp	4 * BClk - 3	External bus minimum read pulse width.
txads	2 * BClk - 3	External bus address setup before read or write. Data setup before write.
txadh	2 * BClk - 3	External bus address hold after read or write. Data hold after write.
txrs	10	External bus read data setup to XRd or XWait.
txrh	0	External bus read data hold from XRd.
txp	3 * BClk - 3	External bus minimum precharge between accesses.

External bus timing parameters

Clock Inputs

Parameter	Symbol	min	max	units	Comment
XtalIn (14.318 MHz nominal) Not a 5V Tolerant Input. Use LVTTL Level Input If XTAL Not Used	XtalIn	14.300	14.336	MHz	
XtalOut (n/a if using LVTTL clock)	XtalOut				
Xtal1 low and high time		40%	60%		
DirectMClk	DirectMClk	0	170	MHz	
DirectPClk	DirectPClk	0	170	MHz	

DC Characteristics

Parameter	Symbol	min	max	Units	Comment
Input High Voltage	Vih	2.0	Vcc+0.5	Volts	
Input Low Voltage	Vil	-0.5	0.8	Volts	
Output High Voltage	Voh	2.4		Volts	
Output Low Voltage	Vol		0.5	Volts	
Input High Current	Iih		10	µA	
Input Low Current	Iil		-10	µA	
Output High Current (non-PCI)	-	-4, -8 or -16		mA	See Pin out
Output Low Current (non-PCI)	-	4, 8 or 16		mA	See Pin out
Max Supply Current	Icc			mA	

Power Pins

Note, it is recommended that the analog power supply (VDDA) have an independent 3.3 volt regulator to isolate the DAC and PLL supplies from the digital power.

Parameter	Symbol	min	max	units	Comment
Digital ground	VSS	0	0		
Digital supply	VDD	3.135	3.465	volts	
Latch-up protection	V5V	4.75	5.25	volts	
Analog ground	VSSA	0	0		
Analog supply	VDDA	3.135	3.465	volts	

DAC Pins

Note that the V2200B and V2200C have different current reference parameters.

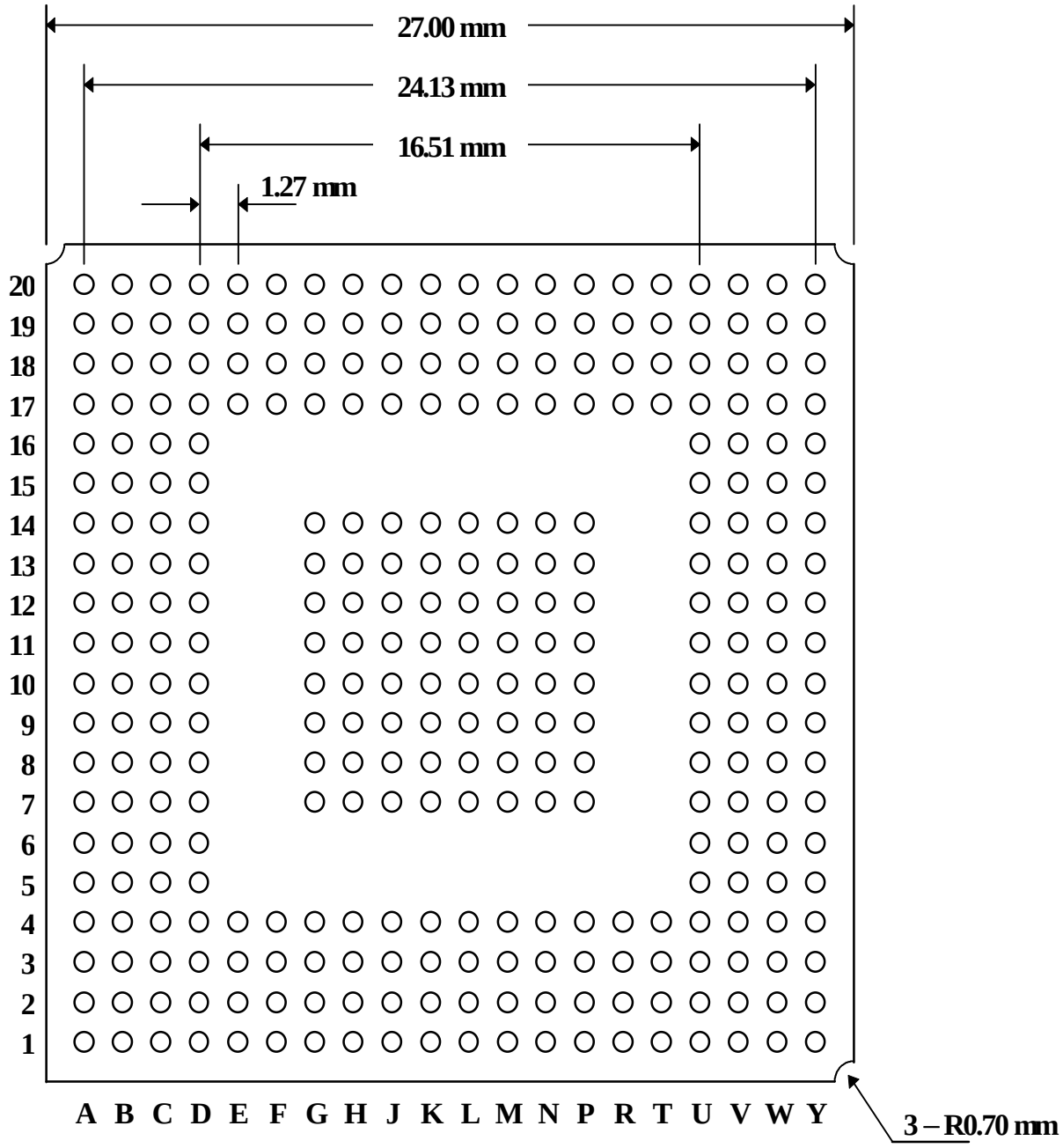
Parameter	Symbol	min	max	units	Comment
DAC bandgap reference output	BGRO	VDDA – 1.23		volts	
DAC Voltage reference input	DACVRefI	VDDA – 1.23		volts	
DAC Current ref (V2200B)	DACIRef	12.4k ohms			to VDDA
DAC Current ref (V2200C)	DACIRef	? ohms			to VSSA
DAC Compensation	DACComp	1 uF			to VDDA

Pin out

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
B1	DirectMClk	W3	n.c.	W20	MA[11]	A19	MD[31]
C2	DirectPClk	Y2	n.c.	V19	MA[10]	B18	MD[30]
D2	XtalOut	W4	n.c.	U19	MA[9]	B17	MD[29]
D3	XtalIn	V4	n.c.	U18	MA[8]	C17	MD[28]
E2	VDDA	U5	n.c.	T17	MA[7]	D16	MD[27]
H3	VSSA	Y3	n.c.	V20	MA[6]	A18	MD[26]
C1	PCLKPLLVDD	Y4	n.c.	U20	MA[5]	A17	MD[25]
D1	PCLKPLLVSS	V5	n.c.	T18	MA[4]	C16	MD[24]
E3	VDDA	W5	IntA#	T19	MA[3]	B16	MD[23]
H4	VSSA	Y5	Rst#	T20	MA[2]	A16	MD[22]
F1	BGRO	V6	Gnt#	R18	MA[1]	C15	MD[21]
F2	VDDA	U7	Req#	P17	MA[0]	D14	MD[20]
E1	DACComp	W6	AD[31]	R19	MD[63]	B15	MD[19]
G1	DACVRef	Y6	AD[30]	R20	MD[62]	A15	MD[18]
H1	DACIRef	V7	AD[29]	P18	MD[61]	C14	MD[17]
J2	VSSA	W7	AD[28]	P19	MD[60]	B14	MD[16]
J3	VSSA	Y7	AD[27]	P20	MD[59]	A14	MDQM[3]#
G4	Blue	V8	AD[26]	N18	MD[58]	C13	MDQM[2]#
F3	VDDA	W8	V5V	N19	MD[57]	B13	MDQM[1]#
F4	VDDA	Y8	AD[25]	N20	MD[56]	A13	MDQM[0]#
K3	VSSA	U9	AD[24]	M17	MD[55]	D12	MClkIn[0]
K4	VSSA	V9	CBE[3]#	M18	MD[54]	C12	MD[15]
J4	Green	W9	IDSEL	M19	MD[53]	B12	MD[14]
G2	VDDA	Y9	AD[23]	M20	MD[52]	A12	MD[13]
G3	VDDA	W10	AD[22]	L19	MD[51]	B11	MD[12]
L3	VSSA	V10	AD[21]	L18	MD[50]	C11	MD[11]
M4	VSSA	Y10	AD[20]	L20	MD[49]	A11	MD[10]
L4	Red	Y11	AD[19]	K20	MD[48]	A10	MD[9]
H2	MCLKPLLVDD	W11	AD[18]	K19	MDQM[7]#	B10	MD[8]
J1	DDCClk	V11	AD[17]	K18	MDQM[6]#	C10	MD[7]
K1	DDCData	U11	BClk	K17	MDQM[5]#	D10	MD[6]
K2	HSync	Y12	AD[16]	J20	MClkIn[1]	A9	MD[5]
L1	VSync	W12	CBE[2]#	J19	MDQM[4]#	B9	MD[4]
M1	EvenFrame	V12	Frame#	J18	MD[47]	C9	MD[3]
L2	XIntr#	U12	IRdy#	J17	MD[46]	D9	MD[2]
N1	VIn[7]	Y13	TRdy#	H20	MD[45]	A8	MD[1]
M2	VIn[6]	W13	DevSel#	H19	MD[44]	B8	MD[0]
M3	VIn[5]	V13	Stop#	H18	MD[43]	C8	VOutClk
P1	VIn[4]	Y14	Par	G20	MD[42]	A7	VOutActive
N2	VIn[3]	W14	CBE[1]#	G19	MD[41]	B7	VOutHSync
R1	V5V	Y15	V5V	F20	MD[40]	A6	VOutVSync
N3	VIn[2]	V14	AD[15]	G18	MD[39]	C7	XRd#
P2	VIn[1]	W15	AD[14]	F19	MD[38]	B6	XWr#
T1	VIn[0]	Y16	AD[13]	E20	MD[37]	A5	XReset#
P3	VInClk	U14	AD[12]	G17	MD[36]	D7	XA[3]
R3	VInActive	V15	AD[11]	F18	MD[35]	C6	XA[2]
R2	VInHSync	W16	AD[10]	E19	MD[34]	B5	XA[1]
U1	VInVSync	Y17	AD[9]	D20	MD[33]	A4	XA[0]
T3	VInVBlank	V16	AD[8]	E18	MD[32]	C5	XWait#
T2	n.c.	W17	CBE[0]#	D19	MClkOut[1]	B4	XD[7]
V1	n.c.	Y18	AD[7]	C20	MClkOut[0]	A3	XD[6]
U3	n.c.	U16	AD[6]	E17	MWE#	D5	XD[5]
U2	n.c.	V17	AD[5]	D18	MCAS#	C4	XD[4]
W1	n.c.	W18	AD[4]	C19	MRAS#	B3	XD[3]
V3	n.c.	Y19	AD[3]	B20	MCS[1]#	B2	XD[2]
V2	n.c.	V18	AD[2]	C18	MCS[0]#	A2	XD[1]
W2	n.c.	W19	AD[1]	B19	MDSF	C3	XD[0]
Y1	n.c.	Y20	AD[0]	A20	RCS#	A1	V5V
VSS	G7 - G14, H7 - H14, J7 - J14, K7 - K14, L7 - L14, M7 - M14, N7 - N14, P7 - P14						
VDD	D4, D6, D8, D11, D13, D15, D17, E4, F17, H17, L17, N4, N17, P4, R4, R17, T4, U4, U6, U8, U10, U13, U15, U17						

14. Mechanical Specifications

The pin locations are shown in the following package diagram.



Solder resist opening: 320 – ∅ 0.635 mm

Top View