# AVIC.v – Audio / Video Interface Circuit

## Overview

This is a full featured audio / video display controller that integrates a bitmap display controller with several coprocessing elements to enhance system performance. It contains a dedicated character blitter supporting multiple font sizes which allows a simple software interface. It also contains a general purpose blitter component allowing fast graphics transfers and manipulation. Also present is a co-processor capable of updating the register set based on the scan position.

The display controller interfaces as a slave device with the cpu via a 64-bit bus. It interfaces to the memory system as a bus master using a 128-bit bus.

## Features

800 x 600 x 16 bpp display resolution

400 x 300 x 16 bpp lowres mode

programmable sync generator

graphics command queue

character blitting, variable bitmap font size from 1 x 1 to 32 x 32

point plot / line draw / rectangle / triangle / curve draw acceleration

general purpose blitter – three sources, one destination

32 hardware cursors / sprites

4 channel audio output with independent timing

am / fm modulation

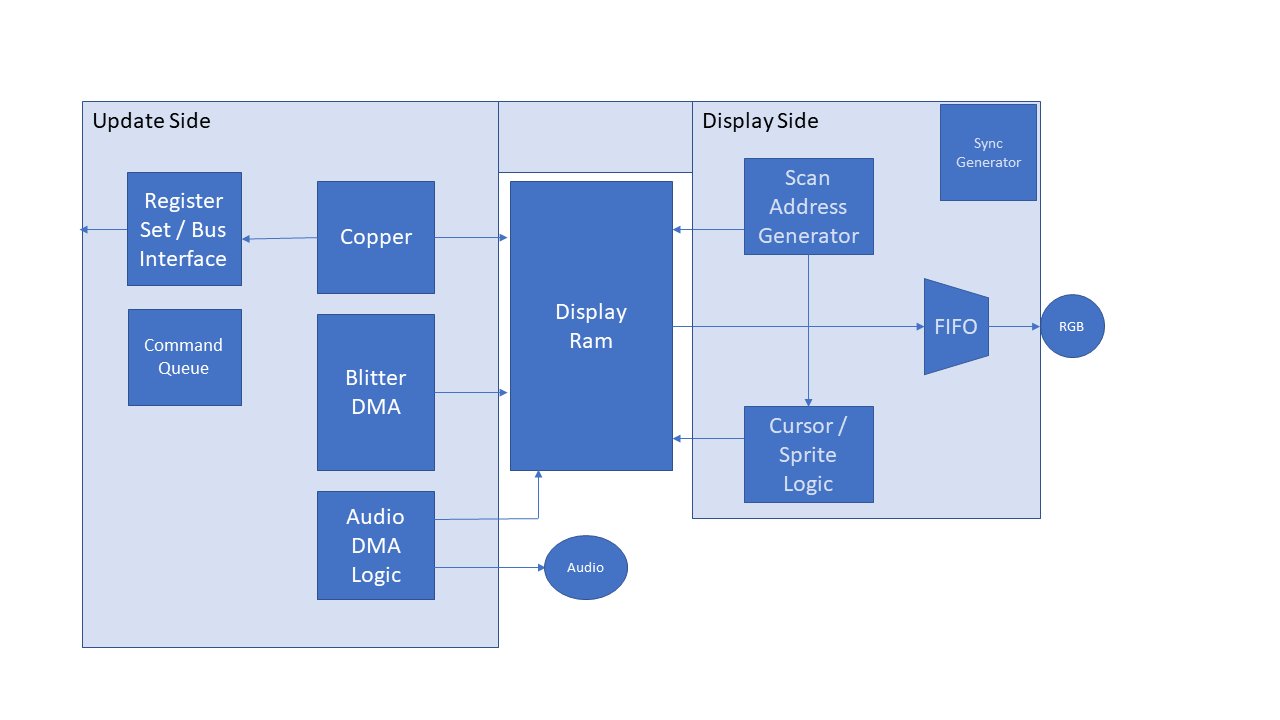
1 channel audio input

## Documentation Notes

Some registers are illustrated as if they were 32 bits in size for easier readability. The lower half of a register is presented first, followed by the upper half. These are given addresses that are four bytes apart. However, registers in the circuit are really 64-bit.

## General Organization

The following block diagram shows the organization of the AVIC core.



## Clocks

The video clock rate is 40MHz for an 800x600 VGA display. The bus clock for interface to the cpu is independent of the video clock.

## Display Format

The display controller supports a default format of 800 x 600. The primary reason for this selection is the amount of memory bandwidth dedicated to display purposes. The controller also supports a lower resolution mode of 400 x 300.

## Color Depth

The controller uses sixteen bits per pixel color depth (RGB555) with an extra sixteenth bit to indicate alpha blending or background transparency.

## Display Memory

The core typically expects a memory of least 64k words by 128 bits wide (1MB) to economize on the pixel format of RGB555, to allow for 16 bit audio samples and make it possible for the processor to use the memory in a general purpose fashion. The controller has a bus master port separate from the slave port used to update the controller’s registers. Typically, memory will be shared between the controller and processor through some sort of bus arbitrator. The controller uses a 128 bit path to memory to maximize the rate of transfer.

# Fonts

The controller features a dedicated text blitter than can handle either fixed or varying width fonts. Multiple fonts may be supported through the use of font tables.

## Font Table

The core supports the use of multiple fonts onscreen at the same time via a font table. The font table is a table of information describing basic characteristics of the font and where to find further font information for a given number of fonts. The font table is in memory and indexed by the font id register to select a font to work with. The font table is a collection of font table entries each of which has the following layout:

Font Table Entry

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Offset | Fields | | | | Use |
| 0 | Address31..0 | | | | Address of character bitmaps |
| 4 | fixed1 | width5 | height5 | ~21 | width and height |
| 8 | Address31..0 | | | | Glyph width table address |
| C | ~32 | | | |  |

Each font table entry is sixteen bytes in size. The font id (register $DE8) is used to index into this table. Setting the font table id register tells the core which font to use. The core then looks up the font information from the table.

The location of the font table in the controller’s memory is specified in the font table address register - register $DE0.

## Glyph Width Table

If the font is a fixed width font then no further table lookup is required, and the font width is determined by the width field in the font table entry. For fonts with characters whose bitmaps vary in width there is an additional table used to describe the width of the bitmap for the character in memory. Thus, variable width character fonts are supported.

## Font Table Address

This register determines where in the controller’s memory the font table is located.

The glyph width table is an array of bytes that specify the character width for each character in the font. The address of the glyph width table is found in the font table entry for the font.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 63 0 | | |  |
| $DE0 | ~32 | Offset31..0 | | font table address |
| $DE8 | ~48 | | Font id16 |  |

## Font ID

This register selects the working font. The core uses this id to determine which entry of the font table to use to lookup font attributes.

# Registers

### Target (screen bitmap) Base

This register determines where in the controller’s memory the target area (bitmap) for the screen display is located. The bitmap base address may be located at any address to allow smooth scrolling the display simply by changing the base address.

|  |  |  |
| --- | --- | --- |
|  | 31 0 |  |
| $FC0 | Offset31..0 | Target bitmap base low |
| $FC4 | ~32 | Target bitmap base high |

### Target Size

This register controls the size of the target bitmap. Note that the target bitmap may be larger than the screen size. The size of the screen is controlled by the sync generator registers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 63 0 | | |  |
| $FD0 | ~32 | Width16 | Height16 | Target Size |

## Command Queue

### Overview

The controller uses a command queue to allow the main processor to continue operating asynchronously to graphics processing. Some graphics operations may require hundreds or thousands of clock cycles to complete. The processor is not stalled while these operations take place. The command queue holding register is written with desired parameters and command code. Then a value is written to the queue trigger register. Writing the queue trigger register causes the holding registers to be placed in a queue. Graphics commands are processed from the queue in first-in first-out order. Note that the holding registers retain their value after being queued so that another command may be queued by modifying only the values that need to change.

The queue has a fixed number of entries 1023 to be exact. More items should not be placed in the queue until there is available queue space. The number of entries currently queue can be read from the queue count register.

### Command Queue Count Register

This register contains the number of entries currently in the command queue. As commands are executed this number will go down. The queue count register should be checked before sending graphics commands.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 63 0 | |  |
| $DD0 | ~48 | Count16 | Command count |

### Command Queue Holding Register

|  |  |  |  |
| --- | --- | --- | --- |
|  | 31 0 | |  |
| $DC0 | Parameter31..0 | | Command parameter |
| $DC4 | ~24 | Command Code8 | Command Code |

### Command Queue Trigger Register

|  |  |  |
| --- | --- | --- |
|  | 63 0 |  |
| $DC8 | ~64 |  |

### Command Register

The command code register specifies which graphics operation to perform. Write to the queue trigger register ($DC8) to queue the command.

### Summary of Graphics Commands

|  |  |  |
| --- | --- | --- |
| Command8 | Operation Performed | Parameters / Set first |
| 0 | Draw character bitmap | X0,Y0 fgcolor, bkcolor, char code |
| 1 | Plot point | X0,Y0 bkcolor |
| 2 | Draw line | x0,y0 x1,y1 bkcolor |
| 3 | Draw filled Rectangle | x0,y0 x1,y1 bkcolor |
| 6 | Draw filled triangle | x0,y0 x1,y1 x2,y2, bkcolor |
| 8 | Draw Bezier curve | x0,y0 x1,y1 x2,y2, bkcolor, fill code |
| 12 | Set pen color | RGB888 value |
| 13 | Set fill color | RGB888 value |
| 14 | Set alpha | 16-bit value |
| 16 | Set X0 | 32-bit fixed point (16,16) value |
| 17 | Set Y0 | 32-bit fixed point (16,16) value |
| 18 | Set Z0 | 32-bit fixed point (16,16) value |
| 19 | Set X1 | 32-bit fixed point (16,16) value |
| 20 | Set Y1 | 32-bit fixed point (16,16) value |
| 21 | Set Z1 | 32-bit fixed point (16,16) value |
| 22 | Set X2 | 32-bit fixed point (16,16) value |
| 23 | Set Y2 | 32-bit fixed point (16,16) value |
| 24 | Set Z2 | 32-bit fixed point (16,16) value |
| 25 | Set Clip X0 | 16-bit value (whole number) |
| 26 | Set Clip Y0 | 16-bit value (whole number) |
| 27 | Set Clip X1 | 16-bit value (whole number) |
| 28 | Set Clip Y1 | 16-bit value (whole number) |
| 29 | Set clip enable / disable | 1 bit value |
| 32 | Set aa transform coefficient | 32-bit fixed point (16,16) value |
| 33 | Set ab transform coefficient | 32-bit fixed point (16,16) value |
| 34 | Set ac transform coefficient | 32-bit fixed point (16,16) value |
| 35 | Set at transform coefficient | 32-bit fixed point (16,16) value |
| 36 | Set ba transform coefficient | 32-bit fixed point (16,16) value |
| 37 | Set bb transform coefficient | 32-bit fixed point (16,16) value |
| 38 | Set bc transform coefficient | 32-bit fixed point (16,16) value |
| 39 | Set bt transform coefficient | 32-bit fixed point (16,16) value |
| 40 | Set ca transform coefficient | 32-bit fixed point (16,16) value |
| 41 | Set cb transform coefficient | 32-bit fixed point (16,16) value |
| 42 | Set cc transform coefficient | 32-bit fixed point (16,16) value |
| 43 | Set ct transform coefficient | 32-bit fixed point (16,16) value |
| 254 | Reset command queue |  |
| 255 | NOP |  |

### Command #0 – Draw Character Bitmap

Command #0 will draw the character specified by the low order 16-bits of the parameter portion of the command holding register at previously set X0, Y0 co-ordinate registers, using a previously set foreground and background color.

### Command #1 – Plot Point

Command #1 will plot a point on the target bitmap using previously set X0, Y0 co-ordinates in the previously set background color. The raster operation used to set the point is determined from bits 4 to 7 of the command parameter.

### Command #2 – Draw Line

Command #2 will draw a line on the target bitmap using the previously set X0, Y0, X1, and Y1 co-ordinates in the previously set background color. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command #3 – Draw Filled Rectangle

Command #3 will draw a filled rectangle on the target bitmap using the previously set X0, Y0, X1, and Y1 co-ordinates in the previously set background color. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command #6 – Draw Triangle

Command#6 will draw a filled triangle on the target bitmap using the previously set X0, Y0, X1, Y1, and X2, Y2 co-ordinates. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command#8 – Draw Bezier Curve

Command#8 will draw a filled or unfilled Bezier curve on the target bitmap using previously set co-ordinates. The least significant two bits of the command parameter determine how the curve is filled. The raster operation used is determined from bits 4 to 7 of the command parameter.

### Command #12 – Set Pen Color

This command will set the graphics pen color for subsequent operations using the pen color. The pen color set must be a RGB888 value.

### Command #13 – Set Fill Color

This command will set the fill color for subsequent operations using the fill color. The fill color set must be an RGB888 value.

### Command #14 – Set Alpha

This command sets the alpha value for subsequent operations using an alpha value. The alpha value is a 16-bit number.

### Command #16 – Set X0

This command sets the graphics X0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #17 – Set Y0

This command sets the graphics Y0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #18 – Set Z0

This command sets the graphics Z0 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #19 – Set X1

This command sets the graphics X1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #20 – Set Y1

This command sets the graphics Y1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #21 – Set Z1

This command sets the graphics Z1 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #22 – Set X2

This command sets the graphics X2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #23 – Set Y2

This command sets the graphics Y2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #24 – Set Z2

This command sets the graphics Z2 position. The position is specified as a 32-bit fixed point number with 16 fractional bits and 16 whole bits.

### Command #25 – Set ClipX0

This command sets the clipping region X0 co-ordinate. The parameter value is a 16-bit integer (whole number only).

### Command #26 – Set ClipY0

This command sets the clipping region Y0 co-ordinate. The parameter value is a 16-bit integer.

### Command #27 – Set ClipX1

This command sets the clipping region X1 co-ordinate. The parameter value is a 16-bit integer (whole number only).

### Command #28 – Set ClipY1

This command sets the clipping region Y1 co-ordinate. The parameter value is a 16-bit integer.

### Command #29 – Set Clip Enable / Disable

This command determines whether clipping of graphics output is enabled or disabled. If the parameter is 1 clipping is enabled, if the parameter is 0 then clipping is disabled. Note that all graphics are automatically clipped according to the target width and height which cannot be disabled. This setting controls the clip region defined by clipping co-ordinates clipx0, clipy0, clip x1, clip y1.

### Command #254 – Reset Command Queue

This command empties out the command queue. Any command in the queue are not performed.

### Command #255 – NOP

This command is a no-operation.

# Cursor / Sprite Control

The controller has 32 hardware cursors or sprites. The cursors may be up to 32 pixels wide and 512 pixels high. The cursors all share a common 256 entry color palette. However, each cursor may have its own set of colors. A sprite by itself may use three different colors simultaneously plus transparency. Note that the cursors are effectively 32 pixels wide, but pixels may be set to be transparent, so the apparent size of the cursor looks smaller.

### Cursor Color Palette

The cursor color palette has 256 entries each of which is a 64-bit vector including additional attributes besides just the color. Attributes include alpha blending, reverse video and flashing.

The registers are organized into groups of four, a group of four registers present for each of the sprites. Thus, each sprite can have a different set of colors from other sprites. Only three of the four registers are used. (The color code 00 is transparent). The registers are further organized into sixteen groups of sixteen for linked sprites. A set of 16 color registers is used when sprites are linked together. The first group establishes a set of colors which are shared between sprite 0 and 1. The second group is shared between sprites 2 and 3, and so on. Note that color palette entry #0 is never used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $000 | ~26 | i | f | rate4 | Alpha8 | RGB88824 | Color0 |  |
| $008 to $7F8 | 255 more registers | | | | | |  |  |

**Alpha8** determines how much of the cursor color is present in the output. A value of zero causes the cursor to be fully output. A value of all ones will make the cursor invisible. Alpha blending can be used to create shadows by selecting a cursor color of black then setting the alpha register to a none-zero value.

**I** - The I flag indicates to reverse the video output. The color under the cursor is xor’d with -1.

**f** – indicates to flash the cursor. The cursor will flash at a rate determined from the rate4 field.

### Cursor Link Register

The cursor link register indicates cursors which are linked to the next cursor to increase the apparent number of colors available to sixteen rather than four. Each bit in the register specifies the link state for the corresponding sprite. Linked cursors must have their coordinates maintained with the same values.

|  |  |  |
| --- | --- | --- |
| $F68 | ~32 | Link31..0 |

### Cursor Enable Register

The cursor enable register controls which sprites are visible on the screen. Bits in the register enable the sprite display when set to a one for the sprite corresponding to the bit number.

|  |  |  |
| --- | --- | --- |
| $F60 | ~32 | Enable31..0 |

### Collision Register

The collision register indicates which sprites are colliding with other sprites. Each bit in the register corresponds to a sprite. If the bit is set then the sprite has collided with another sprite. The bits will remain set in the register until the register is updated.

|  |  |  |
| --- | --- | --- |
| $F70 | ~32 | SprCollision31..0 |

### Cursor Control Registers

The control register layout for all cursors is identical. The layout is shown only for the first cursor, cursor #0. Note that the count and position registers are 16-bit addressable. Any or all of the 16-bit fields may be updated.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| $800 | Address31..0 | | | | | bitmap address low |
| $804 | (Reserved) Address63..32 | | | | | bitmap address high |
| $808 | zpos8 | | ~16 | MCnt16 | | total number of pixels |
| $80C | ~4 | vpos12 | | ~4 | hpos12 | vertical / horizontal position |
| $810 to $81C | Cursor #1 Registers | | | | |  |
| … | … | | | | |  |
| $9E0 to $9FC | Cursor #31 Registers | | | | |  |
| $A00 to $BFC | reserved | | | | |  |

### Cursor Bitmap Address

This register contains the address of the cursor’s bitmap in the core’s memory. The cursor bitmap occupies contiguous words of memory. The amount of memory required is determined by the MCnt field for the cursor. Each raster line of the cursor is composed of one sixty-four-bit value to allow bitmaps up to 32 pixels in width to be defined. The amount of memory required is one word, which is a fixed amount. Even if the sprite is only five pixels wide, a whole word of memory per scanline is still required.

### MCnt

The size register controls the visible size of the sprite. Sprites may be up to 32 pixels in size horizontally, and up to 512 scan lines vertically. The value placed in the register should be one less than the total count of pixels to display. The count should be a multiple of 32 pixels then minus one. For example, for a 32hx30v sprite the count would be 960-1 = 959.

### Horizontal and Vertical Position

The horizontal and vertical position are relative to the top left corner of the visible screen which is position ( 0, 0 ).

### Z-Order

The z-order (zpos) register controls the appearance priority of the sprite compared to other graphics on-screen. If the sprite’s z-order is less than the z-order of the current pixel it will appear in front of the pixel. Otherwise it will be hidden by the pixel.

## Blitter

### Overview

The display controller has a powerful blitter component which may be used to transfer information in the controller’s memory extremely fast. The blitter consists of four DMA channels (A, B, C, and D). A, B, and C are data source channels and D is a data destination channel. The destination channel may be used in a standalone fashion to draw lines or fill areas. Any or all three of the source channels may be active to fetch data to transfer to the destination. A variety of operations between the data fetched by channels A, B, and C are possible including copy and masking operations.

### Pipeline

The blitter features pipelined memory access with a programmable pipeline depth. When pipelining is active the blitter performs burst memory accesses for a given channel until the pipeline queue is full. It then moves onto the next channel. Once all the queues are full then data operations on the values are performed as the destination memory is being written. This provides the most efficient use of memory bandwidth.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 31 | | | | 0 |  |
| $D00 | Address31..0 | | | | | Channel A address low |
| $D04 | ~32 | | | | | Channel A address high |
| $D08 | Offset31..0 | | | | | Channel A modulo low |
| $D0C | ~32 | | | | | Channel A modulo high |
| $D10 | Count32 | | | | | Channel A Count |
| $D14 | ~32 | | | | |  |
| $D20 | Address31..0 | | | | | Channel B address low |
| $D24 | ~32 | | | | | Channel B address high |
| $D28 | Offset31..0 | | | | | Channel B modulo low |
| $D2C | ~32 | | | | | Channel B modulo high |
| $D30 | Count32 | | | | | Channel B Count |
| $D34 | ~32 | | | | |  |
| $D40 | Address31..0 | | | | | Channel C address low |
| $D44 | ~32 | | | | | Channel C address high |
| $D48 | Offset31..0 | | | | | Channel C modulo low |
| $D4C | ~32 | | | | | Channel C modulo high |
| $D50 | Count32 | | | | | Channel C Count |
| $D54 | ~32 | | | | |  |
| $D60 | Address31..0 | | | | | Channel D address low |
| $D64 | ~32 | | | | | Channel D address high |
| $D68 | Offset31..0 | | | | | Channel D modulo low |
| $D6C | ~32 | | | | | Channel D modulo high |
| $D70 | Count32 | | | | | Channel D Count |
| $D74 | ~32 | | | | |  |
| $D78 | ~16 | | | Data16 | | Channel D Data |
| $D7C | ~32 | | | | |  |
| $D80 | BltSrcWid32 | | | | | Source width |
| $D84 | ~32 | | | | |  |
| $D88 | BltDstWid32 | | | | | Destination width |
| $D8C | ~32 | | | | |  |
| $D90 | ~16 | | | Op16 | | Blitter operation code |
| $D94 | ~32 | | | | |  |
| $D98 | ~2 | PLD6 | ~8 | BltCtrl16 | | Blitter Control |
| $D9C | ~32 | | | | |  |

### Channel A Address

This pair of register sets the address of data source for channel A.

### Channel A Modulo

This pair of registers set the modulo amount for channel A. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

### Channel A Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Channel B Address

This pair of register sets the address of data source for channel B.

### Channel B Modulo

This pair of registers set the modulo amount for channel B. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

### Channel B Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Channel C Address

This pair of register sets the address of data source for channel C.

### Channel C Modulo

This pair of registers set the modulo amount for channel C. The modulo amount is an amount added to the current working address once transfers have hit the source width specification.

### Channel C Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Channel D Address

This pair of register sets the address of data destination for channel D.

### Channel D Modulo

This pair of registers set the modulo amount for channel D. The modulo amount is an amount added to the current working address once transfers have hit the destination width specification.

### Channel D Count

This pair of registers indicates how many pixels are present. If the source count is less than the destination count, then data from the source will begin to repeat at the destination. This can be used for tile copying.

### Source Width

The source width specifies the number of horizontal pixels in the bitmap. It is used along with the modulo register to calculate the address of the bitmap data for a source.

### Destination Width

The destination width specifies the number of horizontal pixels in the bitmap. It is used along with the modulo register to calculate the address of the bitmap data for a source. As an example the screen bitmap is 800 pixels wide, so the width value placed in the register would be 800.

### Blit Control

This register contains bits for control of the blit operation. Channels may be independently enabled or disabled. They may also be set to descending mode where the address decrements through memory instead of incrementing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bit | Default | Purpose |  |  |
| 0 | 0 | Channel A bitmap mode enable |  |  |
| 1 | 0 | Channel A DMA enable |  |  |
| 2 | 0 | Channel B bitmap mode enable |  |  |
| 3 | 0 | Channel B DMA enable |  |  |
| 4 | 0 | Channel C bitmap mode enable |  |  |
| 5 | 0 | Channel C DMA enable |  |  |
| 6 | 0 | reserved |  |  |
| 7 | 0 | reserved |  |  |
| 8 | 0 | Channel A descend mode |  |  |
| 9 | 0 | Channel B descend mode |  |  |
| 10 | 0 | Channel C descend mode |  |  |
| 11 | 0 | Channel D descend mode |  |  |
| 12 | 0 | reserved |  |  |
| 13 | 1 | Blit done indicator |  |  |
| 14 | 0 | Blit active indicator |  |  |
| 15 | 0 | Blit operation trigger |  |  |

### Blit Pipeline Depth ($D98 bits 24 to 29)

This register controls the amount of pipelining present during the blit transfer. In some cases when data to be transferred is located nearby the pipeline depth may need to be reduced. The blitter normally makes use of queues of pixels to improve performance. Typically, the blitter works with 16 pixels in burst mode for any given channel. It will read 16 pixels for a channel before writing out the pixels to the destination. This may cause unexpected results in some circumstances.

The pipeline depth may also be reduced to give the main processor a greater share of memory access during the blit. It takes approximately n + 4 clock cycles for each burst read where n is the pipeline depth. If all four channels are active and the pipeline depth is 16, then this is approximately 80 clock cycles to perform one transfer. Note that while the blitter is performing a group transfer the main cpu will stall for the duration if it attempts to write the controller’s memory.

## Sync Generator Controls

This set of registers controls the video sync generation. There is an optional lock to the register set to prevent inadvertent alteration of the registers. As defaulted the sync generation is for 800x600 VGA mode using a 40MHz video clock.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| $F80 | ~4 | 1vTotal11..0 | ~4 | | 1hTotal11..0 | | vert. horiz. total register |
| $F84 | ~32 | | | | | |  |
| $F88 | ~4 | 1hSyncOff11..0 | | ~4 | 1hSyncOn11..0 | | sync on/off |
| $F8C | ~4 | 1vSyncOff11..0 | | ~4 | 1vSyncOn11..0 | |
| $F90 | ~4 | 1hBlankOff11..0 | | ~4 | 1hBlankOn11..0 | | blank on / off |
| $F94 | ~4 | 1vBlankOff11..0 | | ~4 | 1vBlankOn11..0 | |
| $F98 | ~4 | hBorderOff11..0 | | ~4 | hBorderOn11..0 | | border on / off |
| $F9C | ~4 | vBorderOff11..0 | | ~4 | vBorderOn11..0 | |
| $FA0 | ~4 | vstart11..0 | | ~4 | hstart11..0 | |  |
| $FA4 | ~4 |  | | ~4 |  | |  |
| $FA8 | ~4 | vpos12 | | ~4 | hpos12 | | vert., horiz. pos (read only) |
| $FAC | ~4 | ~12 | | ~11 | | fpos5 | frame pos. (read only) |
|  |  | | | | | |  |
| $FE8 | ~32 | | | | | | sync generator lock |

1. These registers may be locked via register $FE8.

Writing register $FE8 with the value $A1234567 will lock the sync generator controls so that they may not be modified. Write register $FE8 with zero to unlock the controls.

# Audio

The AVIC Core contains five audio channels, four output and one input. All channels are organized with registers in a similar fashion.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Regno | Channel registers | | | | | | | | | | |  |
| $C00 | Address31..0 | | | | | | | | | | | Sample buffer address |
| $C04 | ~32 | | | | | | | | | | | Sample buffer address high |
| $C08 | ~16 | | | | | | Length15..0 | | | | | Length of buffer |
| $C0C | ~12 | | | | Period19..0 | | | | | | | Time constant |
| $C10 | Data15..0 | | | | | | Volume15..0 | | | | |  |
| $C14 | ~32 | | | | | | | | | | |  |
| $C18 | ~32 | | | | | | | | | | |  |
| $C1C | ~32 | | | | | | | | | | |  |
| $C20 to $63C | Channel 1 registers | | | | | | | | | | |  |
| $C40 to $C5C | Channel 2 registers | | | | | | | | | | |  |
| $C60 to $C7C | Channel 3 registers | | | | | | | | | | |  |
| $C80 to $C9C | Input channel registers | | | | | | | | | | |  |
| $CA0 | ~ | t | ~ | res5 | | p | | mix2 | Enable5 | | | AUD\_CTRL1 |
| $CA2 |  | | | | | fm3 | | | |  | am3 | AUD\_CTRL2 |

## Buffer Address

This register determines where in memory the buffer of sample values is located. There is a separate buffer for each audio channel.

## Length

This register determines the size of the audio buffer. Once playback has reached the end of the buffer it will circle around back to the beginning again. The maximum buffer size is 65536 entries (128kB).

## Period

This register controls the periodicity at which DMA transfers occur to memory to playback a sound from the sample buffer. The period is in units of clock cycles. For 22.05 kHz playback with a 40MHz master clock a value of 40e6/22.05e3 = 1814. is required. The maximum playback period is 65536 units or 610 Hz. DMA transfers are double buffered to eliminate jitter.

## Audio Control Register #1

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $CA0 | ~ | t | ~ | res5 | p | mix2 | Enable5 | AUD\_CTRL1 |

**Enable5** enables audio output channels 0 to 3 and the input channel. Bits 0 to 3 correspond to channels 0 to 3. Bit 4 corresponds to the audio input channel. A one enables the channel, a zero disables it.

**Mix2** Bit 5 of the control register, if set to one mixes audio channel #1 into audio channel #0. In a similar fashion audio channel #3 is mixed into channel #2 if bit 6 is set.

**res5** There is a bit for each channel to reset it in the res5 field. Bits 8 to 11 reset channels 0 to 3. Bit 12 resets the input channel.

**t** Puts the audio into test output mode. An approximately 600 Hz signal will be output by all channels. The main use of the mode is to test the connection through the physical layer of the audio system.

## Audio Control Register #2

Audio control register number two has bits that enable am or fm modulation of a channel by the previous channel. Channel 0 may module channel 1. Channel 1 may modulate channel 2, and channel 2 may modulate channel 3.

## Copper

### Overview

The copper is co-processor capable of updating the display controller’s register set at specific points during the display generation. The copper has a small instruction set of only four instructions. Copper instructions are 128-bit words. If enabled, during every vertical reset of the display the copper’s program counter is loaded with an address stored in the copper’s program address register and the copper begin executing instructions.

### Copper Address Registers

Copper address registers are used to store addresses for copper programs that include the restart address (address register zero), and subroutine linkage addresses.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 63 0 | | | |  |
| $CC0 | ~32 | Offset31..0 | | | Address register #0 |
| …. |  | | | | 14 more registers |
| $CF8 | ~32 | Offset31..0 | | | Address register #15 |
| $CB0 |  | | F | E | Control Register |

### Copper Control Register

This register has bits to enable the copper. It also has a bit indicating the restart rate for the copper. The copper program may be automatically restarted at the beginning of every video frame (the default configuration) or it may restart every 16th frame.

### Copper Instruction Set Description

### WAIT

**Description:**

The wait instruction waits for the display generation scan to reach a specific horizontal, vertical and frame position. The copper is not active while waiting and other devices may freely access the display register set.

**Instruction Format:**

|  |  |  |
| --- | --- | --- |
| Bits | Field | Description |
| 127:126 | Op2 |  |
| 112 | B | blitter wait |
| 101:96 | F | frame. to wait for |
| 91:80 | V | vert. pos. to wait for |
| 75:64 | H | horiz. pos. to wait for |
| 37:32 | MF | mask: frame |
| 27:16 | MV | mask: vertical pos |
| 11:0 | MH | mask: horizontal pos |

B – indicates that the copper should wait for an outstanding blit operation to complete before continuing.

F – the frame number that the copper should wait for. This may be masked off by the frame mask (MF) field.

V – the vertical position that the copper should wait for. This may be masked by the vertical position mask field (MV)

H – the horizontal position that the copper should wait for. this may be masked by the horizontal position mask field (MH).

### MOVE

**Description:**

The MOVE instruction moves a constant value into one of the display controller registers. This allows the copper to do things like initiate a blitter operation or trigger an interrupt.

**Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 127126 | 125 76 | 75 64 | 63 0 |
| 12 | ~50 | Regno12 | Data63..0 |

### SKIP

**Description:**

The SKIP instruction skips over the next instruction if the display generation scan has already passed the point specified in the instruction. The SKIP instruction has the same format as the WAIT instruction.

**Instruction Format:**

|  |  |  |
| --- | --- | --- |
| Bits | Field | Description |
| 127:126 | Op2 |  |
| 112 | B | blitter wait |
| 101:96 | F | frame. to wait for |
| 91:80 | V | vert. pos. to wait for |
| 75:64 | H | horiz. pos. to wait for |
| 37:32 | MF | mask: frame |
| 27:16 | MV | mask: vertical pos |
| 11:0 | MH | mask: horizontal pos |

B – indicates that an outstanding blit operation should have completed.

F – the frame number. This may be masked off by the frame mask (MF) field.

V – the vertical position. This may be masked by the vertical position mask field (MV)

H – the horizontal position. This may be masked by the horizontal position mask field (MH).

### JUMP

**Description:**

The jump instruction allows the copper to transfer program flow to another point. The instruction may store a linkage address in one of the copper’s address registers, or it may read one of the copper’s address registers to set the target address. This allows the copper to perform a simple subroutine of a single nesting level. The jump instruction may execute conditionally based on whether or not a blitter operation has completed.

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 127126 | 125 84 | 83 80 | 79 75 | 74 72 | 7168 | 67 64 | 63 32 | 31 0 |
| 32 | ~ | L | ~ | Cond3 | ~ | R | ~32 | Addr31..0 |

L – linkage register number to store current address in

Cond – condition and operation to perform

R – linkage register number to load pc from for return operation

Addr – target address to jump to

|  |  |  |
| --- | --- | --- |
| Cond3 | Jump operation |  |
| 0 | jump unconditionally to target addr |  |
| 1 | jump if blitter is done, otherwise wait |  |
| 2 | jump if blitter is not done, otherwise wait |  |
| 4 | return unconditionally to address contained in linkage register R |  |
| 5 | return if the blitter is done, otherwise wait |  |
| 6 | return if the blitter is not done, otherwise wait |  |
| 3,7 | reserved |  |

# Miscellaneous

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $FE0 | ~8 | Strip Count8 | ~14 | Res2 |
| $FE4 | ~32 | | | |

## Strip Count8

The number of 128-bit memory strips necessary to read all the pixel information for a scan-line is programmable. The default is 100 which is the amount required to read 800, 16 bit pixels from memory. In 400x300 mode this should be changed to 50.

## Res2

This register field controls the display resolution. The value in this field acts as a divider of the resolution. The default value is zero for 800x600.

|  |  |  |
| --- | --- | --- |
| Res2 | Typical |  |
| 0 | 800x600 | full resolution |
| 1 | 400x300 | half resolution |
| 2 | 200x150 | quarter resolution |
| 3 | 100x75 | eigth resolution |

# Port Signals

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Width | I/O |  |
| rst\_i | 1 | i | This active high signal resets the core and WISHBONE bus interfaces |
| Slave Signals | | | |
| clk\_i | 1 | i | Clock signal for slave peripheral interface (typically the cpu clock) |
| cs\_i | 1 | i | circuit select |
| cyc\_i | 1 | i | cycle is valid |
| stb\_i | 1 | i | data transfer in progress |
| ack\_o | 1 | o | data transfer acknowledge |
| sel\_i | 8 | i | byte lane selects |
| we\_i | 1 | i | write enable to register set |
| adr\_i | 12 | i | addresses the registers of the core |
| dat\_i | 64 | i | data input for registers |
| dat\_o | 64 | o | data output of registers |
| Master Signals | | | |
| m\_clk\_i | 1 | i | clock signal for bus master interface (typically the memory clock) |
| m\_cyc\_o | 1 | o | cycle is valid |
| m\_stb\_o | 1 | o | data transfer is taking place |
| m\_ack\_i | 1 | i | data transfer acknowledge |
| m\_sel\_o | 16 | o | byte lane selects |
| m\_we\_o | 1 | o | write enable to memory |
| m\_adr\_o | 32 | o | Memory address for bitmap data read |
| m\_dat\_i | 128 | i | data input from bitmap memory |
| m\_dat\_o | 128 | o | this is data output to the memory |
| Video Port | | | |
| vclk | 1 | i | This is the video clock input (40 MHz) |
| hSync | 1 | o | horizontal sync signal |
| vSync | 1 | o | vertical sync signal |
| de | 1 | o | video blanking indicator (display enable) |
| rgb | 24 | o | color output video data in RGB (8,8,8) format |
| Audio Port | | | |
| aud0\_o | 16 | o | digital audio output port #0 |
| aud1\_o | 16 | o | digital audio output port #1 |
| aud2\_o | 16 | o | digital audio output port #2 |
| aud3\_o | 16 | o | digital audio output port #3 |
| audi\_i | 16 | I | digital audio input |
|  |  |  |  |