# AVCore.v – Audio / Video Display Controller Core

## Overview

This is a full featured audio / video display controller that integrates a bitmap display controller with several coprocessing elements in order to enhance system performance. It contains a dedicated character blitter 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 with the system via a 16-bit bus.

## Features

800 x 600 x 16 bpp display resolution

400 x 300 x 16 bpp lowres mode

programmable sync generator

internal dual ported display memory

graphics command queue

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

point plot / line draw / rectangle / acceleration

general purpose blitter – three sources, one destination

‘copper’ coprocessor

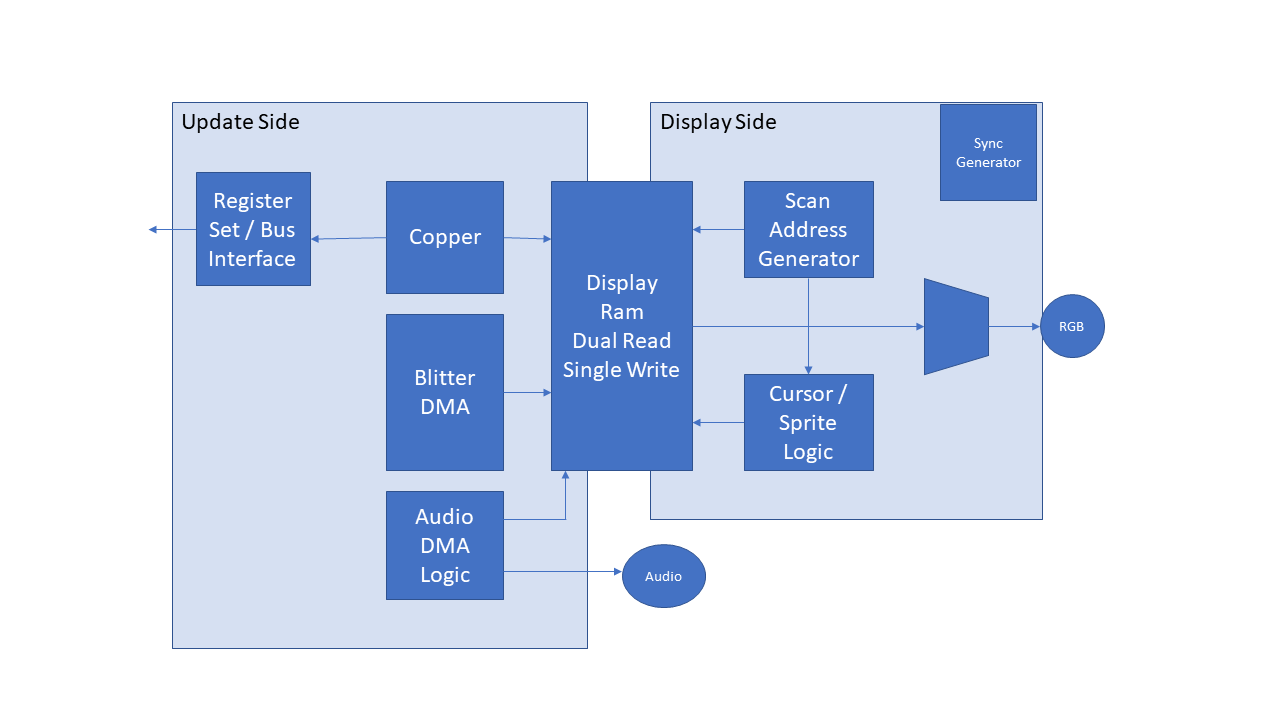
32 hardware cursors / sprites

4 channel audio output with independent timing

1 channel audio input

## General Organization

The following block diagram shows the organization of the AV core. The core is centred around multi-port block ram which has an update side and a display side. Display and update are decoupled from each other so that display scanning doesn’t impact the performance of updates.



## Clocks

The display scanning side and the update side of the core may use independent clocks. The video clock rate for the display side is 40MHz for an 800x600 VGA display.

## Display Format

The display controller supports a default format of 800 x 600. The primary reason for this selection is the amount of memory dedicated to display purposes. A 800 x 600 display requires approximately 1M words of memory. 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 internal memory is 512k words by 16 bits wide 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 memory is dual ported with an update side and a display side. Since the display and update are decoupled from each other the system performance is considerably improved. Twice the bandwidth is available compared to a display memory that uses a single port for access.

## Z-Buffer

There is an additional internal memory used to store the graphics layer that a pixel is located on. This memory is used to prioritize the display relative to sprites. If the sprite’s z-layer is numerically less than or equal to the z-buffer value the sprite will appear in front of the display, otherwise the graphic displays in front of the sprite. The z-buffer is accessible in the same address range as the display memory. Access to the z-buffer may be enabled by setting the z-buffer access enable bit. The z-buffer stores two-bit values for each displayed pixel. Eight values are packed into a memory word.

## Font Table

The core supports the use of multiple fonts onscreen at the same time through the use of 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 located in the core’s internal memory and indexed by the font id register to select a particular 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 | fixed1 | width5 | height5 | Addr20..16 | | width and height |
| 2 | Address15..0 | | | | | Address of character bitmaps |
| 4 | ~12 | | | | Address19..16 |  |
| 6 | Address15..0 | | | | | Glyph width table address |

Each font table entry is eight bytes in size. The font id (register $594) 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 $590.

## 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 the core’s internal memory.

## Font Table Address

This pair of registers 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.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 15 4 | 3 0 |  |
| $590 |  | Offset19..16 | font table address high |
| $592 | Offset15..0 | | font table address low |

# Description / Registers

## Memory Pointers

|  |  |  |  |
| --- | --- | --- | --- |
|  | 15 4 | 3 0 |  |
| $400 |  | Offset19..16 | Screen bitmap base high |
| $402 | Offset15..0 | | Screen bitmap base low |

### Screen Bitmap Base

This pair of registers determines where in the controller’s memory the 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.

## Z Buffer Control

|  |  |  |  |
| --- | --- | --- | --- |
| $43E |  | Z | z buffer access flag |

When the z buffer access flag is set updates to memory update the z buffer instead of the regular display memory.

## 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 are written with desired parameters. Then the command to perform is written to the command register. Writing the command 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 registers that need to change. For instance, a string of text spaces may be queued by simply modifying the X position while leaving the remaining registers alone.

The queue has a fixed number of entries 31 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 index register.

### Command Queue Registers

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 15 9 | | | 8 0 | | | |  | |
| $420 | |  | | | code9 | | | | Char code | |
| $422 | |  | RGB5515 | | | | | | Foreground color | |
|  | | 15 | |  | 14 0 | | | | |  |
| $424 | | A | RGB5559 | | | | | | | Background/Draw Color |
|  | 15 12 | | | | 11 0 | | | | |  |
| $426 |  | | | | pos12 | | | | | x0 position |
| $428 |  | | | | pos12 | | | | | y0 position |
| $42A |  | | | | Sz4 |  | | Sz4 | | font size |
| $42C |  | | | | | | Index5 | | | queue index |
| $42E |  | | | | | Command8 | | | | command |
| $430 |  | | | | pos12 | | | | | x1 position |
| $432 |  | | | | pos12 | | | | | y1 position |

### Char Code

This register specifies the character code to be used in a bitmap lookup for transfer to screen memory. The core supports a nine bit character code allowing for 512 different characters in the font.

### Foreground Color / Raster Operation

This register specifies the foreground color to use for character drawing or the raster operation to use for line drawing.

|  |  |  |
| --- | --- | --- |
| Number | Result |  |
| 0 | set output to black |  |
| 1 | set output to background color (copy) |  |
| 2 | reserved |  |
| 3 | reserved |  |
| 4 | ‘and’ the current pixel color with the background color |  |
| 5 | ‘or’ the current pixel color with the background color |  |
| 6 | ‘xor’ the current pixel color with the background color |  |
| 7 | ‘and not’ the current pixel color with the background color |  |
| 8 to 14 | reserved |  |
| 15 | set output to white |  |

### Background Color / Draw Color

This register specifies the background color for character drawing or the color to be used for other drawing operations (plot, line draw).

A = alpha blend bit

When alpha blending is selected the lower nine bits of the color value indicate how much to shift the RGB components of target pixel to the right. Right shifting the color components blends the color towards black, creating a shadow effect. Bit 0-2 are a shift count for the green component, bits 3,4,5 for the blue and bits 6,7,8 for red. Note that a shift count of zero leaves the color at it’s original value. This allows a transparent effect.

### X0, Y0 Position

This pair of registers sets the graphics position for commands requiring at least one co-ordinate (character plot, point plot, line draw).

### X1, Y1 Position

This pair of registers sets the second graphics coordinate for commands requiring two points (line draw).

### Queue Index

The queue index register may be read to determine how many commands are queued. It may also be written to flush the queue of commands. New commands should not be placed in the queue if it is full.

### Command Register

The command register specifies which graphics operation to perform. Writing to the command register queues the graphics command.

|  |  |  |
| --- | --- | --- |
| Command8 | Operation Performed |  |
| 0 | Draw character bitmap |  |
| 1 | Plot point |  |
| 2 | Draw line |  |
| 3 | Fill Rectangle |  |
| 4 | Set Texture |  |
| 5 | Tile Rectangle |  |

## 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 64 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. \* Colors are shared between the first set of sixteen sprites and the next sixteen sprites.

### Cursor Color Palette

The cursor color palette has 64 entries each of which is a 32-bit vector including additional attributes besides just the color. Attributes include alpha blending, reverse video and flashing. Cursors are organized into two groups of sixteen (0 to 15 and 16 to 31). Both groups of cursors share the same color palette entries. That is cursor 16 uses the same set of colors as cursor 0, cursor 17 uses the same set of colors as cursor 1, and so on.

The registers are organized into groups of four, 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 four 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 to 3. The second group is shared between sprites 4 to 7, and so on. Note that color palette entry #0 is never used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| $000 | Alpha8 | | ~2 | i | f | rate4 | color0 |  |
| $002 | ~ | RGB55515 | | | | |  |
| $004 to $0FE | 63 more register pairs | | | | | |  |  |

**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 in order 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $100 | Link31..16 |  |  |  |
| $102 | Link15..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. Note that sprite data fetches continue to occur however this has no effect on the performance of other operations as sprite fetches occur on the display side of the ram in the controller.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $108 | Enable31..16 |  |  |  |
| $10A | Enable15..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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| $10C | SprCollision31..16 |  |  |  |
| $10E | SprCollision15..0 |  |  |  |

### Cursor Control Registers

The control register layout for all cursors is identical. The layout is shown only for the first cursor, cursor #0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| $200 | ~12 | | | | Address19..16 | bitmap address high |
| $202 | Address15..0 | | | | | bitmap address low |
| $204 | ~4 | pos12 | | | | horizontal position |
| $206 | ~4 | pos12 | | | | vertical position |
| $208 | vsize10 | | ~ | hsize5 | | size |
| $20A |  | | | | zorder4 | z order |
| $210 to $21A | Cursor #1 Registers | | | | |  |
| … | … | | | | |  |
| $3F0 to $3FE | Cursor #31 Registers | | | | |  |

### Cursor Bitmap Address

This pair of registers 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 horizontal and vertical size of the cursor. Each raster line of the cursor is composed of four sixteen-bit values to allow bitmaps up to 32 bits in size to be defined. The amount of memory required is four words, which is a fixed amount. Even if the sprite is only five pixels wide, four words of memory per scanline are still required.

### 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 ).

### Size

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 desired width or height.

### Z-Order

The z-order 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 in order to draw lines or fill areas. Any or all three of the source channels may be active in order to fetch data to transfer to the destination. A number of operations between the data fetched by channels A, B, and C are possible including copy and masking operations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 15 4 | | 3 0 |  |
| $480 |  | | Offset19..16 | Channel A address high |
| $482 | Offset15..0 | | | Channel A address low |
| $484 |  | | Offset19..16 | Channel A modulo high |
| $486 | Offset15..0 | | | Channel A modulo low |
| $488 |  | | Offset19..16 | Channel B address high |
| $48A | Offset15..0 | | | Channel B address low |
| $48C |  | | Offset19..16 | Channel B modulo high |
| $48E | Offset15..0 | | | Channel B modulo low |
| $490 |  | | Offset19..16 | Channel C address high |
| $492 | Offset15..0 | | | Channel C address low |
| $494 |  | | Offset19..16 | Channel C modulo high |
| $496 | Offset15..0 | | | Channel C modulo low |
| $498 |  | | Offset19..16 | Channel D address high |
| $49A | Offset15..0 | | | Channel D address low |
| $49C |  | | Offset19..16 | Channel D modulo high |
| $49E | Offset15..0 | | | Channel D modulo low |
| $4A0 |  | | Width19..16 | Source width high |
| $4A2 | Width15..0 | | | Source width low |
| $4A4 |  | | Width19..16 | Destination width high |
| $4A6 | Width15..0 | | | Destination width low |
| $4A8 | Data16 | | | Destination data reg |
| $4AA | Depth5 | | | Blit Pipeline depth |
| $4AC | Control16 | | | Blit Control Reg |
| $4AE | Operation16 | | | Blit operation reg. |
| $4AA |  | Depth5 | | Blit Pipeline depth |
| $4B0 |  | Count19..16 | | Channel A count high |
| $4B2 | Count15..0 | | | Channel A count low |
| $4B4 |  | Count19..16 | | Channel B count high |
| $4B6 | Count15..0 | | | Channel B count low |
| $4B8 |  | Count19..16 | | Channel C count high |
| $4BA | Count15..0 | | | Channel C count low |
| $4BC |  | Count19..16 | | Channel D count high |
| $4BE | Count15..0 | | | Channel D count low |

### 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 640 pixels wide, so the width value placed in the register would be 640.

### 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 emable |  |  |
| 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

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 in order 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 in order 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.

## 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 six ten-bit words in size for a total of sixty bits. 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 15 4 | 3 0 | | |  |
| $4C0 |  | Offset19..16 | | | Address register #0 high |
| $4C2 | Offset15..0 | | | | Address register #0 low |
| …. |  | | | | 14 more registers |
| $4DC |  | Offset19..16 | | | Address register #15 high |
| $4DE | Offset15..0 | | | | Address register #0 low |
| $4E0 |  | | 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 and vertical position. The copper is not active while waiting and other devices may freely access the display register set.

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6262 | 6159 | 58 | 57 53 | 52 41 | 40 29 | 28 24 | 23 12 | 11 0 |
| 02 | ~ | B | F | V | H | MF | MV | MH |

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:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6362 | 61 43 | 42 32 | 31 16 | 15 8 | 7 0 |
| 12 | ~19 | Regno11 | ~16 | Data15..8 | Data7..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:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6362 | 6159 | 58 | 57 53 | 52 41 | 40 29 | 28 24 | 23 12 | 11 0 |
| 22 | ~ | B | F | V | H | MF | MV | MH |

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:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6362 | 6156 | 55 52 | 51 49 | 48 | 47 44 | 43 20 | 19 0 |
| 32 | ~ | L | Cond3 | ~ | R | ~24 | Addr |

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 |  |

## Audio

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

|  |  |  |  |
| --- | --- | --- | --- |
| Regno | Channel registers | |  |
| $600 | ~12 | Address19..16 | Sample buffer address |
| $602 | Address15..0 | |  |
| $604 | Length15..0 | | Length of buffer |
| $606 | Period15..0 | | Time constant |
| $608 | Volume15..0 | |  |
| $60A | Data15..0 | |  |
| $610 to $61A | Channel 1 registers | |  |
| $620 to $62A | Channel 2 registers | |  |
| $630 to $63A | Channel 3 registers | |  |
| $640 to $64A | Input channel registers | |  |

### Buffer Address

This pair of registers determine where in the core’s internal 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 in order 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.