





| | | | |
|-------------------|----------|-----------------------|--------|
| Title | | Video Output | |
| | | Note | |
| Spec. | DOC. NO. | Revision | |
| A3 | 1 | 1 | ISH(0) |
| Date / Version | | 7/20/2008 11:23:53 AM | |
| Video camera file | | | |

```

; Horizontal low pixel counter          horizlo.eqn
; Counts the number of pixels in a scan line

CHIP horizlo G22V10

; dotclk is 20 MHz = 50 ns period
; 635 count per scanline = 31.75us
; 508 pixels in visible video region

; vsize  lines
; 00    200
; 01    240
; 10    400
; 11    480

dotclk=1 _vramcs=2 newline=3 text=4 vsizel=5 r0=6 vram7=7 _oe_mem=8 _we_mem=9
_hwsel_pa1=10 bpp0=11 gnd=12
bpp1=13 _we_pa1=14 _we_vram=15 _oe_vram=16 invert=17 _ldbyte=18 c3=19 c2=20 c1=21
c0=22 v0=23 vcc=24

EQUATIONS

; Load a new data byte every 4 pixels in 2, 4, or 8 bpp modes, or every 8 pixels
; in 1 bpp mode.
; Note this is registered, in order to reduce the delay from clock to valid value at
; the shifter input. As a result, the equations check for the column before the
; one where _ldbyte should be asserted.
/_ldbyte := bpp0*c1*/c0 +
            bpp1*c1*/c0 +
            /bpp1*/bpp0*c2*c1*/c0

invert = text*vram7

/_oe_vram = _vramcs + /_oe_mem

/_we_vram = /_vramcs*/_we_mem

/_we_pa1 = /_hwsel_pa1*/_we_mem

/c0 := newline +
      c0

/c1 := newline +
      c1*c0 +
      /c1*/c0

/c2 := newline +
      c2*c1*c0 +
      /c2*/c1 +
      /c2*/c0

c3.oe = _vramcs
/c3 := newline +
      c3*c2*c1*c0 +
      /c3*/c2 +
      /c3*/c1 +
      /c3*/c0

v0.oe = _vramcs
/v0 = text +
      /text*/vsizel*/c2 +
      /text*/vsizel*/r0 +

```

horizlo.eqn

```

; Horizontal high pixel counter           horizhi.eqn
; Counts the number of pixels in a scan line, and generates the hsync, hblank, and
; data
; load enable signals.

CHIP horizhi G22V10

; dotclk is 20 MHz = 50 ns period
; 635 count per scanline = 31.75us
; 512 or 504 pixels in visible video region

```

```

dotclk=1 _vramcs=2 c0=3 c1=4 c2=5 c3=6 vsize0=7 vsize1=8 newframe=9 _vblank=10 r9=11
gnd=12
r8=13 _ldmode=14 _hsync=15 _hblank=16 c9=17 c8=18 c7=19 c6=20 c5=21 c4=22 newline=23
vcc=24

```

EQUATIONS

```

newline = c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*/c0

```

```

; If vsize0 = 1 (240/480 lines), then _ldmode is asserted only once per frame,
; instead of every line.
; That allows the two mode bytes per line to be used as pixel data instead.
; For vsize = 01 (240 lines), _ldmode is asserted at the beginning of line 448
; For vsize = 11 (480 lines), _ldmode is asserted at the beginning of lines 480-495
; The mode byte is loaded when c = 7. For 1bpp mode or 400/480 line modes, this is
; the first
; byte of the line. For 2/4/8bpp 200/240 line modes, this is the second byte of the
; line.
/_ldmode = _vramcs*/vsize0*/c9*/c8*/c7*/c6*/c5*/c4*/c3*c2*c1*c0 +
           _vramcs*/vsize1*vsize0*newframe*/c9*/c8*/c7*/c6*/c5*/c4*/c3*c2*c1*c0 +
           _vramcs*vsize1*vsize0*/_vblank*/r9*/r8*/c9*/c8*/c7*/c6*/c5*/c4*/c3*c2*c1*c0

; _hblank is theoretically active for columns 512-634, but it gets complicated:
; If vsize0 = 0 (200/400 lines), then _hblank is also active for columns 0-7, while
; the line mode
; bytes are loaded.
; Note 1: This is registered. As a result, the equations check for the column before
; the
; one where _hblank should be asserted.
; Note 2: Whatever the current column count, the pixel data currently being input to
; the palette
; is from 8 columns ago at 1bpp, or 4 columns ago at 2/4/8bpp.
; When loading every 8 columns:
;   shifter loads when C=0..0111
;   next cycle C=0..1000, and pixel being input is C=0..0000
;   so column count is 8 ahead of actual pixel
; When loading every 4 columns:
;   shifter loads when C=0..0011
;   next cycle C=0..0100, and pixel being input is C=0..0000
;   so column count is 4 ahead of actual pixel
; Note 3: There would ordinarily be a +/- 4 column difference in when the first
; visible pixel appears
; relative to _hsync, between 1bpp and 2/4/8 bpp at the same vertical resolution.
; This is undesirable
; because it requires bpp to be used in the equation for _hblank, and it means
; mixed-mode screens
; with some 1bpp lines and some 2/4/8bpp lines wouldn't be flush with each other.
; The solution adopted
; here is to use the 1bpp _hblank timing everywhere. For 2/4/8bpp lines, this has

```

horizhi.eqn

the effect of
; suppressing the first 4 columns of visible pixels, and attaching them to the end
of the line instead.
; So the logical line begins at byte 1 of the line data for 2/4/8bpp at 200/240
lines with no mode data,
; and ends at byte 0. with mode data, it begins at byte 3 and ends at byte 0.
Software must compensate
; to create the intended display. But for 2/4/8bpp lines with vsizel = 1 (400/480
lines), this
; wrap-around incorrectly displays the mode data as visible pixels.
; For vsizel = 1: _hblank should be inactive for columns 8-519
; For vsizel = 0: _hblank should be inactive for columns 16-519
; Then subtract 1 from all these to account for _hblank being registered.
00 0000 0111 - 7
00 0000 1XXX - 8-15
:
00 0000 1111 - 15
00 0001 XXXX - 16-31
00 001X XXXX - 32-63
00 01XX XXXX - 64-127
00 1XXX XXXX - 128-255
01 XXXX XXXX - 256-511
10 0000 00XX - 512-515
10 0000 010X - 516-517
10 0000 0110 - 518
_hblank := vsizel*/c9*/c8*/c7*/c6*/c5*/c4*/c3*c2*c1*c0 +
vsizel*/c9*/c8*/c7*/c6*/c5*/c4*c3 +
/c9*/c8*/c7*/c6*/c5*/c4*c3*c2*c1*c0 +
/c9*/c8*/c7*/c6*/c5*c4 +
/c9*/c8*/c7*/c6*c5 +
/c9*/c8*/c7*c6 +
/c9*/c8*c7 +
/c9*c8 +
c9*/c8*/c7*/c6*/c5*/c4*/c3*c2 +
c9*/c8*/c7*/c6*/c5*/c4*c3*c2*c1 +
c9*/c8*/c7*/c6*/c5*/c4*c3*c2*c1*c0
;
; _hync lasts for 12% of the total line time, beginning 2% after the end of the
visible video portion
; _hync is active for 76 pixels, 533-608, 3.8us
10 0001 0101 - 533
10 0001 011X - 534-535
10 0001 1XXX - 536-543
10 001X XXXX - 544-575
10 010X XXXX - 576-607
10 0110 0000 - 608
/_hync := c9*/c8*/c7*/c6*/c5*c4*/c3*c2*c1*c0 +
c9*/c8*/c7*/c6*/c5*c4*c3*c2*c1 +
c9*/c8*/c7*/c6*/c5*c4*c3 +
c9*/c8*/c7*/c6*c5 +
c9*/c8*/c7*c6*c5*/c4*c3*c2*c1*c0
;
; column count wraps at 634
; 10 0111 1010
c4.oe = _vramcs
/c4 := c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*c0 +
c4*c3*c2*c1*c0 +
/c4*/c3 +
/c4*/c2 +
/c4*/c1 +
/c4*/c0

horizhi.eqn

```

c5.oe = _vramcs
/c5 := c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*/c0 +
c5*c4*c3*c2*c1*c0 +
/c5*/c4 +
/c5*/c3 +
/c5*/c2 +
/c5*/c1 +
/c5*/c0

c6.oe = _vramcs
/c6 := c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*/c0 +
c6*c5*c4*c3*c2*c1*c0 +
/c6*/c5 +
/c6*/c4 +
/c6*/c3 +
/c6*/c2 +
/c6*/c1 +
/c6*/c0

c7.oe = _vramcs
/c7 := c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*/c0 +
c7*c6*c5*c4*c3*c2*c1*c0 +
/c7*/c6 +
/c7*/c5 +
/c7*/c4 +
/c7*/c3 +
/c7*/c2 +
/c7*/c1 +
/c7*/c0

c8.oe = _vramcs
/c8 := c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*/c0 +
c8*c7*c6*c5*c4*c3*c2*c1*c0 +
/c8*/c7 +
/c8*/c6 +
/c8*/c5 +
/c8*/c4 +
/c8*/c3 +
/c8*/c2 +
/c8*/c1 +
/c8*/c0

c9.oe = _vramcs
/c9 := c9*/c8*/c7*c6*c5*c4*c3*/c2*c1*/c0 +
c9*c8*c7*c6*c5*c4*c3*c2*c1*c0 +
/c9*/c8 +
/c9*/c7 +
/c9*/c6 +
/c9*/c5 +
/c9*/c4 +
/c9*/c3 +
/c9*/c2 +
/c9*/c1 +
/c9*/c0

```

```

; Low vertical pixel counter           vertlo.eqn
; Counts the low bits of the number of lines in a frame, and outputs masked versions
; of the bits.

CHIP vertlo G22V10

; dotclk is 20 MHz = 50 ns period
; 525 count per frame for 512x480 video, 480 in visible video region, negative vsync
; 449 count per frame for 512x400 video, 400 in visible video region, positive vsync

; vsize  lines
; 00    200
; 01    240
; 10    400
; 11    480

dotclk=1 _vramcs=2 rmask=3 text=4 newline=5 newframe=6 _hwsel_video=7 _oe_mem=8
_we_mem=9 d6=10 _vb1ank_in=11 gnd=12
nc=13 vsize1=14 vsize0=15 d7=16 r3=17 r2=18 r1=19 r0=20 r3m=21 r2m=22 r1m=23 vcc=24

EQUATIONS

d7.oe = /_hwsel_video*/_oe_mem
d7 = _vb1ank_in

; These registers are clocked with a different clock than the CPU driving the bus!
; Hopefully this will still work OK?
vsize0 := _we_mem*vsize0 +
          _hwsel_video*vsize0 +
          /_hwsel_video*/_we_mem*d6

vsize1 := _we_mem*vsize1 +
          _hwsel_video*vsize1 +
          /_hwsel_video*/_we_mem*d7

r1m.oe = _vramcs
/r1m := rmask +
        text +
        newframe +
        /newline*/r1 +
        newline*r1*r0 +
        newline*/r1*/r0

r2m.oe = _vramcs
/r2m := text +
        newframe +
        /newline*/r2 +
        newline*r2*r1*r0 +
        newline*/r2*/r1 +
        newline*/r2*/r0

r3m.oe = _vramcs
/r3m := text +
        newframe +
        /newline*/r3 +
        newline*r3*r2*r1*r0 +
        newline*/r3*/r2 +
        newline*/r3*/r1 +
        newline*/r3*/r0

/r0 := newframe +
      /newline*/r0 +

```

vertlo.eqn

```
newline*r0
/r1 := newframe +
/newline*/r1 +
newline*r1*r0 +
newline*/r1*/r0

/r2 := newframe +
/newline*/r2 +
newline*r2*r1*r0 +
newline*/r2*/r1 +
newline*/r2*/r0

/r3 := newframe +
/newline*/r3 +
newline*r3*r2*r1*r0 +
newline*/r3*/r2 +
newline*/r3*/r1 +
newline*/r3*/r0
```

```

; verthi.eqn
; High vertical pixel counter
; Counts the high bits of the number of lines in a frame, and generates the vsync,
; vblank, and blank
; signals.

CHIP verthi G22V10

; dotclk is 20 MHz = 50 ns period
; 525 count per frame for 512x480 video, 480 in visible video region, negative vsync
; 449 count per frame for 512x400 video, 400 in visible video region, positive vsync

; vsize  lines
; 00    200
; 01    240
; 10    400
; 11    480

dotclk=1 _cpuaccess=2 r0=3 r1=4 r2=5 r3=6 newline=7 _hblank=8 vsize0=9 nc=10 nc=11
gnd=12 nc=13 _blank=14 _vsync=15 _vblank=16 r9=17 r8=18 r7=19 r6=20 r5=21 r4=22 newframe=23
vcc=24

EQUATIONS

; 448 = 01 1100 0000
; 524 = 10 0000 1100
newframe = /vsize0*/r9*r8*r7*r6*/r5*/r4*/r3*/r2*/r1*/r0 +
            vsize0*r9*/r8*/r7*/r6*/r5*/r4*r3*r2*/r1*/r0

; _blank = _hblank*_vblank*_cpuaccess ; try this to blank the display during CPU
access
_blank = _hblank*_vblank

; at 512x400, _vsync is high for lines 412 and 413, low otherwise
; at 512x480, _vsync is low for lines 490 and 491, high otherwise
; 01 1001 110x - 412-413 (high)
; 01 1110 101x - 490-491 (low)
_vsync := /vsize0*/r9*r8*r7*/r6*/r5*r4*r3*r2*/r1 +
            vsize0*r9 +
            vsize0*/r8 +
            vsize0*/r7 +
            vsize0*/r6 +
            vsize0*/r5 +
            vsize0*/r4 +
            vsize0*/r3 +
            vsize0*/r2 +
            vsize0*/r1

; _vblank is low for lines 400-479, at 512x400
; _vblank is low for lines 480+, regardless of vsize
; 400 = 01 1001 0000
; 480 = 01 1110 0000
/_vblank := /vsize0*/r9*r8*r7*/r6*/r5*r4 + ; 400-415
            /vsize0*/r9*r8*r7*/r6*/r5 + ; 416-447
            /vsize0*/r9*r8*r7*/r6*/r5 + ; 448-479
            /r9*r8*r7*r6*/r5 + ; 480-511
            r9 ; 512-1023

r4.oe = _cpuaccess
/r4 := newframe +
        /newline*/r4 +
        newline*r4*r3*r2*r1*r0 +

```

```

verthi.eqn
newline*/r4*/r3 +
newline*/r4*/r2 +
newline*/r4*/r1 +
newline*/r4*/r0

r5.oe = _cpuaccess
/r5 := newframe +
/newline*/r5 +
newline*r5*r4*r3*r2*r1*r0 +
newline*/r5*/r4 +
newline*/r5*/r3 +
newline*/r5*/r2 +
newline*/r5*/r1 +
newline*/r5*/r0

r6.oe = _cpuaccess
/r6 := newframe +
/newline*/r6 +
newline*r6*r5*r4*r3*r2*r1*r0 +
newline*/r6*/r5 +
newline*/r6*/r4 +
newline*/r6*/r3 +
newline*/r6*/r2 +
newline*/r6*/r1 +
newline*/r6*/r0

r7.oe = _cpuaccess
/r7 := newframe +
/newline*/r7 +
newline*r7*r6*r5*r4*r3*r2*r1*r0 +
newline*/r7*/r6 +
newline*/r7*/r5 +
newline*/r7*/r4 +
newline*/r7*/r3 +
newline*/r7*/r2 +
newline*/r7*/r1 +
newline*/r7*/r0

r8.oe = _cpuaccess
/r8 := newframe +
/newline*/r8 +
newline*r8*r7*r6*r5*r4*r3*r2*r1*r0 +
newline*/r8*/r7 +
newline*/r8*/r6 +
newline*/r8*/r5 +
newline*/r8*/r4 +
newline*/r8*/r3 +
newline*/r8*/r2 +
newline*/r8*/r1 +
newline*/r8*/r0

r9.oe = _cpuaccess
/r9 := newframe +
/newline*/r9 +
newline*r9*r8*r7*r6*r5*r4*r3*r2*r1*r0 +
newline*/r9*/r8 +
newline*/r9*/r7 +
newline*/r9*/r6 +
newline*/r9*/r5 +
newline*/r9*/r4 +
newline*/r9*/r3 +
newline*/r9*/r2 +
newline*/r9*/r1 +
newline*/r9*/r0

```

verthi.eqn

newline*/r9*/r0

```

; video mode register          vidmode.eqn
; Loads and stores a byte of video mode settings from VRAM.

CHIP vidmode G22V10

; dotclk is 20 MHz = 50 ns period

dotclk=1 d0=2 d1=3 d2=4 d3=5 d4=6 d5=7 d6=8 d7=9 _ldmode=10 nc=11 gnd=12
nc=13 oe_pa1=14 _oe_charrom=15 rmask=16 text=17 bpp1=18 bpp0=19 pa13=20 pa12=21
pa11=22 pa10=23 vcc=24

```

EQUATIONS

$$pa10.oe = oe_pa1$$

$$pa10 := /_{\neg ldmode} * d0 +$$

$$_{\neg ldmode} * pa10$$

$$pa11.oe = oe_pa1$$

$$pa11 := /_{\neg ldmode} * d1 +$$

$$_{\neg ldmode} * pa11$$

$$pa12.oe = oe_pa1$$

$$pa12 := /_{\neg ldmode} * d2 +$$

$$_{\neg ldmode} * pa12$$

$$pa13.oe = oe_pa1$$

$$pa13 := /_{\neg ldmode} * d3 +$$

$$_{\neg ldmode} * pa13$$

$$bpp0 := /_{\neg ldmode} * d4 +$$

$$_{\neg ldmode} * bpp0$$

$$bpp1 := /_{\neg ldmode} * d5 +$$

$$_{\neg ldmode} * bpp1$$

$$text := /_{\neg ldmode} * d6 +$$

$$_{\neg ldmode} * text$$

$$rmask := /_{\neg ldmode} * d7 +$$

$$_{\neg ldmode} * rmask$$

$$_{\neg oe_charrom} = /text$$

$$oe_pa1 = /bpp0 + /bpp1$$

```

; video data bit shifter
; Loads and shifts a byte from VRAM, for input to the palette.

CHIP shifter G22V10

; dotclk is 20 MHz = 50 ns period

dotclk=1 d0=2 d1=3 d2=4 d3=5 d4=6 d5=7 d6=8 d7=9 c0=10 _ldbyte=11 gnd=12
bpp0=13 bpp1=14 invert=15 q7=16 q6=17 q5=18 q4=19 q3=20 q2=21 q1=22 q0=23 vcc=24

```

```

; bpp    definition
; 00    512@1bpp, shift 1 bit every column, don't output q4-q7
; 01    512@2bpp, shift 2 bits every column, don't output q4-q7
; 10    256@4bpp, shift 4 bits every 2 columns, don't output q4-q7
; 11    128@8bpp, don't shift, output q4-q7

```

EQUATIONS

```

q0 := /_ldbyte*/invert*d0 +          ; Load when _ldbyte=0, invert=0, regardless
of hres.                            ; Load and invert when _ldbyte=0, invert=1,
/_ldbyte*invert*/d0 +                ; If not loading, always shift by 1 when
regardless of hres.                  ; If not loading, always shift by 2 when
/_ldbyte*/bpp1*/bpp0*q1 +           ; If not loading, shift by 4 when hres=10
hres=00.                            ; If not loading, maintain present value
/_ldbyte*/bpp1*bpp0*q2 +           ; If not loading, maintain present value
hres=01.                            ; If not loading, maintain present value
/_ldbyte*bpp1*/bpp0*c0*q4 +         ; If not loading, shift by 1 when
and c0=1.                            ; If not loading, shift by 2 when
/_ldbyte*bpp1*/bpp0*c0*q0 +         ; If not loading, maintain present value
when hres=10 and c0=0.               ; If not loading, maintain present value
/_ldbyte*bpp1*bpp0*q0               ; If not loading, maintain present value
when hres=11.

q1 := /_ldbyte*/invert*d1 +          ; Load when _ldbyte=0, invert=0, regardless
/_ldbyte*invert*/d1 +                ; Load and invert when _ldbyte=0, invert=1,
/_ldbyte*/bpp1*/bpp0*q2 +           ; If not loading, always shift by 1 when
/_ldbyte*/bpp1*bpp0*q3 +           ; If not loading, always shift by 2 when
/_ldbyte*bpp1*/bpp0*c0*q5 +         ; If not loading, shift by 4 when hres=10
/_ldbyte*bpp1*/bpp0*c0*q1 +         ; If not loading, maintain present value
/_ldbyte*bpp1*bpp0*q1               ; If not loading, maintain present value

q2 := /_ldbyte*/invert*d2 +          ; Load when _ldbyte=0, invert=0, regardless
/_ldbyte*invert*/d2 +                ; Load and invert when _ldbyte=0, invert=1,
/_ldbyte*/bpp1*/bpp0*q3 +           ; If not loading, always shift by 1 when
/_ldbyte*/bpp1*bpp0*q4 +           ; If not loading, always shift by 2 when
/_ldbyte*bpp1*/bpp0*c0*q6 +         ; If not loading, shift by 4 when hres=10
/_ldbyte*bpp1*/bpp0*c0*q2 +         ; If not loading, maintain present value
/_ldbyte*bpp1*bpp0*q2               ; If not loading, maintain present value

q3 := /_ldbyte*/invert*d3 +          ; Load when _ldbyte=0, invert=0, regardless
/_ldbyte*invert*/d3 +                ; Load and invert when _ldbyte=0, invert=1,
/_ldbyte*/bpp1*/bpp0*q4 +           ; If not loading, always shift by 1 when
/_ldbyte*/bpp1*bpp0*q5 +           ; If not loading, always shift by 2 when
/_ldbyte*bpp1*/bpp0*c0*q7 +         ; If not loading, shift by 4 when hres=10
/_ldbyte*bpp1*/bpp0*c0*q3 +         ; If not loading, maintain present value
/_ldbyte*bpp1*bpp0*q3               ; If not loading, maintain present value

q4.oe = bpp0*bpp1
q4 := /_ldbyte*/invert*d4 +          ; Load when _ldbyte=0, invert=0, regardless
/_ldbyte*invert*/d4 +                ; Load and invert when _ldbyte=0, invert=1

```

```

shifter.eqn
_1dbyte*/bpp1*/bpp0*q5 +
_1dbyte*/bpp1*bpp0*q6 +
_1dbyte*bpp1*/bpp0*/c0*q4 +
_1dbyte*bpp1*bpp0*q4

q5.oe = bpp0*bpp1
q5 := /_1dbyte*/invert*d5 +
/_1dbyte*invert*/d5 +
_1dbyte*/bpp1*/bpp0*q6 +
_1dbyte*/bpp1*bpp0*q7 +
_1dbyte*bpp1*/bpp0*/c0*q5 +
_1dbyte*bpp1*bpp0*q5

q6.oe = bpp0*bpp1
q6 := /_1dbyte*/invert*d6 +
/_1dbyte*invert*/d6 +
_1dbyte*/bpp1*/bpp0*q7+
_1dbyte*bpp1*/bpp0*/c0*q6 +
_1dbyte*bpp1*bpp0*q6

q7.oe = bpp0*bpp1
q7 := /_1dbyte*/invert*d7 +
/_1dbyte*invert*/d7 +
_1dbyte*bpp1*/bpp0*/c0*q7 +
_1dbyte*bpp1*bpp0*q7

```