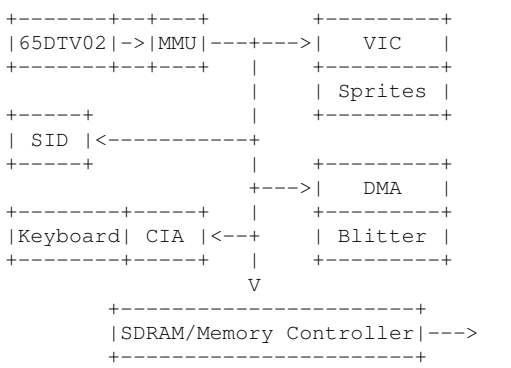
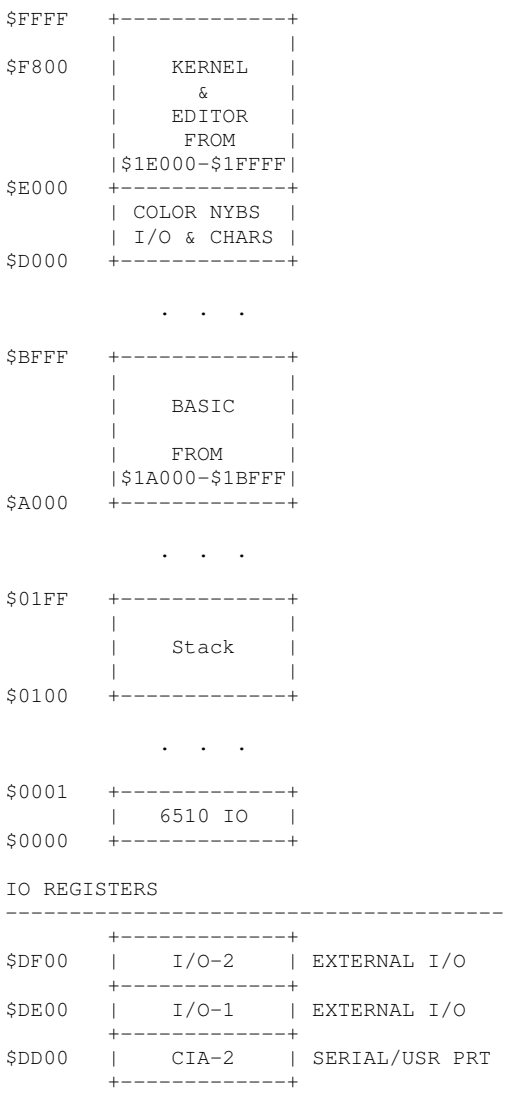
DTV Programming

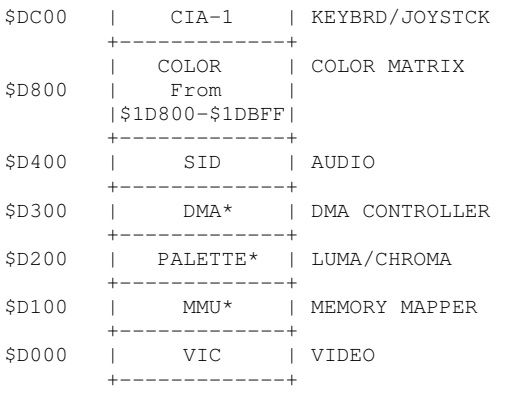
Basic Block Diagram



6502 map at reset

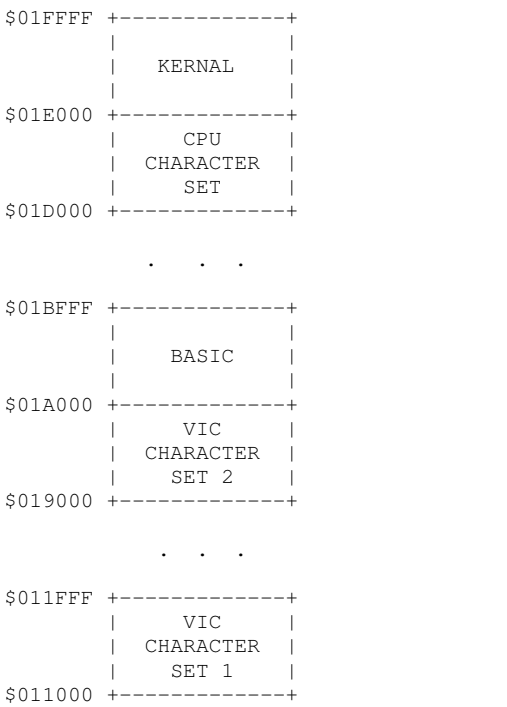


IO REGISTERS

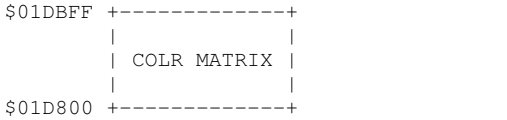


*Extended control must be enabled to address these registers

ROM BANK 1



RAM BANK $01



EXTENDED SID REGISTERS

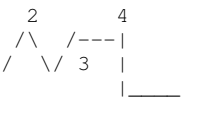
$D41D Writes to voice 1's upper 8 bits of waveform accumulator.
$D41F Writes to voice 2's 8bit envelope generator.

Writing to voice 1's waveform accumulator when frequency is set to 0

can be used for 8 bit digital sample playback.
  The accumulator may also be set to a non-zero frequency and written to for compressed sample playback.

Saw tooth:
  The MSB will set the direction of the ramp.
  0 = ascending
  1 = descending

  The accumulator frequency sets the angle off ramp.



Complex waveforms can be generated with fewer data points

1 Accumulator is loaded with %0000000 (ascending MSB = 0) and frequency is set to fast rise time.

2 Accumulator is loaded with %11111111 (descending MSB = 1) and slower frequency.

3 Accumulator is loaded with %11111111 and frequency is set to 0.

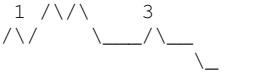
4 Accumulator is loaded with %00000000 and frequency is set to 0.

Noise:
  Toggling bit[3] from 0 to 1 will advance the noise LFSR register.

During Channel 2 Sync modulation:
  Toggling accumulator bit[7] from 1 to 0 will sync modulate voice2.

During Channel 2 Ring modulation:
  Setting bit[7] = 1 in accumulator will make channel 2 waveform descend.

Writing to voice 2's envelope generator.



During attack:
  Writing a lower value than current attack level will start attack over at the lower position (see point 1)
  Writing %11111111 will start decay state.

During Decay:
  Writing a higher value than current decay level will start decay over at the higher position (see point 2)
  Writing sustain value will change to sustain state.

During Sustain:
  Writing a higher value than sustain will start decay again. (see point 3).
  Writing a lower value than decay will start release state.

During Release:
  Writing higher value than current release will start release at a higher value.

VIC FUNCTION BLOCKS

  The VIC has 4 major functional blocks: Address generator, pixel shifters, color decoder and color/character data line buffer.

  The address generator forms all of address to be used fetches of graphics, character pointers and color data.

  Pixel shifters take fetched data and shifts out 1,2,4 or 8 bits per dot clock (or pixel on screen)

  The color decoder maps colors to pixel data that is shifted out of the pixel shifters. Color data may come from fetched color matrix, character matrix or color registers.

  Color/character line buffer is a 40 x 16 bit memory that stores the character and color data every bad line. This memory is read back over the next 8 lines and used with the color encoder.

EXTENDED VIC REGISTERS

$D036 53302 Color Bank Low
             (See address generator)
$D037 53303 Color Bank High
             (See address generator)
$D038 53304 Linear Count A modulo low
             (See address generator)
$D039 53305
      Bits[3:0]
             Linear Count A modulo high
             (See address generator)
$D03A 53306 Linear Count A Start low
             (See address generator)
$D03B 53307 Linear Count A Start middle
             (See address generator)
$D03C 53308
      Bit[0] Linear addressing when set
      Bit[1] Border off when set
      Bit[2] High color when set
             (Extended color decoder mode)
      Bit[3] Overscan For linear modes
             50 columns NTSC
             47 columns PAL
             This mode is severely broken. It ignores modulo's unless disabled around cycle 57 and messes

```
              with sprite fetches.
      Bit[4]  ColorRAM Fetch Disable
              Repeats value that are in
              line buffer.  Line buffer
              is cleared during VBlank.
      Bit[5]  CPU bad line Disable
              (bad lines are emulated
              for CPU for compatibility)
      Bit[6]  Chunky Enable
              (See color decoder and
              address generator)

$D03D 53309   Graphics fetch bank
              (used for all classic VIC
              graphics fetches)
$D03F 53311
      Bit[0]  Enable extended feature
              registers when set.  This
              will also enable 256 color
              $D020, $D021, $D022,
              $D023, $D024
              Registers set under this
              Mode will remain after bit
              is cleared.
      Bit[1]  Setting disables extended
              modes until next reset.
$D040 53312
      Bit[0]  PAL line timing when set
              (63 cycles in PAL 65 in
              NTSC)
      Bit[1]  Burst phase alternate when
              set

      Bit[2]  V1 DTV Palette
              compatibility when set

$D041 53313   Burst rate modulus high
              Default = 28
$D042 53314   Burst rate modulus middle
              Default = 19
$D043 53315   Burst rate modulus low
              Default = 120

Fout = SysClk * N/16777216

Where N is the modulo value and Fout is
desired burst frequency

NTSC/32.64mhz SysClk
3.579545mhz / 0.00000194549560546875=
1839914.204862745098039215686274 5
NTSC/32.64mhz = 1C132A

PAL/31.36mhz SysCLk
4.433619mhz / 0.00000186920166015625=
2371931.875787755102040816326530 6
PAL/31.36mhz Modulus = 24315B

Ntsc/32.7272mhz SysClk
0.00000195069313049316406 25
1835011.844787210638245862768583 9
NTSC/32.7272mhz Modulus = $1C0000

PAL 31.5279mhz SysClk
0.00000187920928001403808593 75
2359300.290368340422229263604616 86
PAL/31.5279mhz  Modulus = 240000


$D044 53316
      Bits[6:0]
```

```
              During reads
              CPU Cycle

              Writes IRQ trigger cycle
               NTSC = 0->64
              PAL = 0->55 & 58->64
              (PAL Skips 2 cycles)
              Default = 64
$D045 53317
      Bits[5:0]
              Linear Count A Start high
              (See address generator)
$D046 53318  Linear Count A Step
              (See address generator)
$D047 53319  Linear Count B modulo low
              (See address generator)
$D048 53320
      Bits[3:0]
              Linear Count B modulo high
              (See address generator)
$D049 53321  Linear Count B start low
              (See address generator)
$D04A 53322  Linear Count B start Mid
              (See address generator)
$D04B 53323
      Bits[5:0]
              Linear Count B start high
              (See address generator)
$D04C 53324  Linear Plane B Step
              (See address generator)
$D04D 53325
      Bits[5:0]
              Sprite Bank

$D04E 53326  Scan line timing adjust.
              Adds about 25ns per count

              NTSC/32.64mhz = $D
              PAL/31.36mhz  = $5

              NTSC 32.72mhz = $0
              PAL  32.5279mhz = $0
$D04F 53327
      Bit[1:0]
              Saturation
                00 lowest
                11 highest
      Bit[2] Burst lock to line
              PAL/NTSC
      Bit[3] Burst lock to line with
              negative phase "walk"

VIC ADDRESS GENERATION
---------------------------------------

Sprite Addresses
  Sprite Pointer Fetch:
  (Sprite Bank & Character Matrix)

  Sprite Graphics Fetch:
  (Sprite Bank & Pointer & DMA Count)

Linear addressing = 0
  Graphics Fetches:
  (GfxBank[5:0] & VIC Address[15:0])

VIC Addresses are standard 64k
addressing modes set with MCM/BMM/ECM

  Character Fetches:
  (LinearCountA[21:16] & VIC
```

Address[15:0])

VIC Addresses are standard 64k
addressing modes set with MCM/BMM/ECM

Set linear count to step0 and set Start
Address A[21:16] to desired character
bank. Linear A counter will count 40
steps and add 1 modulus per active scan
line.

  Color Fetches:
  (LinearCountA[11:0] & Matrix[9:0])

For c64 compatibility set linear count
to step = 0, modulus = 0 and set Start
Address A[11:0] to desired character
bank. Linear A counter will count 40
steps and add 1 modulus per active scan
line.

Linear addressing = 1
Color Fetch Disable = 0
Chunky Enable = 0

  Graphics Fetches:
  (Plane A Linear Address[21:0])

  Character Fetches:
  (Plane B Linear Address[21:0])

  Color Fetches:
  (ColorBank[11:0] & Matrix[9:0])

COLOR DECODER

ECM = 0 BMM = 0 MCM = 0 HIGHCOLOR = 1
  Plane A = 0 (8bit background color 0)
  Plane A = 1 (8bit color data)

ECM = 1 BMM = 0 MCM = 0 HIGHCOLOR = 1
  Plane A = 0
      Character data [7:6]
      +-----------------------------+
      |00 = 8bit background color 0 |
      |01 = 8bit background color 1 |
      |10 = 8bit background color 2 |
      |11 = 8bit background color 3 |
      +-----------------------------+
  Plane A = 1 (8bit color data)

ECM = 0 BMM = 0 MCM = 1 HIGHCOLOR = 1
  Color data[3] = 0
    Plane A = 0 (8bit background 0)
    Plane A = 1 Color[7:4]‘0’Color[2:0]
  Color data[3] = 1
      Plane ‘A’ pixels only
      +-----------------------------+
      |00 = 8bit background color 0 |
      |01 = 8bit background color 1 |
      |10 = 8bit background color 2 |
      |11 = Color[7:4] ‘0’ Color[2:0]
      +-----------------------------+

ECM = 0 BMM = 1 MCM = 1 HIGHCOLOR = 1
  Plane A = 00 (8bit background 0)
  Plane A = 01 (‘0000’ Character[7:4])
  Plane A = 10 (‘0000’ Character[3:0])
  Plane A = 11 (8bit color data)

Six's FRED MODE
8bpp Packed Bitmap

ECM = 1 BMM = 1 MCM = 1 HIGHCOLOR = 1
  8 bit pixel is made up of
 (ColorRam[3:0],PlaneBShifter[1:0],
  PlaneAShifter[1:0])

  One could think of this mode as FLI with no cpu overhead and a re-definable palette, but it is actually much more powerful. It is a cellular mode, with 4x8 cells. Each pixel is 2 hires pixels wide. Each 4x8 cell can contain any
of 16 colors, the downside being that those 16 colors have to have the same high nibble, which is determined by the ColorRam for that cell. Thus, if ColorRam is $00, you can use $00-$0f in that cell, $01 you can use $10-$1f in that cell. The lower nibble is set as shown above, bits 0-1 being from Plane A, bits 2-3 from Plane B. So if Color Ram is $40, the byte in Plane A is %10101100, and the byte in Plane B is %00011010, the pixel colors will be $48,$49,$4e,$42.

Six's FRED MODE2

ECM = '1' BMM = '1' MCM = '1'
HIGHCOLOR = 0 LinearAddressing = '1'
  8 bit pixel is made up of
 (PlaneBShifter[1:0],ColorRam[3:2],
  PlaneAShifter[1:0],ColorRam[1:0])

Two Plane Bitmap

ECM = 1 BMM = 1 MCM = 0 HIGHCOLOR = 1
LinearAddress = 1
  Plane A = 0 Plane B = 0
    (Background 0)
  Plane A = 0 Plane B = 1
    ('0000' Color[7:4])
  Plane A = 1 Plane B = 0
    ('0000' Color[3:0])
  Plane A = 1 Plane B = 1
    (Background 1)

CHUNKY 8BPP Bitmap

ECM = 1 BMM = 0 MCM = 1 HIGHCOLOR = 1
ColorFetchDisable = 0 LinearAddress = 1
ChunkyEnable = 1

  Chunky mode displays 8 8bit pixels per CPU cycle. The first 4 pixels come from counter B. Last 4 pixels come from counter A. To set up a linear video frame buffer the step size must be set to 8(4 pixels are fetched per access per plane) and counter A's start address should be 8 more than plane B's start address(i.e. plane A = 00000 plane B = 00008).

Pixel data for this example
B0 B1 B2 B3 A0 A1 A2 A3 B4 B5 ...

8BPP Pixel Cell

ECM = 1 BMM = 0 MCM = 1 HIGHCOLOR = 1
ColorFetchDisable = 1 LinearAddress = 1
ChunkyEnable = 1

```
  Cell 1 Data      Cell 2 Data
0 1 2 3 4 5 6 7    64   ...     72
8             16
      ....

55            63
```

VIC ADDRESS GENERATOR

The VIC has three cycles to fetch data per 8 pixels displayed

Cycle 1: Character Fetch/Counter A
Cycle 2: Color Fetch/DMA/Blitter
Cycle 3: Graphic Fetch / Counter B
Cycle 4: CPU Access

Addresses for each of the cycles are generated with counters (some with modulus).

Counter A : 22bits with start, modulo and step
Counter B : 22bits with start, modulo and step

RowCounter : 3 bits that count the lines from the last bad line. It terminates at 7

Matrixcount : 10 bits that increments Every character read during bad lines

Character Fetch Addresses

Linear count = 0 ChunkyEn = Don’t Care
ColorDisable = Don’t care

Used during normal legacy vic operation to read character matrix. Linear count A can be enabled to change banks during screen fetches or set to a constant for a bank.

```
Address = LinearCountA(21 downto 16) &
          pa & vm & matrix_counter
```

Linear count = 1 ChunkyEn = 1
ColorDisable = 0

Used with 8bpp cell mode. Color pointers fetched during bad lines are used to make up cell addresses.

```
Address = LinearCountB(21 downto 14) &
          next_color_fetch_data &
          row_counter & '1' &
          LinearCountB(1 downto 0)
```

Linear count = 1 ChunkyEn = Don't Care
ColorDisable = 1

Used with plane type bitmaps and chunky 8bpp

Address = LinearCountB

Color Fetch Addresses

ChunkyEn = 1 ColorDisable = 0
I can't remember why this is here at the moment
Address = LinearCountA

ChunkyEn = 0

This addressing mode is used during legacy VIC addressing and 8bbp cell mode character fetches.
Address = ColorBankHigh & ColorBankLow & matrix_counter

Graphics Fetch
---
bmm = 0 ecm = 0 LinearAddressing = 0

Address = GraphicsBank & pa & cb & CharacterPointer & row_counter

bmm = 0 ecm = 1 LinearAddressing = 0

Address = GraphicsBank & pa & cb & "00" & CharacterPointer(5 downto 0) & row_counter

bmm = 1 LinearAddressing = 0

Address = GraphicsBank & pa & cb(2) & matrix_counter & row_counter

ChunkyEn = 1 ColorDisable = 0

This mode is used for 8bpp cell mode.

Address = LinearCountB(21 downto 14) & ColorPointer & row_counter & '0' & LinearCountB(1 downto 0)

ChunkyEn = 1 ColorDisable = 1

Used for 8bpp bitmap mode. Note the inverted linearCountB. This keeps character fetch and this fetch 4 bytes apart with the same counter.

Address = LinearCountB(21 downto 3) & not LinearCountB(2) & LinearCountB(1 downto 0)

SETTING VIDEO STANDARDS

The DTV allows individual control of different components of PAL and NTSC. The components can be mixed and matched to create NTSC, NTSC(J) and PAL

Control registers are:
$D040 53312
      Bit[0] PAL line timing when set
      Bit[1] Burst alternate when set
      (Other bits are in this
       this register)
$D041 53313 Burst rate modulus high
$D042 53314 Burst rate modulus middle
$D043 53315 Burst rate modulus low
$D04E 53326 Scan line timing adjust
$D04F 53327 Scan line phase
            relationship

$D040 53312
      Bit[0] PAL line timing when set

  This switch adjusts PAL line timing to have 63 CPU cycles horizontal proper scan rate with a 31.xxx mhz crystal. When cleared there will be NTSC scan line timing to have 65 cycles and a proper scan rate with a 32.xxxmhz crystal.

     BIT[1] Burst alternate when set

  This switch enables PAL backwards 1/4 phase backwards burst “walk” per scan line and 180deg alternation. NTSC mode locks 180 drift per scan line.

$D041 53313 Burst rate modulus high
$D042 53314 Burst rate modulus middle
$D043 53315 Burst rate modulus low

  Color is generated with reference to the burst frequency. The burst modulus registers set a fractional digital synthesizer.

Fout = SysClk * N/16777216

Where N is the modulo value and Fout is desired burst frequency

NTSC/32.64mhz SysClk
32.64/16777216 = 0.0000019454956054687
Burst 3.579545mhz /
0.00000194549560546875 =
1839914.204862745098039215686274 5

NTSC Modulus = $1C132A

PAL/31.36mhz SysCLk
Burst 4.433619mhz /
0.00000186920166015625 =
2371931.875787755102040816326530 6

PAL Modulus $24315B

Ntsc 32.7272
0.00000195069313049316406 25
1835011.844787210638245862768583 9
NTSC Modulus = $1C0000

PAL 31.5279mhz xtal
0.00000187920928001403808593 75
2359300.290368340422229263604616 86
PAL Modulus = 240000

$D04E 53326 Scan line timing adjust.

Color information in PAL and NTSC have a precise relationship with horizontal timing. The lower nibble of this register will add ~20ns(crystal dependant) per value to the scan line. Adjust this to have stable color lock

```
        NTSC/32.64mhz = $D
        PAL/31.36mhz  = $5

        NTSC 32.72mhz = $0
        PAL  32.5279mhz = $0
```

```
$D04F 53327 Scan line phase
            relationship
```

The PAL video standard alternates the color information 180 degrees every other scan line and NTSC maintains a constant phase relationship. Phase alternating relationship can be adjusted in 22.5 deg steps relative to burst and relative every other line with $D04F. Use this to fine tune hue and interline color.

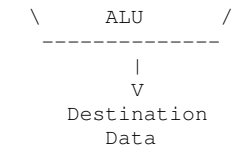
DMA REGISTERS

Base $D3XX

```
$D300 Source [7:0]   (Low)
$D301 Source [15:8]  (Middle)
$D302 Source [23:16] (High)
      Bits[23:22] 00 = ROM
                  01 = RAM
                  10 = RAM + Registers
$D303 Destination[7:0]  (Low)
$D304 Destination[15:8] (Middle)
$D305 Destination[23:16] (High)
      Bits[23:22] 00 = ROM
                  01 = RAM
                  10 = RAM + Registers
$D306 Source Step[7:0]
$D307 Source Step[15:8]
$D308 Destination Step[7:0]
$D309 Destination Step[15:8]
$D30A DMA Length[7:0]
$D30B DMA Length[15:8]
$D30C Source Modulo[7:0]
$D30D Source Modulo[15:8]
$D30E Destination Modulo[7:0]
$D30F Destination Modulo[15:8]
$D310 Source Line Length[7:0]
$D311 Source Line Length[15:8]
$D312 Destination Line Length[7:0]
$D313 Destination Line Length[15:0]
$D31D ClearIRQ[0] Write '1' to clear
                  IRQ
$D31E Source Modulo Enable[0] when set
      Destination Modulo Enable[1]
$D31F Bit[0]   Force Start DMA when set
      Bit[1]   Swaps source with
               Destination when set
      Bit[2]   Source Direction
               Positive when set
      Bit[3]   Destination Direction
               Positive when set
      Bit[4]   VIC IRQ Start enables
               DMA on VIC IRQ when set
```

```
      Bit[5]   Start on blitter done
               when set
      Bit[6]   VBlank Start when set
      Bit[7]   IRQ Enable Enables DMA
               Done IRQ's when set

      During reads
      Bit[0]  DMA Busy
      Bit[1]  IRQ


BLITTER REGISTERS
---------------------------------------
$D320  Source A [7:0]   (Low)
$D321  Source A [15:8]  (Middle)
$D322  Bits[5:0]
       Source A [21:16] (High)
$D323  Source A Modulo[7:0]
$D324  Source A Modulo[15:8]
$D325  Source A Line Length[7:0]
$D326  Source A Line Length[15:8]
$D327  Source A Fractional Step
       point between bit 3 and 4
$D328  Source B [7:0]   (Low)
$D329  Source B [15:8]  (Middle)
$D32A  Bits[5:0]
       Source B [21:16] (High)
$D32B  Source B Modulo[7:0]
$D32C  Source B Modulo[15:8]
$D32D  Source B Line Length[7:0]
$D32E  Source B Line Length[15:8]
$D33F  Source B Fractional Step
       point between bit 3 and 4
$D330  Destination [7:0]   (Low)
$D331  Destination [15:8]  (Middle)
$D332  Bits[5:0]
       Destination [21:16] (High)
$D333  Destination Modulo[7:0]
$D334  Destination Modulo[15:8]
$D335  Destination Line Length[7:0]
$D336  Destination Line Length[15:8]
$D337  Destination Fractional Step
       point between bit 3 and 4.
$D338  Blit Length[7:0]  (Low)
$D339  Blit Length[15:0] (high)

$D33A  Bit[0] Force Start
              Strobe when set
       Bit[1] Source A Direction
              Positive when set
       Bit[2] Source B Direction
              Positive when set
       Bit[3] Destination Direction
              Positive when set
       Bit[4] VIC IRQ Start when set
       Bit[5] CIA IRQ Start when
              set($DCXX CIA)
       Bit[6] V Blank Start when set
       Bit[7] Blitter IRQ Enable when
              set

$D33B
     Bit[0]   Disable Channel B
              (data into b port of ALU
              is forced to %00000000.
              ALU functions as normal)
     Bit[1]   Write Transparent Data
              when set
              (Data will be written if
              source a data *IS*
               %00000000.  This can be
```

```
              used with channel b and
              ALU set to "OR" to write
              Data masked by source A.)
              Cycles will be saved if
              No writes.
     Bit[2]   Write Non Transparent
              when set
              (Data will be written
              if SourceA fetched data
              is *NOT* %00000000.  This
              may be used combined with
              channel b data and/or
              ALU)  Cycles will be
              Saved if no write.

$D33E  Bit[2:0] Source A right Shift
              000 SourceA Data
              001 LastA[0],SourceA[7:1]
                        ...
              111 LastA[6:0],SourceA[7]
     Bit[5:3] Minterms/ALU
              000 AND
              001 NAND
              010 NOR
              011 OR
              100 XOR
              101 XNOR
              110 ADD A + B
              111 SUB A - B

$D33F  Bit[0] Clear Blitter IRQ
     Bit[1] Source A Continue
     Bit[2] Source B Continue
     Bit[3] Destination Continue
            Restart counters from
            location they stop during
            last blit.

     During Reads
     Bit[0] Busy when set
     Bit[1] IRQ when set


BLITTER DATAPATH
----------------

              Source A       Source B
                  |              |
                  V              V
            +-----------+  +-----------+
            | Comparator|  |DisableGATE|
            | == or !=  |  | %0000000  |
            +-----------+  +-----------+
                  |              |
          +-------+              |
          |       |              |
          |       |              |
    +-----------+ |              |
    |Last A Data| |              |
    +-----------+ |              |
          |       |              |
          |       |              |
          |       V              |
          |  +-------+           |
          +->|Shifter|           |
             +-------+           |
                  |              |
                  V              V
               \-----\______/-----/
                \                /
```



Last SourceA register stores data from previous access and can be shifted into MSB's of current SourceA. Last SourceA register is cleared at start of DMA's and at the end of each DMA line (when modulus value is applied to SourceA address). This is useful for scrolling video data.

Blitter and DMA length is the total number of bytes to be transferred in one triggered event.

Line length is the length of one contiguous stream of data, before a modulus value is added to address.

Modulus values are added to addresses when the line length for the channel has been reached. This is useful for moving rectangular blocks of data.

Continue bits when set will keep last address value for channel after DMA stops. These need to be set after the first DMA access or addresses will not be set with start value.

IRQ start bits when set will automatically start a DMA access when the IRQ condition is true. These are edge triggered and will only start if in idle state.

Blitter accesses take advantage of the burst access of SDRAM and can only operate on SDRAM. The DMA channel can operate on any RAM, ROM or registers, but does not support burst access (transfers are slower)

The blitter will cache 4 bytes of data and will not access RAM for that channel if new data is not needed. This saves memory bandwidth and allows for constant values.

Disabling Color accesses will give maximum cycles for DMA. Sprite DMA will have still higher priority than blitter.

Bandwidth Examples:
During 65 cycle line, no sprites, read/write steps of 1,no color fetch and 1 channel reads.

Cycle 0 4 reads
Cycle 1-4 4 writes
Cycle 5 4 reads
Cycle 6-9 4 writes
Cycle 10 4 reads
Cycle 11-14 4 writes
Cycle 15 4 reads

```
Cycle 16-19 4 writes
Cycle 20    4 reads
Cycle 21-24 4 writes
Cycle 25    4 reads
Cycle 26-29 4 writes
Cycle 30    4 reads
Cycle 31-34 4 writes
Cycle 35    4 reads
Cycle 36-39 4 writes
Cycle 40    4 reads
Cycle 41-44 4 writes
Cycle 45    4 reads
Cycle 46-49 4 writes
Cycle 50    4 reads
Cycle 51-54 4 writes
Cycle 55    4 reads
Cycle 56-59 4 writes
Cycle 60    4 reads
Cycle 61-64 4 writes

Total bytes transferred = 52
Bytes transferable in 262 lines = 13624
(Not counting pre-buffered reads and
transparent pixel optimizing)

(~10) 40 X 32 pixel 8bpp BOB's (1280
bytes) can be placed per frame.
(~18) 40 x 32 Pixel Fred1/2 BOB's(1280
bytes.


  During 65-cycle line, no sprite,
read/write steps of 1, no color fetch
and 2 channels reads.

Cycle 0       4 reads
Cycle 1       4 reads
Cycle 2-5     4 writes
Cycle 6       4 reads
Cycle 7       4 reads
Cycle 8-11    4 writes
Cycle 12      4 reads
Cycle 13      4 reads
Cycle 14-17   4 writes
Cycle 18      4 reads
Cycle 19      4 reads
Cycle 20-24   4 writes
Cycle 25      4 reads
Cycle 26      4 reads
Cycle 27-30   4 writes
Cycle 31      4 reads
Cycle 32      4 reads
Cycle 33-36   4 writes
Cycle 37      4 reads
Cycle 38      4 reads
Cycle 39-42   4 writes
Cycle 43      4 reads
Cycle 44      4 reads
Cycle 45-46   4 writes
Cycle 47      4 reads
Cycle 48      4 reads
Cycle 49-52   4 writes
Cycle 53      4 reads
Cycle 54      4 reads
Cycle 55-58   4 writes
Cycle 59      4 reads
Cycle 60      4 reads
Cycle 61-64   4 writes

Total bytes transferred = 44
Bytes transferable in 262 lines = 11528
```

(Not counting pre-buffered reads and transparent pixel optimizing)

(~9) 40 X 32 pixel 8bpp BOB's (1280 bytes) can be replaced per frame.
(~16) 40 x 32 Pixel Fred1/2 BOB's(1280 bytes.

Fractional incrementing can be used for scaling data.

Bits[7:4] are whole number of steps.
Bits[3:0] are fractions of steps per Access

Examples:
Default %00010000 (step of 1 to 1)
%00001000 (step of 1 to .5)
%00000100 (step of 1 to .25)
%00001100 (step of 1 to .75)
%00000000 (no step)

Steps less than 1 on source channels can save read accesses. For example source steps of .25 will take 1 read access instead of 4 for the same amount of writes.

It may be advantageous to have a source channel with a slower step than the other source channel when using minterms to transform higher frequency data with low frequency data.

Steps of 0 on read channels will cause the blitter to read(once) the start address and use that value as a constant during the blit operation. Any update to this memory location after blit has started will not be recognized, since the value is in the blitters cache.

Step of 0 on destination channel will cause all writes to the start address. (may not be very useful)

Non-transparency blits write whole bytes to memory if channel A is non zero value (good for placing images on chunky bitmaps). Zero values will save one write cycle.(Very good thing!)

Transparency blits write whole bytes to memory if values in channel a are zero. Combined with channel B and the alu set to "OR" the reverse of non-transparency can be done affectively only replacing parts of bitmap that had been written before. Non-zero values save one write cycle. (Very good thing!)

Reads on blits are 4x faster than writes.

MEMORY MAPPER

The memory mapper can select which bank the ROMs are fetched from. The

ROMs may be fetched from RAM, but still will remain write protected.

$D100 Kernal bank
$D101 Basic bank

Bits [5:0] Bank Location
Bits [7:6] 00 = ROM
           01 = RAM

  Registers are write only and may only be accessed when extended mode is active.

  Moving Kernal and Basic into SDRAM will allow a 4x speed increase in CPU burst mode.

CPU EXTENSIONS

REGISTER FILE

  The original 6502 only contained 3 registers (A, X and Y). The DTV now contains 16 registers which can be mapped into A, X and Y.
  Registers 10 - 15 are dual purpose banking registers and can also be used with ALU operations.

REG 0	DEFAULT ACCUMLATOR
REG 1	DEFAULT Y REGISTER
REG 2	DEFAULT X REGISTER
REG 3 TO Reg 7	Reserved
REG 8	Bank0-3 Access Mode
REG 9	CPU Control
REG 10	BASE PAGE(Was ZP)
REG 11	STACK BASE
REG 12 $0000-$3FFF	SEGMENT BANK
REG 13 $4000-$7FFF	SEGMENT BANK
REG 14 $8000-$BFFF	SEGMENT BANK
REG 15 $C000-$FFFF	SEGMENT BANK

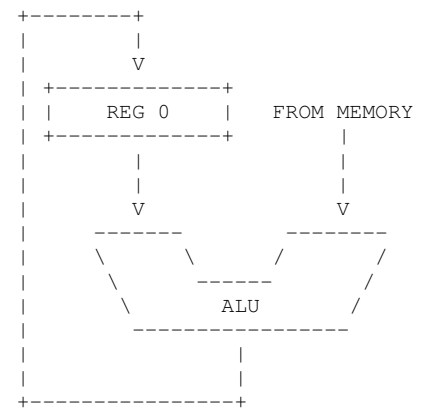
ACCUMULTOR CONTROL

  Besides selecting between 16 registers the accumulator may also have separate source and destination register file. This allows the source register to remain constant while only

updating the destination. The accumulator may also be pointed at the same register that X or Y is pointing at.

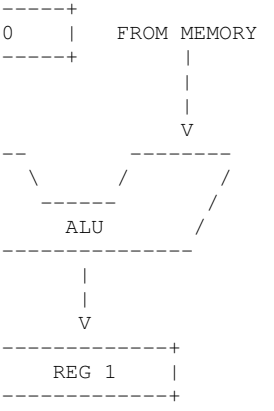
The two byte opcode $32 sets the source and destination register file. The immediate value bits [7:4] set the destination and bits [3:0] set the source.

ACCUMULATOR WITH SAME SOURCE AND DEST



ACCUMULATOR SOURCE = REG 0
ACCUMULATOR DESTINATION = REG 0

ALU OPERATION BETWEEN TWO REGISTERS



ACCUMULATOR SOURCE = REG 0
ACCUMULATOR DESTINATION = REG 1

INDEX REGISTERS

Index registers will have the same source and destination and are set with the two byte $42 immediate opcode.

Immediate value bits [7:4] = Y Register
Immediate value bits [3:0] = X Register

SEGMENT MAPPER

The CPU can "see" 64k of contiguous memory. To access more than 64k the segment mapper sets the upper 8 bits of the CPU's 24 bit address bus. There are four 32k segments that may be set independently.

Setting banks can be achieved by loading or executing ALU operations with the accumulator, X or Y register destinations pointing to one of the four segment register files.

REG 12	SEGMENT BANK
$0000-$3FFF	
Default Value: %00000000	

REG 13	SEGMENT BANK
$4000-$7FFF	
Default Value: %00000001	

REG 14	SEGMENT BANK
$8000-$BFFF	
Default Value: %00000010	

REG 15	SEGMENT BANK
$C000-$FFFF	
Default Value: %00000011	

Bits[1:0] = AddressOut[15:14]
Bits[7:2] = AddressOut[21:16]

REG 8	BANK 3 - 0 Access Control
Default Value: %01010101	

Bank 0
Bits[1:0]

Bank 1
Bits[3:2]

Bank 2
Bits[5:4]

Bank 3
Bits[7:6]

00 = ROM
01 = RAM
10 = Reserved
11 = reserved

CPU CONTROL REG 9

Bit 0 Skip internal cycle when set
Bit 1 Burst enable when set

BRANCH ALWAYS

Branch always $12 (BRA) is a two byte relative opcode. BRA will branch relative 127 forward or 128 back.

OPTIMIZED MEMORY ACCESS

Memory accesses repeat on a 32-cycle pattern. All reads to SDRAM are performed in burst of 4 and writes are single access. SRAM, ROM and register writes are single read and single write.

CPU CYCLES

When "skip internal cycles" is set in the CPU's control register the instruction timing is as follows.

```
Implied          = 1 cycle
Immediate        = 2
Relative         = 1
Push             = 2
Pull             = 2
ZeropageRMW      = 4
ZeropageIndexed  = 4
Zeropage writes  = 3
Zeropage reads   = 3
Absolute         = 4
AbsoluteRMW      = 5
Absolute indexed = 5
Jump             = 3
Jump Indirect    = 5
IndirectY Read   = 5
Indirecty RMW    = 6
IndirectX RMW    = 7
IndirectX Write  = 6
```

When the "burst" bit is enabled the CPU will fetch 8 bytes at a time and will used them with instructions that have sequential memory accesses. For example immediate instructions have one opcode byte and one data byte in sequential order. You can execute 4 immediate instructions per 1mhz cycle or 8 implied instructions.

The CPU will halt burst execution any time there is a non-sequential read or any write.

The multi-byte burst fetches are on 4 byte boundaries. For maximum performance instructions that execute in sequential order (immediate, implied, absolute..) should start at word 0, so 4 bytes of data/instructions can be executed in the same time 1 instruction would be executed.

Example:

```
All 4 instructions can execute before
next memory access
C000  LDA #$01
C002  ROR
C003  SEI
```

LDA Can not execute in 1 memory access, since it crosses a 4-word boundary. Instructions C005-C007 all execute next memory access.

```
C003  LDA #$01
C005  ROR
C006  SEI
C007  NOP
```

Access 1 execution stops with reads to non-immediate memory locations. Memory access 2 will be from zero page and execution stops. Access 3 will execute instructions C002-C003

```
C000  LDA $01
C002  ROR
C003  SEI
```

Access 1 execution stops with writes to non-immediate memory locations. Memory access 2 will be to zero page and execution stops. Access 3 will execute instructions C002-C003

```
C000  STA $01
C002  ROR
C003  SEI
```

Access 1 executes all cycles, except the read to $D020. Access 2 reads from $D020 and cycle 3 executes C003.

```
C000  LDA $D020
C003  SEI
```

Cycle 1 executes up to the actual write.
Cycle 2 does the write
Cycle 3 executes your self-modified code. (Bad. Bad. Boo. Hiss.)

```
C000 STA $C003
C003 ...
```

PALETTE

There are 16 adjustable colors. $0-f When chroma is set to 0 there is no modulation and can be used for white, black and grays.

DTV palette compatibility bit when set will distribute color 15 chroma across $10-$ff. This allows you to have 16 colors that can be changed with one write to a register.

Colors $0-$f are [chroma] [luma] from adjustable palette.

Colors $10-$ff are [chroma] [luma] from color decoder only.

Default PALLETTE

```
Color0Luma  = $0  black
Color1Luma  = $f  white
Color2Luma  = $6  Red
Color3Luma  = $e  cyan
Color4Luma  = $8  purple
Color5Luma  = $b  green
Color6Luma  = $6  blue
Color7Luma  = $f  yellow
Color8Luma  = $9  orange
Color9Luma  = $6  brown
Color10Luma = $b  light red
```

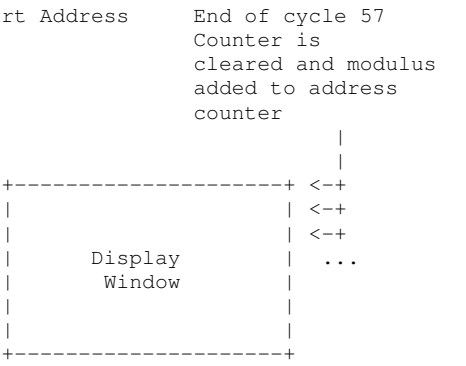
```
Color11Luma = $5  dark gray
Color12Luma = $7  medium gray
Color13Luma = $f  light green
Color14Luma = $a  light blue
Color15Luma = $a  light gray

Color0Chroma = $0 black
Color1Chroma = $0 white
Color2Chroma = $3 Red
Color3Chroma = $b cyan
Color4Chroma = $5 purple
Color5Chroma = $d green
Color6Chroma  = $8  blue
Color7Chroma  = $f  yellow
Color8Chroma  = $2  orange
Color9Chroma  = $2  brown
Color10Chroma = $3  light red
Color11Chroma = $0  dark gray
Color12Chroma = $0  medium gray
Color13Chroma = $d  light green
Color14Chroma = $9  light blue
Color15Chroma = $0  light gray


Default VIC Registers
----------------------------------------
raster_compare <= (others => '1')
light_pen_irq_en   <= '0'
sprite_sprite_irq_en<= '0';
sprite_background_irq_en     <= '0';
raster_irq_en                <= '0';
ExtendedRegEnableB           <= '0';
GBankA             <= (others => '0');
GBankB             <= (others => '0');
LinearAddressing   <= '0';
HiColor            <= '0';
ColorBankHigh      <= "0000";
ColorBankLow       <= "01110110";
border_color       <= (others => '0');
bkgnd0_color       <= (others => '0');
bkgnd1_color       <= (others => '0');
bkgnd2_color       <= (others => '0');
bkgnd3_color       <= (others => '0');
ExtendedRegKill    <= '0';
PAL                <= '0';
bmm                <= '0';
ecm                <= '0';
vm                 <= (others => '0');
cb                 <= (others => '0');
AlwaysSetToZero    <= '0';
c_sel              <= '0';
r_sel              <= '0';
x_scroll           <= (others => '0');
y_scroll           <= (others => '0');
den                <= '0';
clear_light_pen_irq<= '0';
clear_sprite_sprite_irq      <= '0';
clear_sprite_background_irq  <= '0';
clear_raster_irq             <= '0';
Sprite7Priority              <= '0';
Sprite6Priority              <= '0';
Sprite5Priority              <= '0';
Sprite4Priority              <= '0';
Sprite3Priority              <= '0';
Sprite2Priority              <= '0';
Sprite1Priority              <= '0';
Sprite0Priority              <= '0';
sprite_colora      <= (others => '0');
sprite_colorb      <= (others => '0');
sprite_0_color     <= (others => '0');
sprite_1_color     <= (others => '0');
```

```
sprite_2_color     <= (others => '0');
sprite_3_color     <= (others => '0');
sprite_4_color     <= (others => '0');
sprite_5_color     <= (others => '0');
sprite_6_color     <= (others => '0');
sprite_7_color     <= (others => '0');
PhaseAlternate     <= '0';
BurstRate <=
"000111000001001000101010"; --new 18
OldDTVCompatibility<= '0';
BorderOff          <= '0';
IrqTriggerCycle    <= "1000000"; --at
the end of cycle 64
SpriteBank         <= (others => '0')
LinearModuloA      <= (others => '0')
LinearStartA       <=
"00000000000000001110110"; --color bank
LinearStepA        <= (others => '0')
LinearModuloB      <= (others => '0')
LinearStartB       <= (others => '0')
LinearStepB        <= (others => '0')
OverScan           <= '0'
ColorDisable       <= '0'
CPUBadlineDisable  <= '0'
ChunkyEnable       <= '0'
LineAdjust         <= "00001101";
PhaseAdjust        <= (others => '0')
mcm                <= '0'
```

MODULO PROGRAMMING

The DTV contains 3 locations where address are calculated with modulus counters. VIC, DMA and Blitter.

Modulus counters can be used to format data fetches from a contiguous memory.



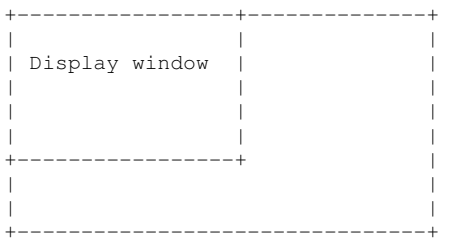
Displaying or moving portions of images that are bigger than the display window can be achieved by properly setting the modulus values.

Example:
320x200 display window
800x400 image

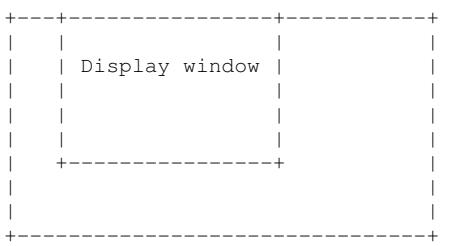
Set line count to 320, which is the number of pixels in a scan line.

Set modulus to 800(total pixels in image) - 320(scan line pixels)

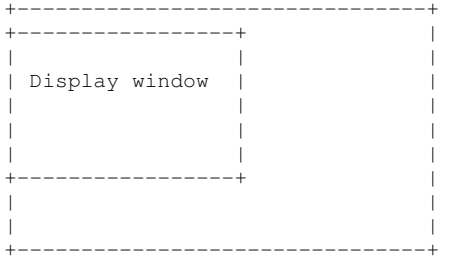
Start address at beginning of image.



Moving start address along the first line will scroll horizontally in the image.

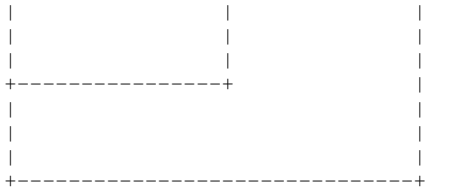


Moving start by image horizontal size (In this example 800) will scroll the image down by 1 line.

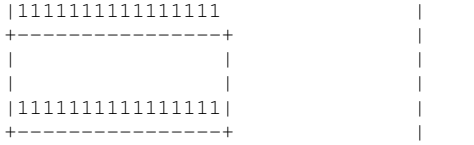


Unlimited scrolling in 640 x 400 buffer.

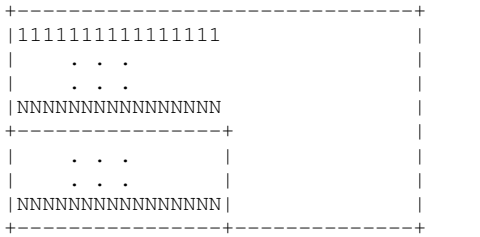
Scroll = 0,0



Scrolled down to 0,7
5120 bytes of new data plotted by blitter, placed at bottom of view port and exact copy placed above view port.



Scrolled down to 0,319
5120 bytes of new data plotted by blitter, placed at bottom of view port and exact copy placed above view port.



If y position of view port = 320 then next scroll will jump back to y position = 0. This allows unlimited y scrolling.

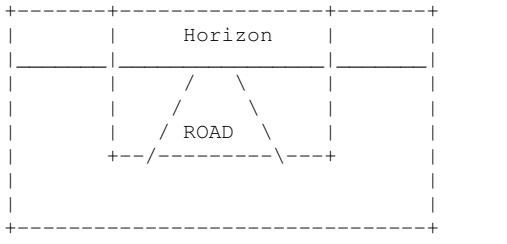
The same concept works in the x-axis and will jump to x position = 0 when x position = 640.

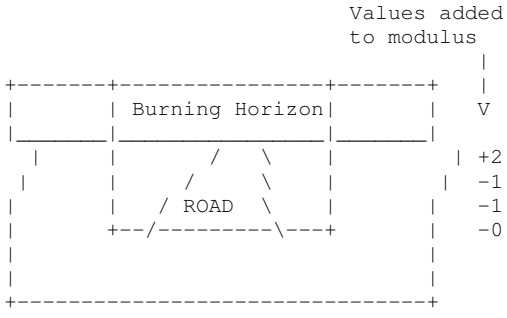
Combining both x and y are possible allowing scrolling in any direction with only 8320 bytes of blits per 8 pixel plotting.

Moving by multiples of lines and pixels can be used to scroll in all directions.

VIC Start addresses are loaded on line 49(line 11 in over-scan mode).

Changing the modulus on every scan line can generate effects like twisting roads.





Plotting BOB's into a twisted display would require the blit start address be changed by the opposite amount on every line modified

RATINGS

VDD	3.3v
MAX	3.6v
Operating Temp	0C - 70C
High Level Input	1.7v
Low Level Input	1.1v
Schmitt hysteresis	.6v
Capacitance Input (die)	2.4pF
Capacitance Output (die)	5.6pF
Capacitance Bidir (die)	6.6pF

PIN ASSIGNMENTS

Input Pins

Name		
ATNIn	5v Tol	
Clk32mhz	5v Tol	
KeyboardClk	5v Tol	
KeyboardData	5v Tol	
LightPen	5v Tol	Schmitt
nDMA	5v Tol	
nReset	5v Tol	
nSRAMSelect	5v Tol	
nSVideo	5v Tol	

Output Pins

Name		
AddressBufferDir	LVTTL	
CPUAddressEn	LVTTL	
CSync	LVTTL	
Chroma[0]	LVTTL	12ma
Chroma[1]	LVTTL	12ma
Chroma[2]	LVTTL	12ma
Chroma[3]	LVTTL	12ma
Clk1mhzEn	LVTTL	
DataBufferDir	LVTTL	
DataBuffer_nOE	LVTTL	
IECATN	LVTTL	
Luma[0]	LVTTL	12ma
Luma[1]	LVTTL	12ma
Luma[2]	LVTTL	12ma
Luma[3]	LVTTL	12ma
SDRAMCLK	LVTTL	Low Skew
SDRAMLDM	LVTTL	
SDRAMUDM	LVTTL	
SDRAM_nCS	LVTTL	
SDRAMnCAS	LVTTL	
SDRAMnRAS	LVTTL	
Voice1Sigma	LVTTL	12ma
Voice2Sigma	LVTTL	12ma
Voice3Sigma	LVTTL	12ma
VolumeSigma	LVTTL	12ma
nIORd	LVTTL	
nRAMCS	LVTTL	
nROMCs	LVTTL	
nWrite	LVTTL	

```
------------------------
Bidir Pins
------------+-------+--
Name        ; Pin # ; I
------------+-------+--
Address[0]  ; LVTTL ; 5V Tol
Address[10] ; LVTTL ; 5V Tol
Address[11] ; LVTTL ; 5V Tol
Address[12] ; LVTTL ; 5V Tol
Address[13] ; LVTTL ; 5V Tol
Address[14] ; LVTTL ; 5V Tol
Address[15] ; LVTTL ; 5V Tol
Address[16] ; LVTTL ; 5V Tol
Address[17] ; LVTTL ; 5V Tol
Address[18] ; LVTTL ; 5V Tol
Address[19] ; LVTTL ; 5V Tol
Address[1]  ; LVTTL ; 5V Tol
Address[20] ; LVTTL ; 5V Tol
Address[21] ; LVTTL ; 5V Tol
Address[2]  ; LVTTL ; 5V Tol
Address[3]  ; LVTTL ; 5V Tol
Address[4]  ; LVTTL ; 5V Tol
Address[5]  ; LVTTL ; 5V Tol
Address[6]  ; LVTTL ; 5V Tol
Address[7]  ; LVTTL ; 5V Tol
Address[8]  ; LVTTL ; 5V Tol
Address[9]  ; LVTTL ; 5V Tol
Data[0]     ; LVTTL ; 5V Tol
Data[1]     ; LVTTL ; 5V Tol
Data[2]     ; LVTTL ; 5V Tol
Data[3]     ; LVTTL ; 5V Tol
Data[4]     ; LVTTL ; 5V Tol
Data[5]     ; LVTTL ; 5V Tol
Data[6]     ; LVTTL ; 5V Tol
Data[7]     ; LVTTL ; 5V Tol
IECClk      ; LVTTL ; 5V Tol ;Pullup
IECData     ; LVTTL ; 5V Tol ;Pullup
JoyA[0]     ; LVTTL ; 5V Tol ;Pullup
JoyA[1]     ; LVTTL ; 5V Tol ;Pullup
JoyA[2]     ; LVTTL ; 5V Tol ;Pullup
JoyA[3]     ; LVTTL ; 5V Tol ;Pullup
JoyA[4]     ; LVTTL ; 5V Tol ;Pullup
JoyA[5]     ; LVTTL ; 5V Tol ;Pullup
JoyB[0]     ; LVTTL ; 5V Tol ;Pullup
JoyB[1]     ; LVTTL ; 5V Tol ;Pullup
JoyB[2]     ; LVTTL ; 5V Tol ;Pullup
JoyB[3]     ; LVTTL ; 5V Tol ;Pullup
JoyB[4]     ; LVTTL ; 5V Tol ;Pullup
JoyB[5]     ; LVTTL ; 5V Tol ;Pullup
PA2         ; LVTTL ; 5V Tol ;Pullup
Paddle      ; LVTTL ; 5V Tol ;Schmitt
USR[0]      ; LVTTL ; 5V Tol ;Pullup
USR[1]      ; LVTTL ; 5V Tol ;Pullup
USR[2]      ; LVTTL ; 5V Tol ;Pullup
USR[3]      ; LVTTL ; 5V Tol ;Pullup
USR[4]      ; LVTTL ; 5V Tol ;Pullup
USR[5]      ; LVTTL ; 5V Tol ;Pullup
USR[6]      ; LVTTL ; 5V Tol ;Pullup
USR[7]      ; LVTTL ; 5V Tol ;Pullup
nIRQ        ; LVTTL ; 5V Tol ;Pullup
nNMI        ; LVTTL ; 5V Tol ;Pullup


Input Pins
-----------------+-----
Name             ;
-----------------+-----

CPUDataPort[4]
```

  Input port to bit 4 of register $0000
and $0001

Clk32mhz
  ~32Mhz clock input (PAL 31.36, NTSC
32.64)

KeyboardClk
  PS/2 Keyboard clock

KeyboardData
  PS/2 Keyboard Data

LightPen
  Active low lightpen trigger

nDMA
  External DMA (active low). Tristates
data, address and nWrite.

nReset
  Global reset(active low) synchronized
to system clock.

nSRAMSelect
  Disables SDRAM when low and nRAMCS
activated.

nSVideo
  Selects Separate luma and chroma when
low. (internally mixed when high)

Output Pins
-----------------+-----
Name   ;
-----------------+-----

CPUAddressEn
  Indicates CPU address cycle when
high.

CSync
  Drives high during non sync times to
set black level.

Chroma[0]
Chroma[1]
Chroma[2]
Chroma[3]
  Chrominance output during nSVIDEO =
gnd, otherwise composite.

Clk1mhzEn
  32mhz strobe at CPU execution.

DataBufferDir
  Controls direction of buffers if long
external bus used.

DataBuffer_nOE
  Controls output of buffers if long
external bus used.

IECATN
  Disk serial attention.

Luma[0]
Luma[1]
Luma[2]
Luma[3]

  Luminace output during nSVIDEO = gnd,
otherwise composite.

SDRAMCLK
SDRAMLDM
SDRAMUDM
SDRAM_nCS
SDRAMnCAS
SDRAMnRAS
  SDRAM control signals.

Voice1Sigma
Voice2Sigma
Voice3Sigma
VolumeSigma
  Sigma converts. Must be run through
a lowpass filter.

nIORd
  Active low read

nRAMCS
  Active low SRAM chips elect when
nSRAM = gnd

nROMCs
  Active low ROM chip select.

nWrite
  Global write for SRAM, Flash and
SDRAM. This should be buffered if used
on long external bus

Bidir Pins
------------+-------+--
Name   ; Pin # ; I
------------+-------+--
Address[0]
Address[10]
Address[11]
Address[12]
Address[13]
Address[14]
Address[15]
Address[16]
Address[17]
Address[18]
Address[19]
Address[1]
Address[20]
Address[21]
Address[2]
Address[3]
Address[4]
Address[5]
Address[6]
Address[7]
Address[8]
Address[9]
  Address should be buffered if used on
long external bus.

Data[0]
Data[1]
Data[2]
Data[3]
Data[4]
Data[5]
Data[6]

Data[7]
  Data should be buffered if used on
long external bus.

IECClk
  Disk clock.

IECData
  Disk Data.

JoyA[0]
JoyA[1]
JoyA[2]
JoyA[3]
JoyA[4]
JoyA[5]
JoyB[0]
JoyB[1]
JoyB[2]
JoyB[3]
JoyB[4]
JoyB[5]
  Open collector joystick ports.

PA2
  CIA PA line.

Paddle
  Charge dump analog A/D converter.

USR[0]
USR[1]
USR[2]
USR[3]
USR[4]
USR[5]
USR[6]
USR[7]
  Open collector user port pins

nIRQ
  Negative assert IRQ (bidir!)

nNMI
  Negative assert NMI (bidir!)

Pin Locations
----
160 <CORNER>
159 VSS
158 VDD
157 VSS
156 VDD
155 TMODE
154 USR_0
153 USR_1
152 VSS
151 VDD
150 USR_2
149 USR_3
148 USR_4
147 USR_5
146 VSS
145 VDD
144 USR_6
143 USR_7
142 Luma_0
141 Luma_1
140 VSS
139 VDD

138 Luma_2
137 Luma_3
136 JoyB_0
135 JoyB_1
134 VSS
133 VDD
132 JoyB_2
131 JoyB_3
130 JoyB_4
129 JoyB_5
128 VSS
127 VDD
126 JoyA_0
125 JoyA_1
124 JoyA_2
123 JoyA_3
122 VSS
121 <CORNER>
120 <CORNER>
119 VDD
118 VSS
117 VDD
116 JoyA_4
115 JoyA_5
114 Data_0
113 Data_1
112 VSS
111 VDD
110 Data_2
109 Data_3
108 Data_4
107 Data_5
106 VSS
105 VDD
104 Data_6
103 Data_7
102 Chroma_0
101 Chroma_1
100 VSS
99 VDD
98 Chroma_2
97 Chroma_3
96 Address_0
95 Address_1
94 VSS
93 VDD
92 Address_2
91 Address_3
89 Address_5
88 VSS
87 VDD
86 Address_6
85 Address_7
84 Address_8
83 Address_9
82 VSS
81 <CORNER>
80 <CORNER>
79 VDD
78 VSS
77 VDD
76 Address_10
75 Address_11
74 Address_12
73 Address_13
72 VSS
71 VDD
70 Address_14
69 Address_15
68 Address_16
67 Address_17

66 VSS
65 VDD
64 Address_18
63 Address_19
62 Address_20
61 Address_21
60 VSS
59 VDD
58 Paddle
57 PA2
56 nNMI
55 nIRQ
54 VSS
53 VDD
52 IECData
51 IECClk
50 Clk1mhzEn
49 CPUAddressEn
48 VSS
47 VDD
46 VolumeSigma
45 Voice3Sigma
44 Voice2Sigma
43 Voice1Sigma
42 VSS
41 <CORNER>
40 <CORNER>
39 VDD
38 VSS
37 VDD
36 CSync
35 nRAMCS
34 nROMCs
33 nWrite
32 VSS
31 VDD
30 nIORd
29 SDRAMnRAS
28 SDRAMnCAS
27 IECATN
26 VSS
25 VDD
24 SDRAMLDM
23 SDRAMUDM
22 SDRAM_nCS
21 DataBuffer_nOE
20 VSS
19 VDD
18 DataBufferDir
17 AddressBufferDir
16 clko
15 Clk32mhz
14 SDRAMCLK
13 VSS
12 VDD
11 nSVideo
10 nSRAMSelect
9   LightPen
8   ATNIn
7   VSS
6   VDD
5   nDMA
4   nReset
3   KeyboardData
2   KeyboardClk
1 <CORNER>

PAD COORDINATES

Please see the die pad coordinates
below. They're all measured

from the center of the die, so X coordinates are negative to the left of center and positive to the right, while Y coordinates are negative below center and positive above.

Also, the coordinates are scaled by 10 for some reason, including the die size, itself, which should be 3.785mm x 3.759mm. Die pad #1 is given as "-15.109 17.335", but this translates to X being 1.5109mm left of center and Y being 1.7335mm above center.

#Pads list
1 -15.109 17.335
2 -14.311 17.335
3 -13.513 17.335
4 -12.715 17.335
5 -11.917 17.335
6 -11.119 17.335
7 -10.321 17.335
8 -9.523 17.335
9 -8.725 17.335
10 -7.927 17.335
11 -7.129 17.335
12 -6.331 17.335
13 -5.533 17.335
14 -4.735 17.335
15 -3.937 17.335
16 -3.139 17.335
17 -2.341 17.335
18 -1.543 17.335
19 -0.745 17.335
20 0.053 17.335
21 0.851 17.335
22 1.649 17.335
23 2.447 17.335
24 3.245 17.335
25 4.043 17.335
26 4.841 17.335
27 5.639 17.335
28 6.437 17.335
29 7.235 17.335
30 8.033 17.335
31 8.831 17.335
32 9.629 17.335
33 10.427 17.335
34 11.225 17.335
35 12.023 17.335
36 12.821 17.335
37 13.619 17.335
38 14.417 17.335
39 15.215 17.335
40 16.013 17.335
41 17.335 15.095
42 17.335 14.297
43 17.335 13.499
44 17.335 12.701
45 17.335 11.903
46 17.335 11.105
47 17.335 10.307
48 17.335 9.509
49 17.335 8.711
50 17.335 7.913
51 17.335 7.115
52 17.335 6.317
53 17.335 5.519
54 17.335 4.721

55 17.335 3.923
56 17.335 3.125
57 17.335 2.327
58 17.335 1.529
59 17.335 0.731
60 17.335 -0.066
61 17.335 -0.864
62 17.335 -1.662
63 17.335 -2.460
64 17.335 -3.258
65 17.335 -4.056
66 17.335 -4.854
67 17.335 -5.652
68 17.335 -6.450
69 17.335 -7.248
70 17.335 -8.046
71 17.335 -8.844
72 17.335 -9.642
73 17.335 -10.440
74 17.335 -11.238
75 17.335 -12.036
76 17.335 -12.834
77 17.335 -13.632
78 17.335 -14.430
79 17.335 -15.228
80 17.335 -16.026
81 15.095 -17.335
82 14.297 -17.335
83 13.499 -17.335
84 12.701 -17.335
85 11.903 -17.335
86 11.105 -17.335
87 10.307 -17.335
88 9.509 -17.335
89 8.711 -17.335
90 7.913 -17.335
91 7.115 -17.335
92 6.317 -17.335
93 5.519 -17.335
94 4.721 -17.335
95 3.923 -17.335
96 3.125 -17.335
97 2.327 -17.335
98 1.529 -17.335
99 0.731 -17.335
100 -0.066 -17.335
101 -0.864 -17.335
102 -1.662 -17.335
103 -2.460 -17.335
104 -3.258 -17.335
105 -4.056 -17.335
106 -4.854 -17.335
107 -5.652 -17.335
108 -6.450 -17.335
109 -7.248 -17.335
110 -8.046 -17.335
111 -8.844 -17.335
112 -9.642 -17.335
113 -10.440 -17.335
114 -11.238 -17.335
115 -12.036 -17.335
116 -12.834 -17.335
117 -13.632 -17.335
118 -14.430 -17.335
119 -15.228 -17.335
120 -16.026 -17.335
121 -17.335 -15.109
122 -17.335 -14.311
123 -17.335 -13.513
124 -17.335 -12.715
125 -17.335 -11.917

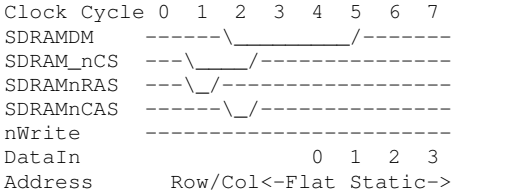
```
126 -17.335 -11.119
127 -17.335 -10.321
128 -17.335 -9.523
129 -17.335 -8.725
130 -17.335 -7.927
131 -17.335 -7.129
132 -17.335 -6.331
133 -17.335 -5.533
134 -17.335 -4.735
135 -17.335 -3.937
136 -17.335 -3.139
137 -17.335 -2.341
138 -17.335 -1.543
139 -17.335 -0.745
140 -17.335 0.053
141 -17.335 0.851
142 -17.335 1.649
143 -17.335 2.447
144 -17.335 3.245
145 -17.335 4.043
146 -17.335 4.841
147 -17.335 5.639
148 -17.335 6.437
149 -17.335 7.235
150 -17.335 8.033
151 -17.335 8.831
152 -17.335 9.629
153 -17.335 10.427
154 -17.335 11.225
155 -17.335 12.023
156 -17.335 12.821
157 -17.335 13.619
158 -17.335 14.417
159 -17.335 15.215
160 -17.335 16.013
```

SDRAM Row/Column Address mapping

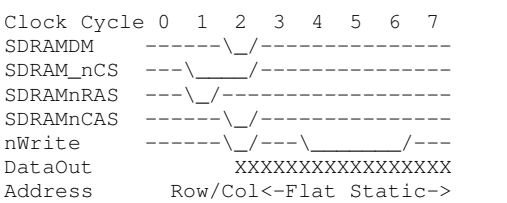
Address bus during row accesses
(Address[15:12], Address[19:16],
Address[15:8])

Address bus during column accesses
(Address[15:12], Address[19],'1',"00",
Address[7:0])

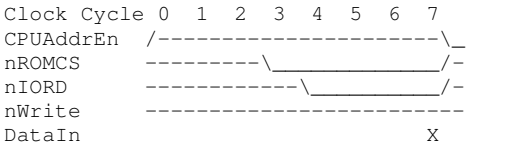
SDRAM Read



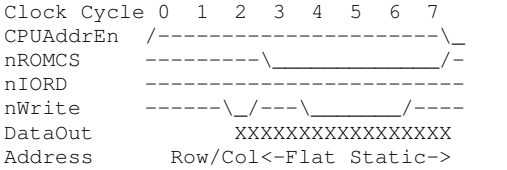
SDRAM Write



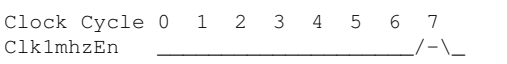
SRAM/ROM Read



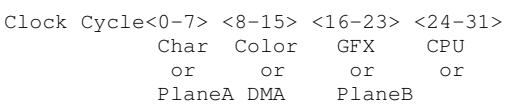
SRAM/Flash Write



Clk1mhzEn during CPU access.



32 cycle accesses(1mhz)



Clk1mhzEn strobes during cycle 31.

A note on global set and reset :

When TMODE is high ; the chip is in "test" mode and now nNMI and nIRQ are active high global set and active high global reset pins respectively.

Toggling these 2 pins when TMODE is high will put all the flops in a known and defined state.

Startup User Port Bits

On power the user port is polled by the boot ROM and video modes are set by the state the bits.

0 PAL/NTSC Line timing (1=PAL 63cycles/line)
1 PAL/NTSC burst alternation enable (1=alternate)
2 Saturation 0
3 Saturation 1
4 Burst lock enable (1=enable)
5 Burst lock type
6 Line timing fine tune (1=PAL)
7 PAL/NTSC burst select (1=PAL)

D6510bit[4] 1=old xtals, 0=new

Original -

```
;8 bit multiply with 16 bit product
; MULND(8) * MULR(8) = PROD(16)

MULTIPLY:  lda     #$00   ;Clear lower
half of product
```

```
           sta     PROD+1 ;Clear upper
half of product
           ldx     #8     ;Set count
SHIFT:     asl     a      ;Shift
product left one bit
           rol     PROD+1
           asl     MULR   ;Shift
multiplier left
           bcc     CHCNT  ;No addition
if next bit is zero
           clc
           adc     MULND
           bcc     CHCNT
           inc     PROD
CHCNT      dex
           bne     SHIFT
           sta     PROD


---------------------------------------
Using register bank -

; macros need to be fancier...
.MACRO      AS0D0  .BYT $32, %00000000
.MACRO      AS3D3  .BYT $32, %00110011
.MACRO      AS4D4  .BYT $32, %01000100

MULTIPLY:  lda     #$00   ;Clear lower
half of product
           AS3D3          ;Use reg 3 as
storage
           lda     #$00   ;Clear upper
half of product
           AS4D4          ;Use reg 4 as
storage
           lda     MULR
           ldx     #8     ;Set count
SHIFT:     AS0D0          ;Back to reg
0
           asl     a      ;Shift
product left one bit
           AS3D3
           rol
           AS4D4
           asl            ;Shift
multiplier left
           bcc     CHCNT  ;No addition
if next bit is zero
           clc
           adc     MULND  ;Can't use
extra reg's for adds :(
           bcc     CHCNT
           AS3D3
           clc
           adc     #1     ;This is an
"inc a" - should have added that op
code like the c02 has :)
CHCNT      dex
           bne     SHIFT
           AS3D3
           sta     PROD+1
           AS0D0
           sta     PROD

 This can be optimized more by -
pointing the accumulator to the y
register so you can iny the
accumulator, by pre-decrementing the
loop count, and by self modifying the
adc multnd (changing to an immediate
add) which is *naughty*.
```

Have fun hacking,
Jeri