

Altirra Hardware Reference Manual

by

Avery Lee

Version: 10/18/09

Table of Contents

| | |
|--|----|
| 1.1. Introduction..... | 5 |
| 2. CPU..... | 6 |
| 2.1. Registers..... | 7 |
| 2.2. Cycle timing..... | 7 |
| 2.3. Interrupts..... | 9 |
| 2.4. Examples..... | 9 |
| 2.5. Further reading..... | 9 |
| 3. System control..... | 10 |
| 3.1. System Reset button..... | 11 |
| 3.2. Peripheral Interface Adapter (PIA)..... | 11 |
| 3.3. Bank switching..... | 12 |
| 3.4. Extended Memory..... | 12 |
| 3.5. Miscellaneous Connections..... | 14 |
| 3.6. Examples..... | 14 |
| 3.7. Further reading..... | 14 |
| 4. ANTIC..... | 15 |
| 4.1. Addressing..... | 16 |
| 4.2. Display modes..... | 16 |
| 4.3. Display list..... | 18 |
| 4.4. Scrolling..... | 21 |
| 4.5. Display list interrupts..... | 22 |
| 4.6. WSYNC..... | 24 |
| 4.7. VCOUNT..... | 25 |
| 4.8. Player/missile DMA..... | 25 |
| 4.9. DMA timing..... | 25 |
| 4.10. Cycle counting example..... | 35 |
| 4.11. Examples..... | 36 |
| 4.12. Further reading..... | 37 |
| 5. POKEY..... | 38 |
| 5.1. Addressing..... | 39 |
| 5.2. Initialization..... | 39 |
| 5.3. Sound generation..... | 39 |
| 5.4. Serial port..... | 41 |
| 5.5. Clock generation..... | 42 |
| 5.6. Pseudo-random number generators..... | 42 |
| 5.7. Interrupts..... | 43 |
| 5.8. Keyboard scan..... | 43 |
| 5.9. Examples..... | 43 |
| 5.10. Further reading..... | 44 |
| 6. CTIA/GTIA..... | 45 |
| 6.1. Player/missile graphics..... | 46 |
| 6.2. Collision detection..... | 47 |
| 6.3. Priority control..... | 48 |
| 6.4. High resolution mode (ANTIC modes 2, 3, and F)..... | 50 |
| 6.5. GTIA special modes..... | 50 |
| 6.6. Cycle timing..... | 52 |
| 6.7. Further reading..... | 53 |
| 7. Reference..... | 54 |
| 7.1. Memory map..... | 55 |
| 7.2. Register list..... | 56 |

7.3. GTIA registers..... 57

7.4. POKEY registers..... 77

7.6. ANTIC registers..... 93

8. Bibliography..... 105

Copyright © 2009 Avery Lee, All Rights Reserved.

Permission is granted to redistribute this document in verbatim form as long as it is done free of charge and for non-commercial purposes.

All trademarks are the property of their respective owners.

While the information in this document is presumed correct, no guarantee is provided as to its accuracy or fitness for a particular use.

1.1 Introduction

This document describes the hardware programming model used by Altirra, an emulator for the Atari 8-bit series of home computers, including the 400, 800, 600XL, 800XL, 1200XL, and 130XE models. Although the emulator provides a virtual programming environment, it is intended to mimic the actual hardware. This document attempts to describe the hardware in detail as the target to which the emulator aspires to imitate. Some of this information has been collected from both official and unofficial sources, and some of it has been determined by hand through testing on a real, still functioning Atari 800XL.

While I've spent a lot of time tracking down details myself, I have to acknowledge the substantial amount of literature already available which provided background for this document. First and foremost, I'm indebted to the technical staff behind the *Atari Home Computer System Hardware Manual*, which did a very good job of describing the behavior and programming specifications for the official functionality in the Atari hardware, and which should be considered required reading prior to this document. Similar shout-outs go to the authors of Atari's *OS Manual*, which similarly documents the software side, and to Ian Chadwick and his *Mapping the Atari, Revised Edition*, which contains the most detailed and complete memory map of the Atari I know of.

If you have the time and inclination, please check out my Altirra emulator, available at the following web address:

<http://www.virtualdub.org/altirra.html>

-- Avery Lee

2 CPU

The 6502 chip is the CPU of the Atari. Used in many computers of the time and still in use as a microcontroller in enhanced forms, both the official and unofficial behaviors of the 6502 are well known. While the 6502 was later superceded by chips such as the 65C02 and the 65C816, the Atari 8-bit line continued using the original 6502 until the very end.

Note that there is some confusion as to the precise chip used in the Atari 8-bit series. The original 400/800 use the NMOS 6502, along with a handful of extra circuitry to provide the ability to halt the CPU for ANTIC DMA; this was later replaced with the 6502C, a custom version that contains the HALT logic built-in. This should not be confused with the CMOS 65C02, which is an enhanced 6502 with additional instructions and which was never used in the Atari 8-bit line.

The 6502 contains many nuances and unusual undocumented behaviors which are crucial to understand when programming to the metal on the Atari 8-bit series. For the sake of brevity, the basic architecture of the 6502 will be omitted here to allow more space for documenting these corner cases.

2.1 Registers

Unused flag

The 6502 does not use bit 5 of the P register. It can't be cleared and always reads as a 1.

On the 65C816, bit 5 is reused as the (M)ode bit in native mode.

Break (B) flag

Bit 4 of the processor status register is the (B)reak bit and is used to indicate whether an IRQ or a BRK instruction caused the IRQ routine to be run. It is set if the trigger was an IRQ and cleared if it was a BRK.

Contrary to both official and unofficial documentation, the B bit does not actually exist in the P register. Attempting to clear bit 4 of P and reading the result back always gives a 1 bit. The only time the B flag is visible is when the 6502 pushes the P register on the stack as part of interrupt handling. In that case, the P value pushed onto the stack will have bit 4 cleared for a BRK.

Decimal (D) flag

The D bit (bit 3) in the processor status register activates decimal mode in the 6502. When set to 1, the ADC and SBC instructions perform BCD correction. CMP is not affected.

NMOS 6502s do not clear the D flag automatically, so it must be cleared on reset. It should also be cleared in an interrupt handler if the interrupt code uses ADC or SBC and mainline code may use decimal mode.

There do not appear to be any documented rules for use of decimal mode in Atari programs – the floating-point library sets D without blocking interrupts, and neither the NMI nor IRQ routines in the OS-B firmware ensure that D is cleared. Most of the time this is harmless since ADC and SBC are the only instructions affected and those are not generally used in interrupt handlers. The SIO handlers, however, use ADC for checksums. If you use SIO, you must ensure that any IRQ or NMI handlers you install do not set the D flag while interrupts are enabled. This includes the system timer 2 handler, which is called from stage 2 (deferred) handling with interrupts re-enabled.

Unlike OS-B, the XL/XE firmware does clear D before executing VBI code. It still does not do so for IRQs or for DLIs, however.

Decimal handling was changed significantly in the 65C02 and 65C816. On the 65C02, decimal mode is cleared automatically when entering interrupt handlers, and the ADC/SBC instructions take an extra cycle to compute proper flags. The 65C816 also clears decimal mode and computes proper flags, but takes the same number of cycles as the NMOS 6502. These differences are used by programs to detect the 65C02 and 65C816.

2.2 Cycle timing

Clock speed

On an NTSC machine, the 6502 runs at exactly half the speed of the color clock, or 1.789773MHz. There are exactly 114 cycles per scan line and 29,868 cycles per frame. On a PAL machine, the 6502 runs at 2/5ths the color subcarrier frequency, or 1.77345MHz; there are still 114 cycles per scan line, but 35,568 cycles per frame.

DMA contention

On occasion the Atari's custom chips must fetch data from memory. This is known as Direct Memory Access (DMA), and when it occurs, the 6502 is blocked from the memory bus while ANTIC does a read cycle. This phenomenon slows down execution of code on the CPU and is known as DMA contention. All DMA in the Atari is related to the display and therefore the graphics setup determines the reduction in CPU performance. For NTSC, the highest rate at which the CPU can run is 92% (1.65Mcycles/sec); the standard Graphics 0 display reduces this to 64% (1.14Mcycles/sec). PAL runs noticeably faster since all display related DMA runs only 5/6ths as often.

Dead memory cycles

The 6502 uses the memory bus on every cycle without exception. Most of the time this is for useful work and therefore leads to very efficient bus utilization. There are cases, however, when these cycles are wasted cycles, such as:

- The second cycle of an implied mode instruction. (TXA)
- The ALU cycle of a read-modify-write instruction. (INC abs)
- The second-to-last cycle of an absolute or indirect indexed write. (STA abs, X)
- The second-to-last cycle of an absolute or indirect indexed read that crosses a page boundary (AND abs, Y).
- Conditional branches that cross a page boundary (BNE).

A memory transaction is issued during these dummy cycles and therefore these dead cycles cannot be overlapped by DMA – the CPU must still be halted. For the most part these cycles are harmless, as the Atari is a fairly safe platform where reads to hardware registers seldom have side effects. This may not be true if external hardware on the cartridge or PBI bus is involved, however. It is also definitely not true in the special case of a read-modify-write instruction, however, where the extra cycle is a write cycle instead of a read cycle.

Crossing page boundaries

The 6502 attempts to optimize indexed reads by issuing a speculative read before it has adjusted for a possible carry in the high byte. If no carry is required, a cycle is saved. Otherwise, if a carry is required, it will retry the read with the correct address. For example, given the following sequence:

```
LDX    #$80
LDA     $20F0,X
```

...the 6502 will read \$2070 first, and then retry with the correct address \$2170. For the most part this is harmless on the Atari, since only writes to registers have side effects. The only modes that have this behavior are: abs,X, abs,Y, and (zp),Y. The zp,X, zp,Y, and (zp,X) modes do not need to index outside of zero page and wrap from \$00FF to \$0000 without an extra cycle. The (abs) mode, unique to JMP, also lacks the extra clock due to the well-known bug on the NMOS 6502 of accessing \$xxFF and \$xx00.

Because writes cannot be done speculatively, stores using the abs,X, abs,Y, and (zp),Y modes always take the extra clock cycle. The first clock cycle is a speculative read and the second clock cycle is a write with the correct address. Read-modify-write instructions also always take an extra clock cycle, indexed or not, except that the dummy cycle is a **write** cycle.

Branches that cross a page boundary also have this behavior, doing a read with an incorrect address high byte first, and taking four clock cycles instead of three. No additional cycle is taken to cross a page boundary for a non-taken branch, a JMP or JSR instruction, or any other non-branch execution.

2.3 Interrupts

Interrupt timing

Interrupts are only responded to at the end of an instruction. This means that longer instructions can increase interrupt response delay. The longest instruction possible on the 6502 is seven clocks, which can be due to a (zp),Y access crossing a page boundary, a read-modify-write instruction using abs.X mode, or a BRK/interrupt. However, much longer delays can occur if a store to WSYNC [D40A] is performed, which can lengthen an instruction by as much as a hundred clock cycles. Use of WSYNC should be avoided if display list interrupts or other time-critical interrupts are active.

Clearing I with an interrupt pending

If an interrupt is already pending but is blocked by the I flag, clearing the I flag will result in the interrupt occurring at the end of the next instruction that clears it, and not after the clearing instruction. For instance, given the following code:

```
CLI  
NOP
```

The pending interrupt will not be serviced until the end of the NOP instruction.

2.4 Examples

Pole Position

The decrementing counters seen at the end of a race rely on the undocumented behavior of the N flag in decimal mode. If the N flag is not emulated correctly, the counters may underflow and count indefinitely.

2.5 Further reading

For a witty introduction to 6502 assembly language programming, read [LAN84].

Everyone knows about the official 6502 instruction set and about the JMP indirect bug, but sources giving exact corner-case behavior in other areas are scarcer. For cycle-level operation of the 6502, [MOS76] and [MOS76a] give details that can be difficult to find elsewhere, such as precise timing for acknowledging non-maskable interrupts. The datasheet in [EYE86] gives similar information for the 65C816 and has valuable information about differences between the NMOS 6502, 65C02, and 65C816.

For undocumented instruction details, consult [VIC09] for a thorough overview and for functionality and timing details. Note, however, that there are some errors in compared to the actual 6502 and the VICE emulator in the BCD correction algorithm.

3 System control

3.1 System Reset button

On the original 400/800, the [SYSTEM RESET] key is connected to the RNMI line on ANTIC, which then causes an NMI to be issued to the 6502. The system NMI routine detects this condition via bit 5 of NMIST and invokes warm start behavior.

Starting with the 1200XL, this behavior was changed to use real reset logic instead. On the XL/XE models, pressing the Reset button causes the reset lines to be pulled on the 6502, ANTIC, FREDDIE, and PIA. This causes NMIs to be masked, memory banking to be reset to default, and the 6502 to restart execution at the reset vector. The RNMI line is permanently wired with a pullup to +5V and thus ANTIC will never signal a system reset NMI on these models.

3.2 Peripheral Interface Adapter (PIA)

The 6502 PIA chip controls several miscellaneous functions within the Atari.

I/O ports

The PIA contains two data ports, port A and port B. Each contains eight bits which are individually switchable between input mode or output mode by a data direction register. Port A is controlled by control register PACTL [\$D300] and data register PORTA [\$D301]; port B uses control register PBCTL [\$D302] and data register PORTB [\$D303].

The data register and input/output registers share the same address. In order to read or write the data direction register, bit 2 of the control register must be set to 0, and to read or write the data port, bit 2 must be set to 1.

I/O direction

Each bit in the data direction register controls whether a bit is in input or output mode. A zero bit sets the bit to input mode, while a one bit enables output for that bit. A bit in the output register is ignored when that bit is set to input, but all bits in the input register are valid even for output bits. This is sometimes used to mask off incoming bits; a bit will read as zero if either the PIA or an external device is pulling the line low.

Control lines

The proceed and interrupt lines of the SIO bus are connected to control lines CB2 and CA2 of the PIA, respectively. These are generally unused and disabled by setting bits 0 and 1 of PACTL and PBCTL to zero.

Control lines CB1 and CA1, however, are connected to the SIO command and motor control lines, respectively. Bits 3-5 of PACTL/PBCTL are used to control the line state and should be set to 110 for a low state or 111 for a high state. The command line is pulled low by the Atari while a command is being sent to an SIO device; the motor line is pulled low when a cassette tape deck should begin recording or playback.

The control lines can be used to issue an IRQ to the CPU, but this is seldom useful unless an external SIO device is specially made to take advantage of this ability.

Typically the values \$34 and \$3C are written to PACTL/PBCTL; this disables interrupts, raises or lowers the CA2/CB2 line, and keeps the PORTA/PORTB register in data mode so the OS VBI routine can read the joystick ports.

Reset behavior

The PIA is reset only on startup on the 800; it is also reset by the Reset button on XL/XE models. When the PIA is reset, all registers are cleared to \$00. This disables all interrupts, switches PORTA/PORTB to the data direction register, and sets all peripheral port bits to input mode.

3.3 Bank switching

Bank switching allows the CPU to access more memory than would ordinarily be reachable via the 64K address space dictated by its 16 address lines by multiplexing address regions based on bank switching registers. On the XL series, this allows ROM to be selectively disabled, permitting access to 62K of memory.

800XL banking

The PIA's port B is repurposed in the 800XL to control the memory map instead of the third and fourth controller ports. Bit 0 enables the kernel ROM at \$C000-CFFF and \$D800-FFFF; bit 1 enables the BASIC ROM. BASIC takes precedence over a cartridge, if present. Bit 7 *disables* the self test ROM at \$5000-57FF. By setting PORTB to \$83, it is possible to access 62K of memory. The 2K block of hardware registers at \$D000-D7FF cannot be disabled.

The self-test ROM at \$5000-57FF only appears if the kernel ROM is enabled. In other words, the values \$7E and \$FE have the same effect, disabling both the kernel and self-test ROMs.

The remapping of PORTB means that one must be careful when clearing hardware registers – carelessly clearing memory across the \$D3xx range can cause the kernel ROM to be swapped out, resulting in a system crash. This is one reason for older games failing on XL/XE computers. Whether or not this happens depends on how the port B control register [\$D303] is set, as clearing bit 2 of that register causes the direction register to be accessed at \$D301 instead.

Writes to ROM

The MMU logic maps addresses to circuitry solely based on address. This means that any writes to addresses that are currently assigned to kernel ROM, BASIC ROM, or cartridge ROM are ignored and do not affect the underlying RAM.

BASIC ROM Overlap (XL/XE only)

The priority in the \$A000-BFFF address space is cartridge ROM, then BASIC ROM, and then RAM. If both the cartridge and BASIC ROM are enabled in that area, the cartridge is visible.

3.4 Extended Memory

130XE banking

The 130XE additionally uses bits 2-5 to control access to an additional 64K of memory through a window at 4000-7FFF. Bits 4 and 5 enable CPU and ANTIC access to extra memory when *cleared*. Bits 2 and 3 control the memory bank, selecting one of four extra 16K banks. The CPU and ANTIC must access the same bank of memory if both are using extended memory. There is no way to redirect the 4000-7FFF window to any memory in the primary 64K.

320K modification (RAMBO)

Bits 4-7 of PORTB are used to access 16 extended banks of 16K at 4000-7FFF. This results in $64K + 256K = 320K$ of memory. Because bit 5 is used, ANTIC external memory access is not available if this mod is used on 130XE hardware. Instead, both CPU and ANTIC are switched at the same time.

576K modification

Reusing the BASIC bit (bit 1) raises the number of selectable banks to 32, for a total of $64K + 512K = 576K$. Bits 0, 4, and 7 still control the kernel ROM, CPU access, and the self-test ROM on XL/XE hardware.

1088K modification

Reclaiming the self-test bit (bit 7) gives six bits for bank selection, bits 1-3 and 5-7. 64 banks of 16K plus the main 64K bank gives $64K + 1024K = 1088K$.

3.5 Miscellaneous Connections

TRIG3 Connection (XL/XE only)

On the XL/XE series, the RD5 cartridge line is connected to the trigger 3 input (T3) of GTIA. The RD5 line signals when the cartridge is supplying data in the \$A000-BFFF range and therefore built-in memory should be suppressed. Because RD5 is active high and T3 is active low, the TRIG3 register in GTIA reads as a 0 (button pressed) when cartridge ROM is present and 1 (button not pressed) when it is absent. This is used as a cartridge sense mechanism by the XL/XE OS.

When a cartridge is disabled via bank switching and no longer presenting anything at \$A000-BFFF, TRIG3 reads as a 1.

1200XL option jumpers

The 1200XL has four option jumpers which are connected to unused pot lines. Option jumper J1 is connected to POT4 and causes a self-test on startup if installed.¹

3.6 Examples

Caverns of Mars

This game configures the upper four bits of port A as output in order to force them to zero, and fails to read the joystick if this is not reflected in the values read.

3.7 Further reading

The definitive resource for anything involving the Atari memory map is [CHA85]. Appendix 16 provides information on the new PORTB assignments for the 130XE.

[ATAXL] describes numerous modifications to the hardware and kernel in the 1200XL, such as the option jumpers.

[ATA82] contains both functional and detailed schematics of the Atari 400/800 and is useful in tracing signal flow between the custom chips.

¹ XL addendum, 15

4 ANTIC

ANTIC is the master chip of the Atari 8-bit chipset, controlling frame timing and doing all direct memory access (DMA).

4.1 Addressing

ANTIC occupies the \$D4xx block of address space. Only the low four bits are decoded, so any address of the form \$D4XY will address mirror X of register Y.

4.2 Display modes

ANTIC supports fourteen playfield display modes:

| Mode | Type | Scan lines | Colors | Bytes (normal width) | Resolution | Color mode | Pixel size |
|------|-----------|------------|--------|----------------------|------------|------------|------------|
| 2 | Character | 8 | 1.5 | 40 | 40 | Hi-res | 8x8 |
| 3 | Character | 10 | 1.5 | 40 | 40 | Hi-res | 8x8 |
| 4 | Character | 8 | 5 | 40 | 40 | Lo-res | 8x8 |
| 5 | Character | 16 | 5 | 40 | 40 | Lo-res | 8x16 |
| 6 | Character | 8 | 5 | 20 | 20 | Lo-res | 16x8 |
| 7 | Character | 16 | 5 | 20 | 20 | Lo-res | 16x16 |
| 8 | Bitmap | 8 | 4 | 10 | 40 | Lo-res | 8x8 |
| 9 | Bitmap | 4 | 2 | 10 | 80 | Lo-res | 4x4 |
| A | Bitmap | 4 | 4 | 20 | 80 | Lo-res | 4x4 |
| B | Bitmap | 2 | 2 | 20 | 160 | Lo-res | 2x2 |
| C | Bitmap | 1 | 2 | 20 | 160 | Lo-res | 2x1 |
| D | Bitmap | 2 | 4 | 40 | 160 | Lo-res | 2x2 |
| E | Bitmap | 1 | 4 | 40 | 160 | Lo-res | 2x1 |
| F | Bitmap | 1 | 1.5 | 40 | 320 | Hi-res | 1x1 |

Mode 2

Mode 2 is the standard 40-column screen seen on startup. Each playfield byte selects an 8x8 character from an array of 128 pointed to by CHBASE; bit 7 controls inversion or blinking, based on modes in CHACTL.

The character set requires 1K of memory and must be aligned to a 1K boundary. Each of the 128 characters is described by 8 contiguous bytes, where the first byte corresponds to the data for the first scan line. With each byte, each bit corresponds to a pixel on screen, where bit 7 is the left-most pixel. Because mode 2 is a hi-res mode, the entire playfield uses the PF2 color, and each bit indicates whether luminance comes from PF2 (0 bit), or PF1 (1 bit).

Mode 3

Mode 3 is similar to mode 2, except that each mode line is 10 scan lines tall instead of 8. The extra two scan lines reuse the same data from the first two, but only one of the pairs displays valid data. Characters 00-5F display data for scan lines 0-7, while characters 60-7F display on scan lines 2-9 instead. This permits one-

quarter of the character set to have descenders.

Mode 4

Mode 4 is another character mode that produces 40 characters across in normal width, but unlike modes 2 and 3, mode 4 is a lo-res mode that produces up to five colors. Instead of each character producing monochrome characters in an 8x8 block, each character is instead 4x8 with pixels twice as wide. Normally each pair of bits produces either the background color (00) or PF0-PF2 (01-11). If bit 7 is set, however, the 11 pair produces PF3 instead of PF2.

Mode 5

Mode 5 is the same as mode 4, except that scan lines are repeated once and each character is 16 scan lines tall instead of 8.

Mode 6

Mode 6 is the familiar single-color, double-wide signature character mode of the Atari. At normal width, it produces 20 8x8 characters per row, where each pixel is one color clock wide. The character set is half the size in mode 6, requiring only 512 bytes and 512 byte alignment. Only 64 characters are available in the mode because the upper two bits are used to select the foreground color used by 1 bits, with 00-11 producing PF0-PF3. 0 bits in the character data always produce the background color.

Mode 7

Mode 7 is the same as mode 6, except that scan lines are doubled and each character is 16 scan lines tall.

Mode 8

Mode 8 is the lowest resolution graphics mode, producing 40 pixels across with one of four colors. Bits 7 and 6 of a byte correspond to the left-most pixel; 00 selects the background color while 01-11 produces PF0-PF2. Each pixel is 4 color clocks wide and 8 scan lines tall.

Mode 9

Mode 9 is double the horizontal and vertical resolution of mode 8, with each pixel being 2 color clocks wide and 4 scan lines tall. However, it is only a two-color mode, with each bit selecting the background (0) or PF0 (1). Bit 7 is the left-most pixel in each byte.

Mode A

Mode A is the four-color version of mode 9. Each pixel selects the background (00) or PF0-PF2 (01-11).

Mode B

Mode B increases resolution further to 1 color clock and 2 scan lines per pixel, with two colors per pixel (background and PF0).

Mode C

Mode C is the same as mode B, except that mode lines are only one scan line high. It is the highest resolution two color bitmap mode available.

Mode D

Mode D is the same as mode B, except that each pixel is two bits and selects from one of four colors.

Mode E

Mode E is the same as mode C, except that each pixel is two bits and selects from one of four colors. It is the highest resolution four color bitmap mode available.

Mode F

Mode F produces 320 pixels across at normal width, with each bit corresponding to a pixel one-half color clock wide and one scan line tall. It is a high-resolution mode, meaning that the whole playfield uses the PF2 color and the luminance from either PF2 (0) or PF1 (1).

This mode is also the mode that serves as the basis for the three new modes added with the GTIA; the only difference in setup is that bits 6 and 7 of PRIOR on the GTIA are set to a value other than 00.

4.3 Display list

The display list determines how and when ANTIC fetches playfield data for display through GTIA. It is composed of a series of one-byte or three-byte instructions, each of which controls the display of at least one scan line on screen, and is normally repeated for every frame.

Instruction pointer

DLISTL/DLISTH set the instruction pointer used to fetch the display list. It can be placed anywhere in the 64K address space, but cannot cross a 1K boundary without an explicit jump instruction as only the lower 10 bits increment.²

Any write to DLISTL/DLISTH will immediately change the memory pointer used for the next display list fetch. Because of the possibility of display list interrupts, it is dangerous to do this in the middle of a display list, as changing only one of the address bytes may cause ANTIC to execute random memory as a display list and therefore issue spurious DLIs. A \$C1 instruction is particularly dangerous as it will cause a DLI to activate every scan line until vertical blank and can easily cause a crash. Therefore, the display list pointer should normally only be updated when either display list DMA is disabled or during vertical blank.

Instruction format

A display list instruction is described in a single byte as follows:

| | | | | |
|-----|-----|----|----|------|
| DLI | LMS | VS | HS | Mode |
|-----|-----|----|----|------|

D7 Display list interrupt

² Hardware II.10

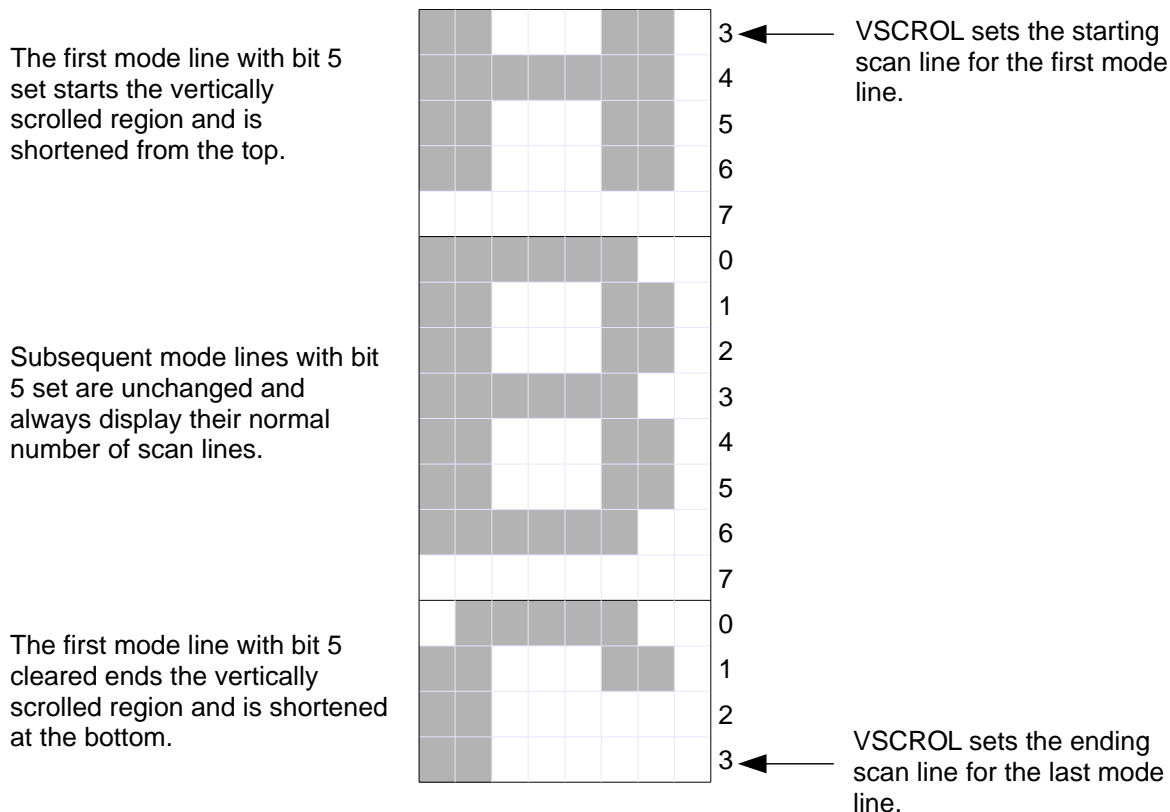


Figure 1: Effect of vertical scrolling on mode lines

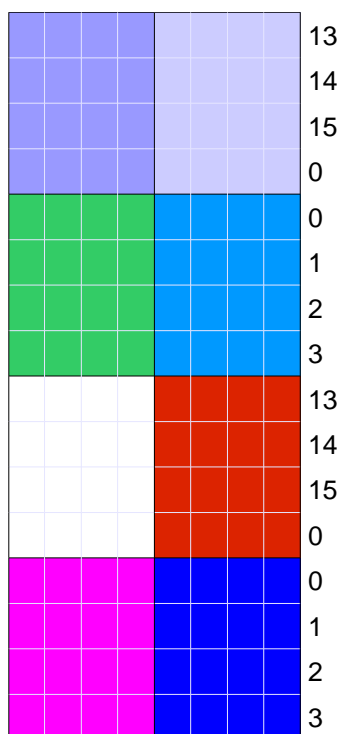
| | |
|-------|--|
| 0 | No interrupt |
| 1 | Interrupt CPU at beginning of last scan line |
| D6 | Load memory scan counter (LMS operation) |
| 0 | Normal |
| 1 | Load memory scan counter with new 16-bit address |
| D5 | Vertical scroll |
| 0 | Disable vertical scrolling |
| 1 | Enable vertical scrolling |
| D4 | Horizontal scroll |
| 0 | Disable horizontal scrolling |
| 1 | Enable horizontal scrolling |
| D0:D3 | Mode |
| 0000 | Blank |
| 0001 | Jump |
| other | Non-blank mode line |

Blank mode lines

A blank mode line is specified by an instruction byte whose lowest four bits are 0000. In this case, bits 4-6 specify a scan line count instead, where 000-111 specify 1-8 scan lines. Thus, a blank mode line is never

Even mode lines begin a vertically scrolled region and start from scan line 13, wrapping around to 0.

Odd mode lines end a vertically scrolled region and stop on scan line 3.



The boundary between vertically scrolled regions is critical. Here VSCROL must be set to 3 and then 13 in quick succession.

Figure 2: Abusing vertical scrolling in the “Turbo GTIA” mode

considered to have scrolling enabled or to initiate an LMS operation.

Jump command

Instruction bytes with a mode of 0001 are jump commands and are always followed by two bytes indicating the new instruction pointer for the display list. This produces a three-byte instruction similar to a 6502 JMP instruction, where the new 16-bit address is specified as low-byte first. Because the jump instruction occupies a display list slot, a blank line is displayed during its execution.

A jump instruction with bit 6 set (\$41) causes ANTIC to suspend the display list until vertical blank and restart it at the new address at the end of VBLANK at scan line 8. This is usually used to terminate the display list and restart it for the next frame. When using a display list that loops using such an instruction, it is not necessary to write DLISTL/DLISTH per frame as ANTIC will autonomously repeat the display list every frame.

Playfield mode lines

Modes 2-F select a playfield mode line for display.

Load Memory Scan (LMS) commands

Setting bit 6 on a non-blank mode line causes the playfield memory scan pointer to be reloaded with a new address from the two following bytes, LSB first. This can be done on any such mode line and as frequently or infrequently as required; no blank line is incurred and the display appears uninterrupted. Normally one LMS is

required at the beginning of the display list to reset the playfield address to the beginning of the screen memory. Screen modes that require more than 4K of memory require at least one other LMS command in the middle of the screen to hop the 4K boundary.

Excessively long display lists

If a display list is too long, ANTIC automatically suspends the display list at the beginning of vertical blank at scan line 248 and resumes it at the end of vertical blank on scan line 8 of the next frame. This means that if a display list were exactly 480 scan lines tall and looped with a jump (\$01) instruction, it would alternate perfectly between two images. Typically this doesn't happen, though, because the vertical blank routine reloads DLISTL/DLISTH.

Suspended display list DMA

If display list DMA is turned off in the middle of the display list, ANTIC reuses the last fetched instruction byte. This essentially repeats the last mode line until display list DMA is re-enabled. Load Memory Scan (LMS) commands simply act when repeated as though the LMS bit (bit 6) were not set.

Turning off display list DMA has no effect if a wait for vertical blank (\$41) instruction is executing, as no fetches occur anyway while the instruction loops.

4.4 Scrolling

Normally, a playfield can only be scrolled by changing the memory scan pointer used to begin fetching data. This restricts scrolling to byte granularity, which is fairly large on-screen for most display modes. ANTIC has support for both fine horizontal and vertical scrolling, which allows playfields to be scrolled more finely than by LMS instructions.

Horizontal scrolling

Setting bit 4 of a display list instruction enables horizontal scrolling for that mode line. This has two effects. First, it causes the displayed data to be delayed and pushed right by a specified amount. Second, it increases the fetch width for narrow and normal playfields to provide extra data to fill in on the left side. This means that when horizontally scrolling a normal playfield, the borders still correspond to normal playfield width, but you must provide enough data for a wide playfield. This also means that the playfield must either be laid out with the pitch for the wider playfield or an LMS instruction must be used on every mode line to reset the fetch pointer.

The scroll amount is controlled by the HSCROL register, which specifies the number of color clocks to delay output. Scrolling is only possible on color clock boundaries, which means that only two pixel horizontal resolution is possible in the high-resolution modes (2, 3, and F). The limited scroll amount means that both fine horizontal scrolling and changing the playfield fetch pointer are required to scroll over a large playfields. However, by combining changes to HSCROL and LMS instructions, it is possible to scroll over a very large playfield without having to modify any playfield data.

Horizontal scrolling gotchas

ANTIC can't fetch additional data for a wide playfield, so in that case it shifts in background color on the left.

The wider playfield fetch width means that more playfield DMA occurs on horizontally scrolled mode lines. In addition, playfield DMA occurs one cycle later for each additional increase by two in HSCROL. This means that special care is required when using DLIs or changing registers mid-screen in horizontally scrolled mode lines, due to the shifting timings and fewer free cycles.

Horizontal scrolling has special implications for high-resolution and GTIA modes. For hi-res modes, scrolling is only possible on two pixel boundaries since each pixel is one-half color clock. For GTIA modes, HSCROL should only be set to even values, because odd values will produce garbled pixels instead of the expected single color clock shift. The reason is that GTIA does not know about horizontal scrolling, so when ANTIC attempts to delay the wider GTIA mode pixels by one color clock, it causes GTIA to incorrectly pair bits into 4-bit pixels and mix pairs of bits from adjacent pixels.

If the horizontal scroll value is changed in the middle of a mode line, the buffered data within ANTIC is displayed at the new scroll position as expected.

Vertical scrolling

Vertical scrolling in ANTIC is controlled by bit 5 of a display list instruction. When bit 5 is set, the VSCROL (D405) register modifies the height of selected mode lines in the display list to allow portions of the display to be scrolled on a scan line basis. When the vertical scrolling bit changes from a 0 to a 1 on adjacent mode lines, the first line for which it is set is shortened by starting it at the scan line specified by VSCROL. Similarly, when it changes from a 1 to a 0, the first line for which that bit is reset is also shortened by ending it at that scan line. This means that a vertically scrolled region consisting of three mode 2 lines will have bit 5 set on the first two lines and occupy $(8 - \text{VSCROL}) + 8 + (\text{VSCROL} + 1) = 17$ scan lines instead of the usual 24.

VSCROL and the row counter are both 4-bit counters, and odd effects can be created by setting them to out of range values. For instance, a mode F scan line is only one scan line high and ordinarily vertical scrolling doesn't make sense. However, if VSCROL is set to 13 upon entering such a scan line, the row counter will count from 13 to 0, creating a mode F region where each pixel is four scan lines tall, but the DMA overhead is still only for one scan line. This is similarly possible when exiting the vertically scrolled region by setting VSCROL to 3 so that the row counter runs from 0 to 3. This creates the so-called "turbo GTIA" mode where GTIA modes can be run with lower vertical resolution with much lower DMA overhead than if LMS lines were used to produce the same effect.

In order to take effect, VSCROL must be written no later than cycle 0 in order to be used for the initial row counter when entering a vertical scrolling region; the deadline is cycle 108 for the test that determines if the following scan line ends a vertical scrolling region. The six clock window between these deadlines can be abused in order to halve the number of DLIs required to implement a turbo mode. This is done by writing VSCROL twice in quick succession, with the first value terminating the current mode line and the second value setting the height of the next.

Vertical scrolling regions do not have to exclusively use the same mode, as the vertical scrolling functionality only affects the starting and ending mode lines via row count.

Blank mode lines (\$x1) are always considered to have the vertical scroll bit cleared since the scroll bits are used for a blank line count instead. The blank line is still subject to shortening if it ends a vertically scrolled region, however.

4.5 Display list interrupts

A display list interrupt (DLI) is a type of non-maskable interrupt triggered off of the display list. It is mainly useful for updating hardware registers at specific points on screen to effect mid-screen changes that are not possible through the display list itself.

Triggering a DLI

To trigger a DLI, bit 7 should be set on a display list instruction. This causes ANTIC to fire an NMI at the start of the *last* scan line for that mode line. Typically the DLI interrupt handler will then issue an STA WSYNC in order to synchronize to the end of the scan line, enabling it to write to hardware registers just prior to the next mode line at a time where the user will not see artifacts from the changes.

You can set the DLI bit on *any* mode line, including a blank mode line. The strangest use is when the DLI bit is set on a wait for vertical blank instruction (\$C1); this causes a DLI to be issued on *every* scan line until vertical blank begins at scan line 248. Obviously, the DLI must be very short to run reliably in this situation, but it is possible.

If vertical scrolling causes a mode line with a DLI to be shortened, the DLI will still fire at the end of the shortened mode line, and just prior to the next mode line. This can cause surprises if a DLI is attempted at the start or end of a vertically scrolled region, because this can cause the DLI to occur on the more strongly contended first scan line.

Writing a DLI routine

If you are running on Atari 400/800 hardware, the NMI vector is hard-coded in ROM and the earliest interception point is the (VDSLST) vector, after which the kernel has already distinguished the DLI from a possible VBI or SYSTEM RESET interrupt. If you are running on an XL/XE machine, however, you can redirect the NMI vector to a custom routine by disabling the kernel ROM, in which case you can write the entire handler yourself.

ANTIC sets bit 7 of NMIST to indicate that a DLI has occurred. Typically, the NMI handler will use a BIT NMIST instruction to check this and then fall through a BPL instruction to enter the DLI handler. Bit 7 of NMIST can be reset by writing to NMIRES, but this is unnecessary as the hardware automatically clears the DLI bit when it sets the VBI bit (bit 6) and pulls NMI for the vertical blank interrupt on scan line 248. Therefore, it is generally unnecessary to write to NMIRES at all as the DLI bit can simply be left set until it is reset by ANTIC.

DLI timing

Display list interrupts have extremely critical timing for two reasons: they have to change hardware registers within a very narrow window of time (usually horizontal blank), and they need to execute quickly to avoid conflicting with each other or stealing too much CPU from mainline and IRQ routines. As such, it is very useful to count exact cycles for DLI execution.

DLI execution proceeds as follows³:

- ANTIC pulls NMI at cycle 8 at the beginning of a scan line, right after display list and P/M DMA.
- The 6502 requires two cycles to acknowledge the NMI⁴.
- 0-6 cycles pass as the 6502 finishes the currently executing instruction.
- Interrupt entry takes 7 cycles.

Thus, if you are writing a custom NMI handler, the earliest that the handler will run is cycle 17. Note that DMA contention will slow down this sequence, and it's virtually guaranteed that at least refresh DMA will interfere starting with cycle 25.

If the OS handler is used, then the OS will execute a BIT NMIST / BPL / JSR (VDSLST) sequence before executing your handler, resulting in an additional 11 cycles of delay. Including refresh DMA, your handler will execute starting on cycle 28-36.

At this point, the normal procedure is to save registers as needed, load up registers with needed values, STA WSYNC, and then write values to hardware registers as quickly as possible during horizontal blank. Afterward, the exit path will frequently spill into the middle of the next scan line, but that is not nearly as critical.

Note that these timings assume that the DLI is occurring on a blank mode line. Any non-blank line will require playfield DMA cycles that will significantly delay interrupt routine execution: a normal-width mode 0 line will shift

³ De Re Atari also has a good description of DLI timing and explains how to break a DLI routine into phases by timing requirements.

⁴ [MOS76] 38

the entry window for the OS case to cycles 36-44, and horizontal scrolling or wide playfields makes this worse. Extra care is required when using DLIs around vertical scrolling, because it can shorten a mode line to only the first scan line, causing a DLI to fire on a scan line where the active region is blocked by solid playfield DMA. The extreme case occurs if the next mode line is also a character mode line, which can result in so much DMA contention that *two entire scan lines* pass before the 6502 can even enter the DLI handler.

NMI enable timing

Writes to NMIEEN to enable an interrupt must occur by cycle 7 on a scan line in order for the interrupt to fire. If the write occurs on cycle 8 or later, the interrupt is considered disabled and no NMI will occur, although the related bit in NMIST will be set.

A write to enable an interrupt that occurs exactly on cycle 7 will cause the interrupt to take place one cycle later than usual. This means that an NMI cannot interrupt the following sequence, even though interrupts normally activate on cycle 10:

```
STA NMIEEN    ;cycles 4, 5, 6, 7
LDA #$01      ;cycles 8, 9
STA PORTB     ;cycles 10, 11, 12, 13
```

Missed NMIs

If the 6502 responds to an IRQ at exactly cycles 4-10, any NMI that ANTIC would have triggered on cycle 8 will be lost.⁵ This includes DLIs, VBIs, and on the 400/800, the SYSTEM RESET interrupt. NMIST is still updated as usual, however. The most visible artifact caused by this problem is glitching on screen if you attempt to use DLIs while an SIO transfer is in progress. However, it can happen with *any* IRQ source, including POKEY timers. It can also occur with an exactly timed BRK instruction. It cannot, however, occur with a regular instruction, not even one that takes seven cycles (INC abs,X).

4.6 WSYNC

A write to WSYNC [D40A] halts CPU execution until the end of a scan line, allowing the CPU to synchronize to the display. One more cycle elapses before being halted until cycle 105. Because the CPU usually gets to execute the first cycle of the next instruction, this appears as if the instruction started on cycle 104. There are, however, three circumstances that can change this behavior:

- **If the cycle immediately after writing WSYNC is blocked.**

In this case, the CPU doesn't get to execute the first cycle of the next instruction, and that instruction starts from the beginning as usual on cycle 105.

- **If playfield DMA extends to cycle 105.**

Wide playfields, normal playfields with horizontal scrolling, and narrow playfields with high horizontal scroll values can encroach on cycle 105. This causes a one-cycle delay in the CPU restart.

- **If refresh DMA extends to cycle 105 or 106.**

The first scan line of a character mode line can incur solid playfield DMA during the active region such that refresh DMA is pushed all the way to the end of the scan line. This can cause refresh DMA to occupy cycle 105, resulting in a one-cycle delay for the CPU. If playfield DMA is already occupying cycle 105, however, then it will push refresh DMA to cycle 106, resulting in a two-cycle delay.

⁵ Speculation on the AtariAge forums is that this is caused by a bug in ANTIC, which does not assert the NMI line long enough for the CPU to reliably acknowledge the interrupt.

These factors mean that there can be up to a three cycle variance in when an instruction following a write to WSYNC finishes execution, not counting interrupts. Therefore, if you are attempting to use a write to WSYNC to establish an event at an exact time on a scan line, your best bet is to write to WSYNC twice during vertical blank or during blank mode lines, ensuring that no DMA interference occurs. You should also ensure that a DLI or VBI does not take place on the scan line as otherwise the interrupt is guaranteed to fire immediately after the instruction that writes to WSYNC.

Because the 6502 can only respond to interrupts at the end of an instruction, a write to WSYNC can cause long delays in interrupt response time. This is particularly problematic for DLIs, which can be pushed down by an entire scan line. Therefore, STA WSYNC should be avoided in main code when time-critical DLIs are in use. A loop on VCOUNT is a popular alternative:

```
      LDA VCOUNT
LOOP  CMP VCOUNT
      BEQ LOOP
```

Execution resumes anywhere between cycles 0-6 of the next scan line.

Read-modify-write instructions should not be used to strobe WSYNC as they cause two writes to the destination address. However, because the write cycles are back-to-back, an INC WSYNC does **not** cause a two scan line delay.

4.7 VCOUNT

The VCOUNT [D40B] register reflects bits 1-8 of the vertical scan counter. Bit 0 is not connected, so this only permits two-line resolution. VCOUNT increments on cycle 111 of a scan line. For an NTSC machine, VCOUNT counts from \$00 to \$82; for PAL, it counts to \$9B.

If you are using VCOUNT to check for a scan line near the top of the screen, consider using a greater-equal check rather than an equality check, as otherwise the test can lock up if the VBI handler takes too long to execute. This is a common cause of lockup when programs meant for PAL are run under NTSC, where there is much less vertical blank time.

4.8 Player/missile DMA

ANTIC can fetch graphics data for players and missiles on behalf of GTIA. Bit 3 of DMACTL enables player DMA, and bit 2 of DMACTL enables missile DMA. Missile DMA is also enabled if player DMA is enabled in order to preserve proper timing against GTIA.

Bit 4 of DMACTL switches between two-line and one-line resolution. This simply changes the addressing that ANTIC uses to fetch player data. If one-line resolution is selected (bit 4 = 1), each player/missile occupies 256 bytes of memory and unique data is fetched per scan line. If two-line resolution is selected, each player/missile occupies 128 bytes of memory and each byte is used for two scan lines.

The address of player/missile data is specified by PMBASE [D407]. In two-line resolution mode, player/missile data must be aligned on a 1K boundary and the upper six bits of the address are specified by bits 2-7 of PMBASE. In one-line resolution mode, P/M data must be aligned on a 2K boundary and the upper five bits of the address are specified by bits 3-7 of PMBASE.

4.9 DMA timing

Memory refresh DMA

Nine cycles of refresh DMA occur on every scan line in order to refresh DRAM, starting at cycle 25. These

refresh cycles occur even in vertical blank. Refresh DMA can be blocked by playfield DMA, in which case the refresh cycle occurs on the next free cycle. If more than one refresh cycle is blocked, the additional refresh cycles are lost and only one occurs. This only occurs in the first scan line of modes 2-5, where memory is so contended that only 1-2 refreshes can fit.

In wide character modes, the final refresh cycle can be pushed all the way to the end of playfield DMA at cycle 105 or 106, resulting in an additional cycle of delay for a WSYNC on that scan line.

Player/missile graphics DMA

P/M graphics occupy five DMA slots in the scan line, cycle 0 for missiles and cycles 2-5 for players. If enabled, this fetch occurs for each scan line in the visible range (8 to 247)⁶. No P/M DMA occurs during vertical blank. Although player DMA and missile DMA can be controlled independently, missiles are always fetched if players are enabled.⁷

Note that the CGIA documentation is incorrect in that DMA cycles are stolen on every scan line even if two-line resolution is enabled.

Display list DMA

The display list requires one DMA cycle for each mode byte, which occurs at cycle 1, between players and missiles. Mode lines that perform an LMS or a jump also fetch an additional address word at cycles 6 and 7. This fetch occurs at the beginning of the scan line where the mode line takes effect visually.

For modes that span multiple scan lines, the display list fetch only occurs on the first scan line.

Playfield DMA

Three playfield widths are available: narrow, normal, and wide. Normal playfields are 80 cycles wide, while narrow playfields are 64 cycles and wide playfields are 96 cycles long. All fetch windows have the same center, with each wider setting adding 8 clocks on each side. There is a hardware stop that prevents playfield DMA from going beyond clock 105. Any fetch cycles that would occur on clock 106 or later are suppressed, although the playfield memory address is still incremented.

Enabling horizontal scrolling automatically causes narrow playfields to use the normal fetch window, and normal playfields to use the wide fetch window. No additional data is fetched for wide scrolled playfields. Horizontal scrolling causes the playfield fetch window to be delayed by one cycle for every two color clocks of scroll. The additional color clock delay required by odd scroll values is given by internal buffering.

Mapped mode playfield DMA

The mapped graphics modes have three horizontal densities, resulting in fetches every eight clock cycles (modes 8-9), four cycles (modes A-C), or two cycles (D-F). These occur on the first scan line of the mode. ANTIC internally buffers the data so that modes that span more than one scan line do not need to fetch any data on subsequent scan lines. This is used to powerful effect in the so called "turbo GTIA" modes, where mode F lines are extended to four scan lines by vertical scroll trickery, resulting in one-fourth vertical resolution with one-fourth the bandwidth requirements.

Mapped playfield DMA begins at clock 26, 18, or 10 depending on width.

⁶ [AHS00] 45. Note that the cycle counts are incorrect, as it should be 240 per field for missiles and 960 for players.

⁷ [ATA82] II.20.

Character mode playfield DMA

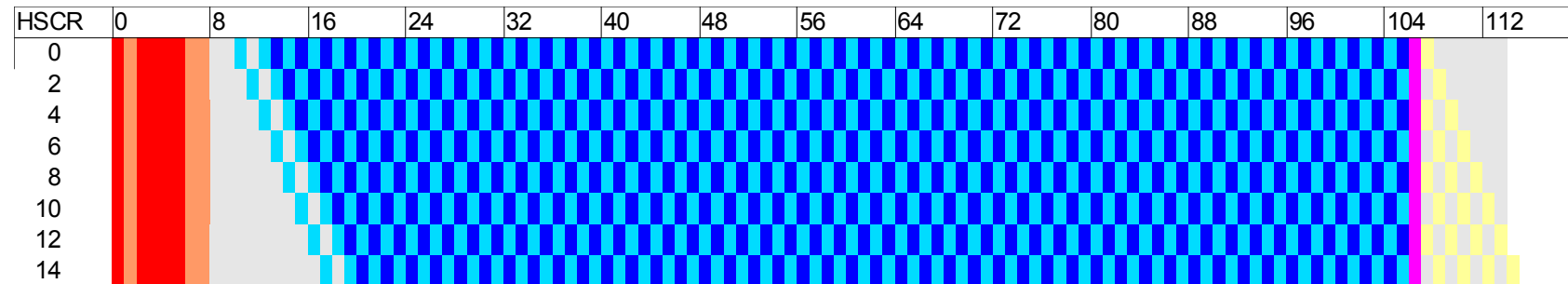
Character modes have two horizontal densities, resulting in name fetches every two clock cycles (modes 2-5) or every four clocks (modes 6-7). The character names are fetched with the same timing as for mapped mode data, at clocks 28, 20, and 12 for the various widths.

Additionally, in these modes the character data itself must be fetched. The data fetch occurs three clocks later than the name fetch. Although the names are buffered internally by ANTIC, the character data isn't, and is always fetched regardless of whether double-height modes are used (modes 5 and 7).

DMA timing charts

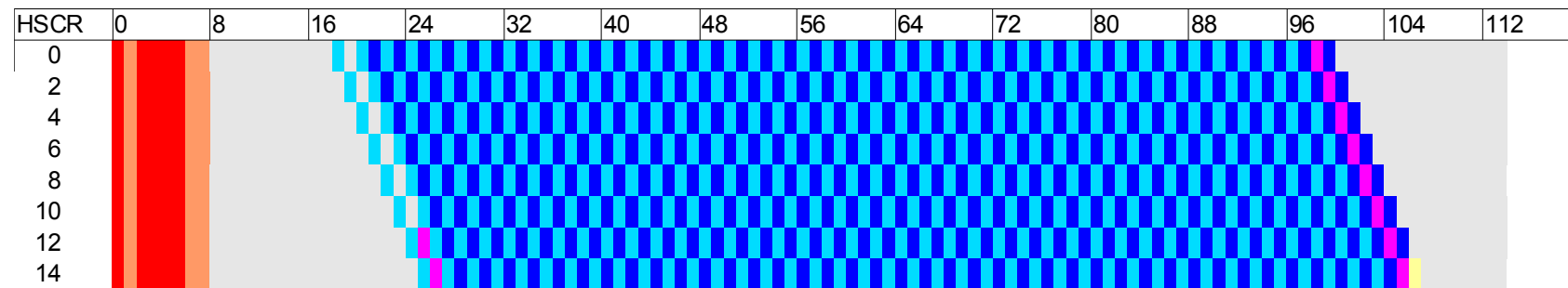
The following charts show the timing of per scan line DMA, based on various modes and settings. IR mode, playfield width, P/M graphics, LMS instructions, and horizontal scrolling all affect DMA timing. Note that the charts are arranged by fetch width, so a narrow playfield with horizontal scrolling is actually described by the narrow playfield chart.

ANTIC modes 2-5, mode line, wide playfield



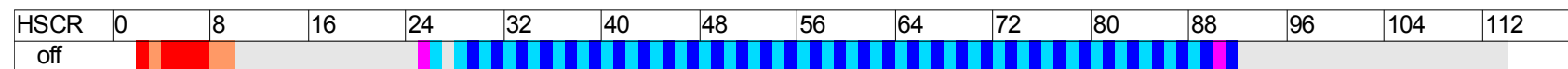
■ Player/missile graphics
 ■ Memory refresh
 ■ Playfield DMA
 ■ Character map DMA
 ■ Display list DMA
 ■ Blocked DMA

ANTIC modes 2-5, mode line, normal playfield



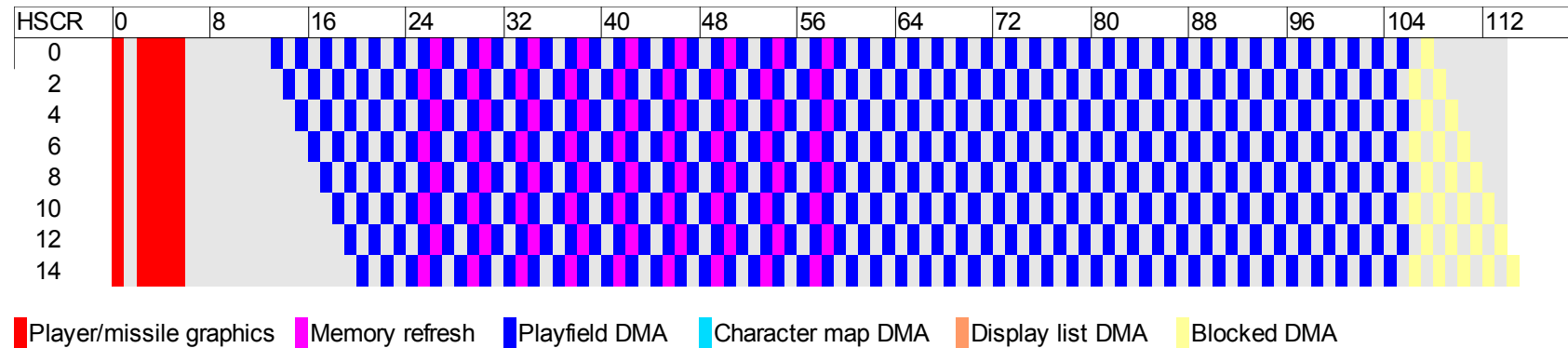
■ Player/missile graphics
 ■ Memory refresh
 ■ Playfield DMA
 ■ Character map DMA
 ■ Display list DMA
 ■ Blocked DMA

ANTIC mode 2-5, mode line, narrow playfield

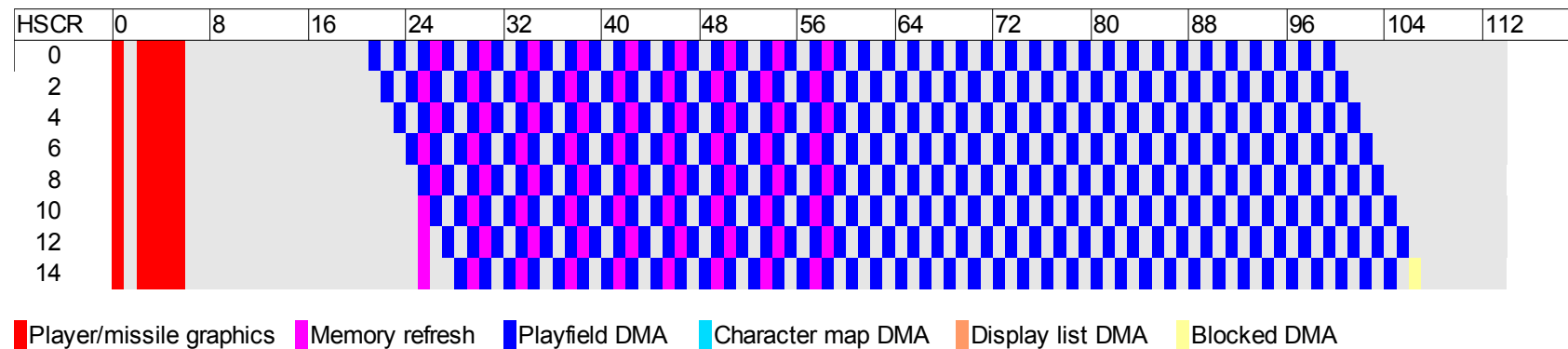


■ Player/missile graphics
 ■ Memory refresh
 ■ Playfield DMA
 ■ Character map DMA
 ■ Display list DMA
 ■ Blocked DMA

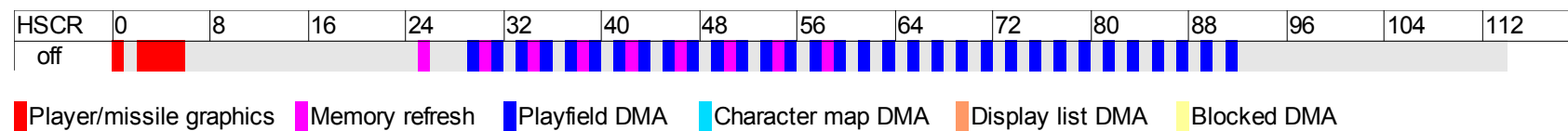
ANTIC modes 2-5, subsequent lines, wide playfield



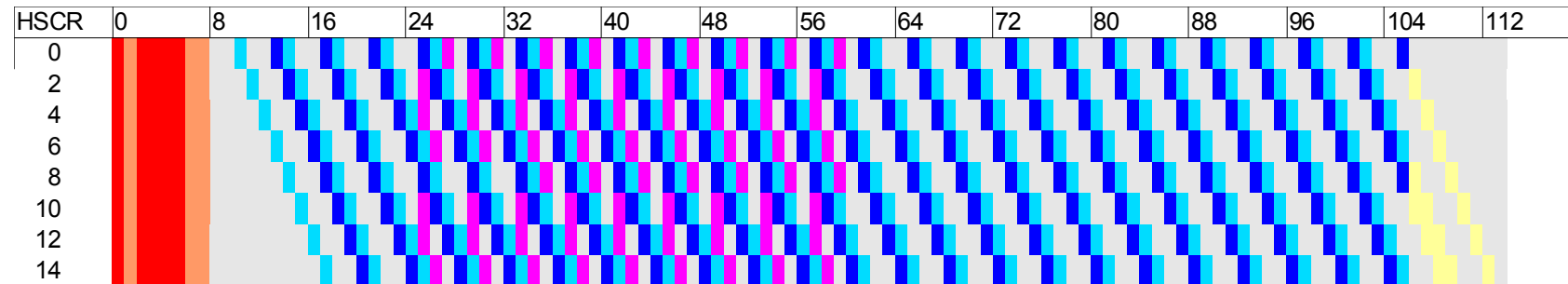
ANTIC modes 2-5, subsequent lines, normal playfield



ANTIC modes 2-5, subsequent lines, narrow playfield

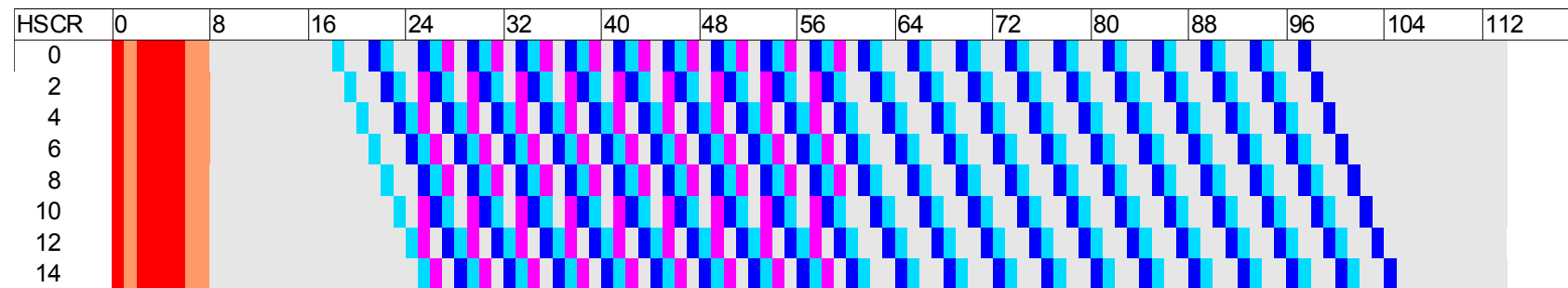


ANTIC modes 6 and 7, mode line, wide playfield



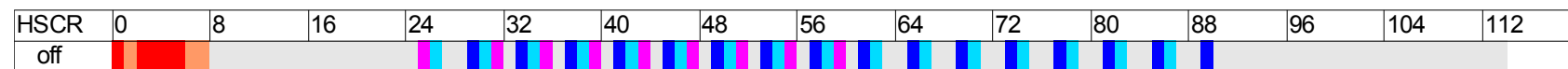
■ Player/missile graphics
 ■ Memory refresh
 ■ Playfield DMA
 ■ Character map DMA
 ■ Display list DMA
 ■ Blocked DMA

ANTIC modes 6 and 7, mode line, normal playfield



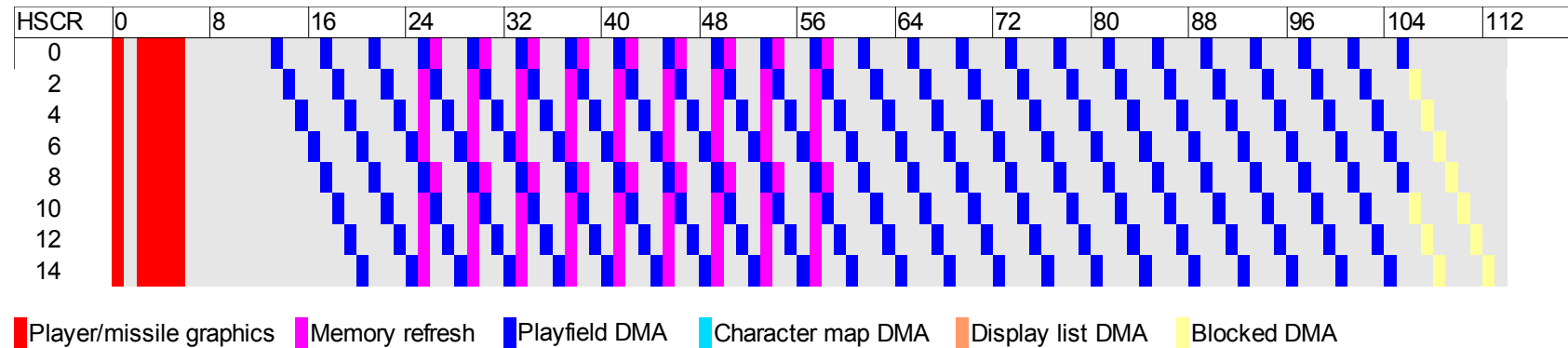
■ Player/missile graphics
 ■ Memory refresh
 ■ Playfield DMA
 ■ Character map DMA
 ■ Display list DMA
 ■ Blocked DMA

ANTIC modes 6 and 7, mode line, narrow playfield

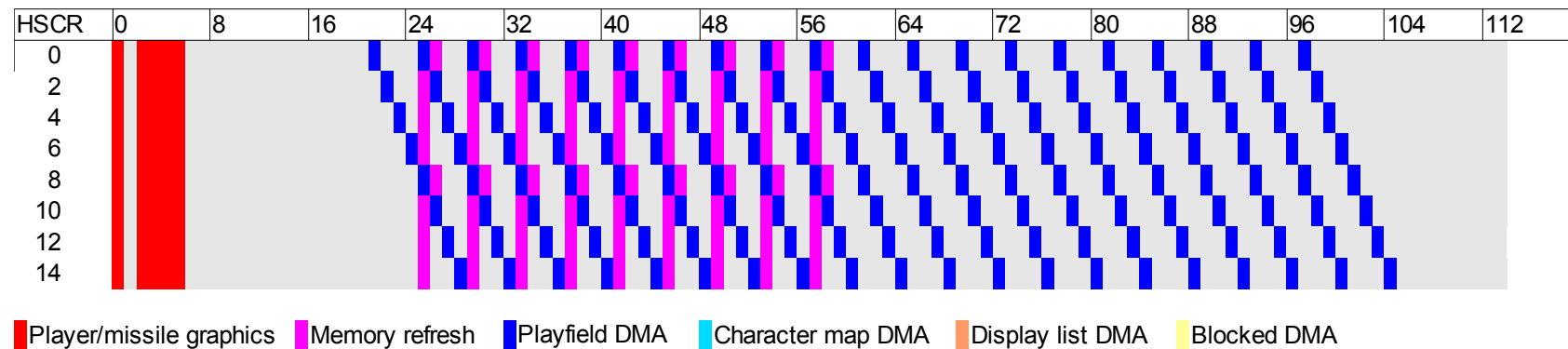


■ Player/missile graphics
 ■ Memory refresh
 ■ Playfield DMA
 ■ Character map DMA
 ■ Display list DMA
 ■ Blocked DMA

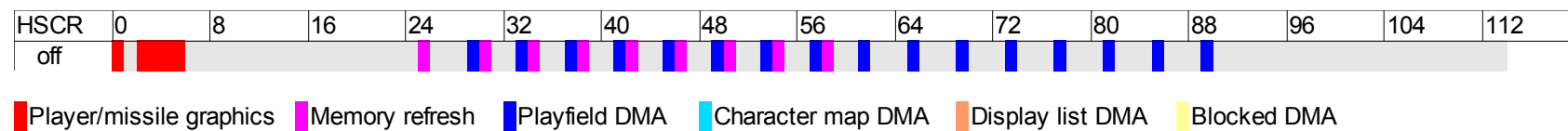
ANTIC modes 6 and 7, subsequent lines, wide playfield



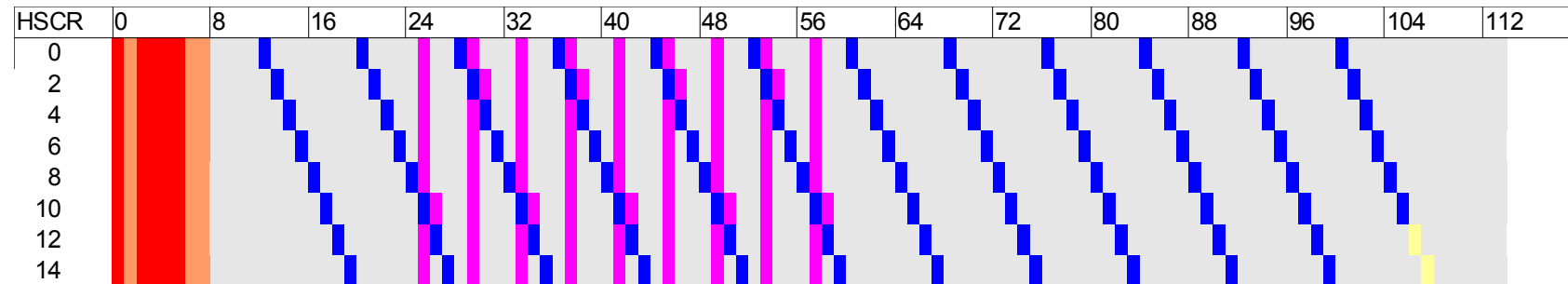
ANTIC modes 6 and 7, subsequent lines, normal playfield



ANTIC modes 6 and 7, subsequent lines, narrow playfield

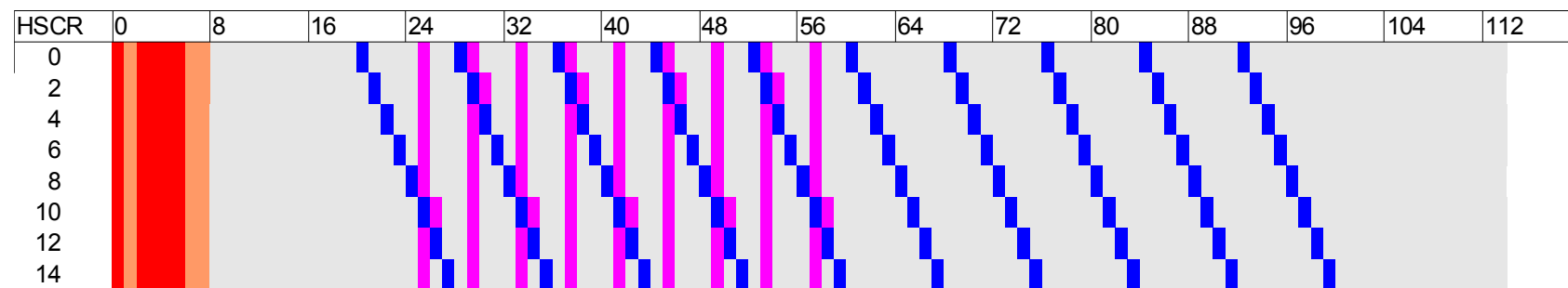


ANTIC modes 8 and 9, mode line, wide playfield



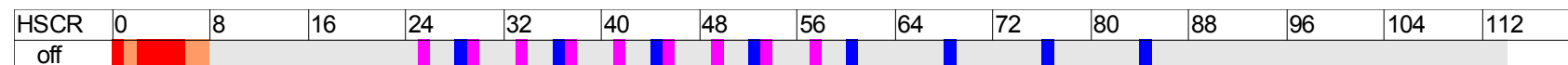
■ Player/missile graphics ■ Memory refresh ■ Playfield DMA ■ Character map DMA ■ Display list DMA ■ Blocked DMA

ANTIC modes 8 and 9, mode line, normal playfield



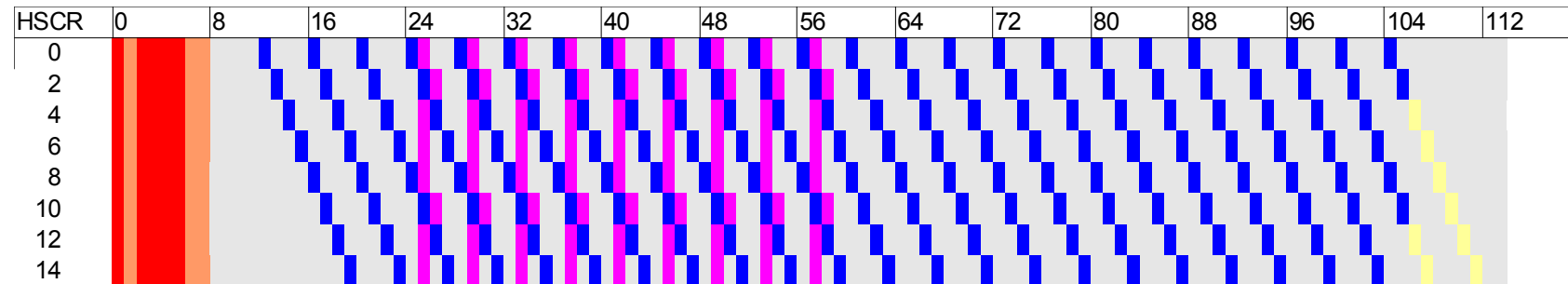
■ Player/missile graphics ■ Memory refresh ■ Playfield DMA ■ Character map DMA ■ Display list DMA ■ Blocked DMA

ANTIC modes 8 and 9, mode line, narrow playfield



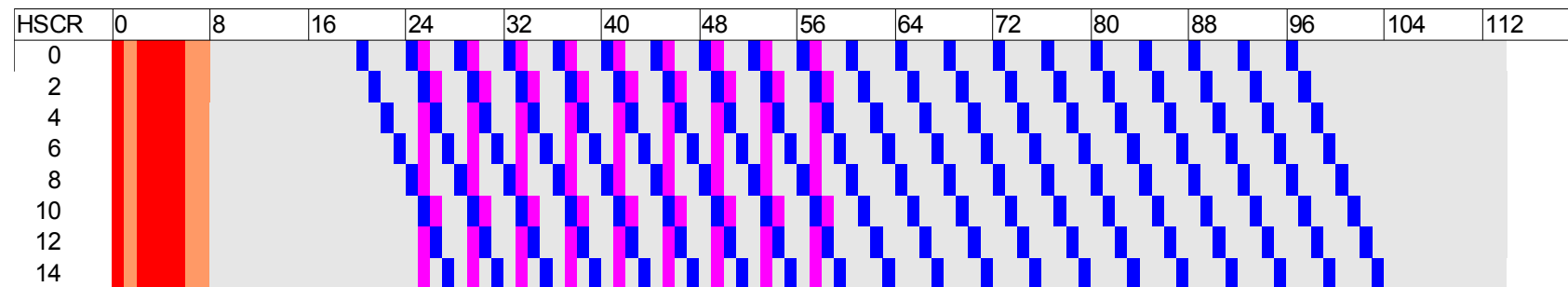
■ Player/missile graphics ■ Memory refresh ■ Playfield DMA ■ Character map DMA ■ Display list DMA ■ Blocked DMA

ANTIC modes A-C, mode line, wide playfield



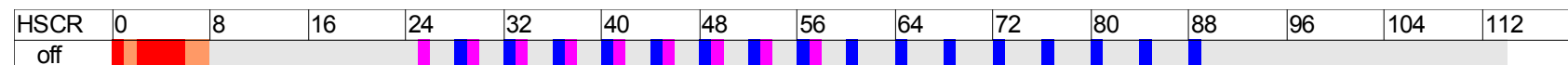
█ Player/missile graphics
 █ Memory refresh
 █ Playfield DMA
 █ Character map DMA
 █ Display list DMA
 █ Blocked DMA

ANTIC modes A-C, mode line, normal playfield



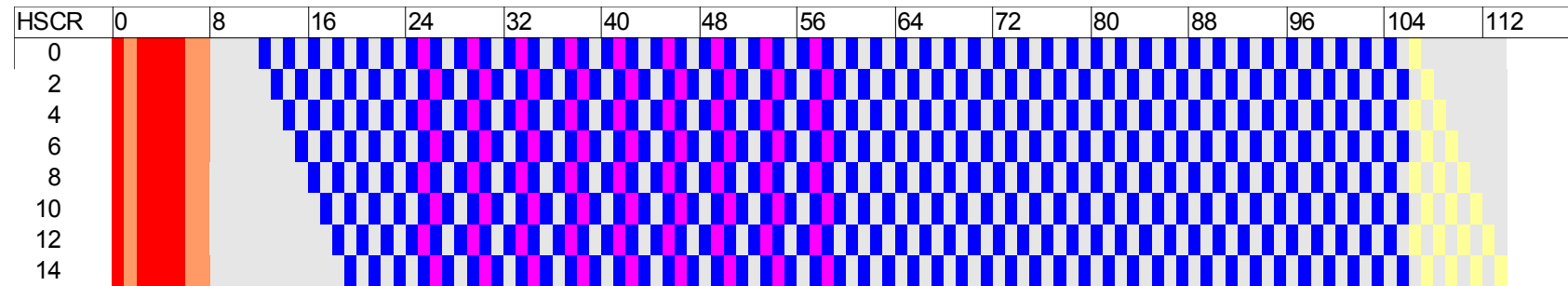
█ Player/missile graphics
 █ Memory refresh
 █ Playfield DMA
 █ Character map DMA
 █ Display list DMA
 █ Blocked DMA

ANTIC modes A-C, mode line, narrow playfield



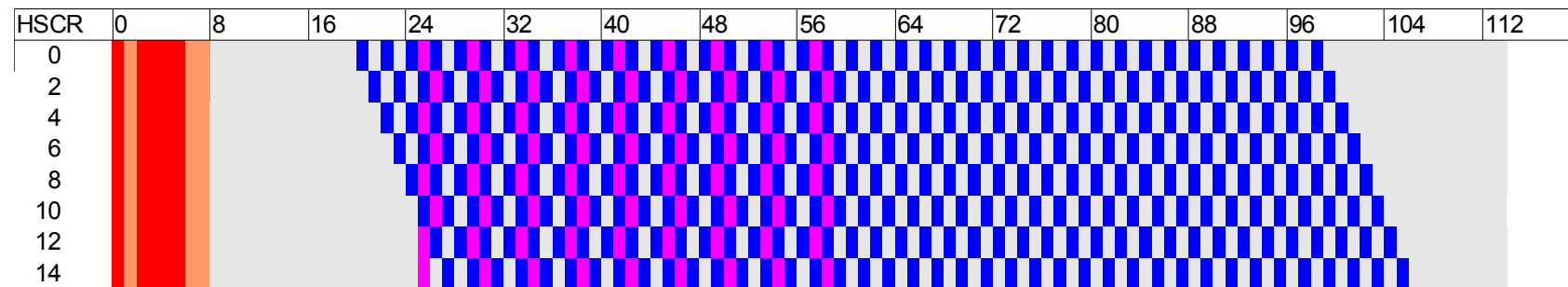
█ Player/missile graphics
 █ Memory refresh
 █ Playfield DMA
 █ Character map DMA
 █ Display list DMA
 █ Blocked DMA

ANTIC modes D-F, mode line, wide playfield



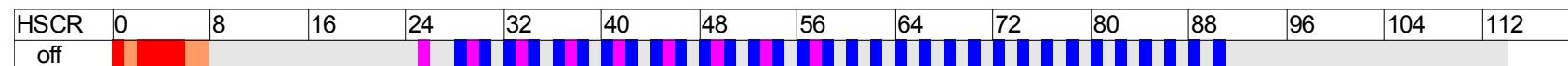
■ Player/missile graphics ■ Memory refresh ■ Playfield DMA ■ Character map DMA ■ Display list DMA ■ Blocked DMA

ANTIC modes D-F, mode line, normal playfield



■ Player/missile graphics ■ Memory refresh ■ Playfield DMA ■ Character map DMA ■ Display list DMA ■ Blocked DMA

ANTIC modes D-F, mode line, narrow playfield



■ Player/missile graphics ■ Memory refresh ■ Playfield DMA ■ Character map DMA ■ Display list DMA ■ Blocked DMA

4.10 Cycle counting example

Let's assume that we want to schedule a series of palette color changes between lines of 40-column text (ANTIC mode 2). To do this, we use the following DLI routine:

```

PHA
TXA
PHA
LDX NEWCL1
LDA NEWCL2
STA WSYNC
STX COLPF1
STA COLPF2
PLA
TAX
PLA
RTI

```

The DLI fires at the beginning of scan line 7 of the previous mode line. Assuming the worst case of a seven clock instruction in progress, the CPU wouldn't start interrupt processing until six clocks later, leading to the following timing analysis:

| HPOS | CPU | ANTIC | HPOS | CPU | ANTIC |
|------|-----------------------------|---------------|------|-------------------|---------------|
| 10 | Finish last instruction (1) | | 42 | | Refresh DMA |
| 11 | Finish last instruction (2) | | 43 | | Playfield DMA |
| 12 | Finish last instruction (3) | | 44 | BPL not taken (1) | |
| 13 | Finish last instruction (4) | | 45 | | Playfield DMA |
| 14 | Finish last instruction (5) | | 46 | | Refresh DMA |
| 15 | Finish last instruction (6) | | 47 | | Playfield DMA |
| 16 | Dummy fetch | | 48 | BPL not taken (2) | |
| 17 | Dummy fetch | | 49 | | Playfield DMA |
| 18 | Push PCH | | 50 | | Refresh DMA |
| 19 | Push PCL | | 51 | | Playfield DMA |
| 20 | Push P | | 52 | JMP (VDSLST) (1) | |
| 21 | | Playfield DMA | 53 | | Playfield DMA |
| 22 | Read NMI vector low | | 54 | | Refresh DMA |
| 23 | | Playfield DMA | 55 | | Playfield DMA |
| 24 | Read NMI vector high | | 56 | JMP (VDSLST) (2) | |
| 25 | | Playfield DMA | 57 | | Playfield DMA |
| 26 | | Refresh DMA | 58 | | Refresh DMA |
| 27 | | Playfield DMA | 59 | | Playfield DMA |
| 28 | BIT NMIST (1) | | 60 | JMP (VDSLST) (3) | |
| 29 | | Playfield DMA | 61 | | Playfield DMA |
| 30 | | Refresh DMA | 62 | JMP (VDSLST) (4) | |
| 31 | | Playfield DMA | 63 | | Playfield DMA |
| 32 | BIT NMIST (2) | | 64 | JMP (VDSLST) (5) | |
| 33 | | Playfield DMA | 65 | | Playfield DMA |
| 34 | | Refresh DMA | 66 | PHA (1) | |
| 35 | | Playfield DMA | 67 | | Playfield DMA |
| 36 | BIT NMIST (3) | | 68 | PHA (2) | |
| 37 | | Playfield DMA | 69 | | Playfield DMA |
| 38 | | Refresh DMA | 70 | PHA (3) | |
| 39 | | Playfield DMA | 71 | | Playfield DMA |
| 40 | BIT NMIST (4) | | 72 | TXA (1) | |
| 41 | | Playfield DMA | 73 | | Playfield DMA |

| HPOS | CPU | ANTIC | HPOS | CPU | ANTIC |
|------|----------------|---------------|------|----------------|------------------|
| 74 | TXA (2) | | 102 | Halted | |
| 75 | | Playfield DMA | 103 | Halted | |
| 76 | PHA (1) | | 104 | Halted | |
| 77 | | Playfield DMA | 105 | STX COLPF1 (2) | |
| 78 | PHA (2) | | 106 | STX COLPF1 (3) | |
| 79 | | Playfield DMA | 107 | STX COLPF1 (4) | |
| 80 | PHA (3) | | 108 | STA COLPF2 (1) | |
| 81 | | Playfield DMA | 109 | STA COLPF2 (2) | |
| 82 | LDX NEWCL1 (1) | | 110 | STA COLPF2 (3) | |
| 83 | | Playfield DMA | 111 | STA COLPF2 (4) | |
| 84 | LDX NEWCL1 (2) | | 112 | PLA (1) | |
| 85 | | Playfield DMA | 113 | PLA (2) | |
| 86 | LDX NEWCL1 (3) | | 0 | PLA (3) | |
| 87 | | Playfield DMA | 1 | | Display list DMA |
| 88 | LDA NEWCL2 (1) | | 2 | PLA (4) | |
| 89 | | Playfield DMA | 3 | TAX (1) | |
| 90 | LDA NEWCL2 (2) | | 4 | TAX (2) | |
| 91 | | Playfield DMA | 5 | PLA (1) | |
| 92 | LDA NEWCL2 (2) | | 6 | PLA (2) | |
| 93 | | Playfield DMA | 7 | PLA (3) | |
| 94 | STA WSYNC (1) | | 8 | PLA (4) | |
| 95 | | Playfield DMA | 9 | RTI (1) | |
| 96 | STA WSYNC (2) | | 10 | RTI (2) | |
| 97 | | Playfield DMA | 11 | RTI (3) | |
| 98 | STA WSYNC (3) | | 12 | RTI (4) | |
| 99 | | Playfield DMA | 13 | RTI (5) | |
| 100 | STA WSYNC (4) | | 14 | RTI (6) | |
| 101 | STX COLPF1 (1) | | | | |

Note that it is crucial that the last write, the write to COLPF2, occurs before cycle 18 on the next scan line. That scan line is the beginning of a character mode line, and the first scan line of a character mode line has 100% DMA contention in the visible region due to the need to fetch both name and character data. Failing to hit that point before cycle 18 would result in the last color change missing the scan line. The remaining portion of the DLI is non-critical and can safely miss this deadline.

4.11 Examples

Zaxxon II

This game uses a display list interrupt (DLI) on a scan line that is highly contended, with a scrolled normal width playfield and P/M graphics active. As a result, the 6502 is unable to read NMIST until past the standard interrupt cycle on the next scan line, and the DLI bit must remain active for more than a full scan line for Zaxxon to work correctly.

Race in Space

Unusually, the interrupt flag is set on the wait for VBL instruction at the end of the display list for the title screen. The game relies on the high number of interrupts that this generates; failing to generate an interrupt per scan line results in the title screen scrolling very slowly or never completing.

Race in Space also uses player collisions against a hi-res (mode F) playfield.

Numen

A lot of tricks are used in this demo, but it almost immediately goes into the “turbo GTIA” mode where VSCROL is alternated to generate mode F with four scan lines per row and one-quarter the DMA overhead.

Bounty Bob Strikes Back!

This game loops on an alias of the VCOUNT register, \$D47B, and jams on startup if address mirroring is not supported.

Chicken

The display list for *Chicken* contains a vertical scrolling region that ends on a blank mode line. The vertical scroll interaction causes this mode line to be variably extended beyond its usual one-scan-line height.

Tarzan of the Apes

The mid-screen DLI routine for the title screen of this game expects VCOUNT to roll over prior to P/M DMA at the start of the next scan line.

Atomix Plus!

There is a buggy loop in this game for copying memory below the kernel ROM (\$D800-FFFF) that enables ANTIC interrupts before re-enabling the kernel ROM. It relies on a DLI or VBI never interrupting the following sequence:

```
LDA #$40
STA NMIEN
LDA #$01
STA PORTB
```

4.12 Further reading

Consult [ATA82] for a overviews and register descriptions for ANTIC. Surprisingly, there is little, if any, additional information in the formerly confidential chip document [AHS99]. A bit more information can be found in [AHS00] , but the accuracy of the additional information appears questionable.

[CRA82] notes a number of nuances about programming ANTIC, most notably the tricky timing in display list interrupts. Note that there appear to be some slight timing discrepancies compared to the real Atari.

5 POKEY

5.1 Addressing

POKEY occupies the \$D2xx block of memory. Only the lowest four bits are significant, so any access of the form \$D2xy accesses mirror x of register y.

One popular modification involves piggybacking a second POKEY onto the system and using address line A4 to select between them. In that case, the even mirrors select the main POKEY, and the odd mirrors select the secondary one. The secondary POKEY has less functionality available due to missing interrupt and I/O connections.

5.2 Initialization

POKEY does not have a RESET line and therefore powers up in indeterminate state. IRQEN must be reset prior to clearing the I bit on the CPU to avoid stray IRQs.

Bits 0 and 1 of SKCTL normally control the keyboard scan and debounce features. However, clearing both of those bits also activates another initialization function, which causes the 15KHz clock, 64KHz clock, serial port hardware, and polynomial noise generators to be reset.

The initialization function can be used to reset the 15KHz and 64KHz clocks to known offsets in the scan line. However, both clocks will have significant offsets from when initialization mode ends, due to the counters being reset to the middle of their cycles. Initialization mode must be asserted for at least seven cycles to ensure that both clocks are fully reset; afterward, the 15KHz and 64KHz clocks will first fire approximately 89 and 11 cycles from when initialization mode ends.

5.3 Sound generation

POKEY has four audio channels with individual timers and audio output circuitry. Each channel has an associated frequency register (AUDF1-4) and control register (AUDC1-4). In addition, there is a shared control register (AUDCTL) for common functions.

Countdown timers

Each channel has an 8-bit countdown timer associated with it to produce clocking pulses. The period for each timer is set by the AUDFx register, specifying a divisor from 1 (\$00) to 256 (\$FF). The countdown timer produces a pulse each time it underflows and resets, which can then be used to drive an interrupt, the serial port, or sound generation.

Three clocks can be used as input to the timers. AUDCTL bit 0 selects either the 15KHz clock or the 64KHz clock as input to the timers. In addition, timers 1 and 3 can be switched to the 1.79MHz clock through AUDCTL bits 6 and 5. Timers 2 and 4, on the other hand, can be switched to use timers 1 and 3 respectively as clocks via bits 4 and 3, pairing timers to produce a more accurate 16-bit timer instead. Note that both audio channels are still active when counters are paired, so normally the output volume for channels 1 and 3 should be muted when used as clocks for channels 2 and 4.

The timers take a small amount of time to reload. With the slow 15KHz or 64KHz clocks, the reload is hidden by the clock, but with the fast 1.79MHz clock, the reload causes a slight change in the timer period. For an 8-bit fast timer with AUDFx value N, the actual period is N+4 clocks, and for a combined 16-bit timer, the period is N+7 clocks.

Waveform selection

Bits 5-7 of AUDCx control the waveform used by the audio circuitry for a channel. This allows each channel to produce a flat level (no output), a square wave, or a more complex wave driven by the polynomial noise generators.

Bit 5 selects either noise (0) or a square wave (1). When the square wave is enabled, each time the timer expires and the output circuitry is clocked, the output toggles, resulting in a square wave with a frequency half that of the timer. When noise is enabled, bit 6 selects either the 9/17-bit generator (0) or the 4-bit generator (1).

Bit 7 controls the sampling mode. If it is set, the timer output directly clocks the output waveform. If it is cleared, however, the 5-bit generator masks out some of the clock pulses, giving a rougher sound.

Due to the short periods of most of the pseudorandom noise generators, it is possible to have undesirable interactions between the period of the countdown timer and the period of the noise generator. For instance, a channel using the 64KHz clock and an AUDFx value of \$CC has a period of 5740 clocks. When used with the 5-bit noise generator, five different sounds can result because the 5-bit generator has a period of 15 and the timer period is divisible by 5, meaning that only three bits of the pattern are used. Exactly which three are used depends on when the sound is started. In a more extreme case, \$D1 would produce no noise at all, because the period is 5880 clocks, which is divisible by 15 – meaning that it will always sample the same bit from the noise pattern.

Volume control

Bits 0-3 of AUDCx control the volume level for a channel, from 0 (silent) to 15 (maximum volume). The volume level only matters if the channel output is currently a 1; if it is a 0, then there will be no output from the channel regardless of the volume level.

Volume output from POKEY is non-linear in that adding two channels of equal volume doesn't produce output with twice the amplitude, but somewhat less. Instead, two channels at volume 15 will only be about 50% louder as each one individually. This has the effect of compressing the output, amplifying quieter sounds and attenuating louder ones.

Volume-only mode

Bit 4 of AUDC1-4 activates volume-only mode for a channel. This causes the channel output to be forced to a 1, ignoring the output of the timer, noise generators, and high-pass logic, and only producing sound based on the volume set by bits 0-3 of AUDCx. This is often used for playback of digital sound effects at 4-bit/sample precision.

Note that because the volume-only mode is enforced after the high-pass logic, the normal inversion of channels 1 and 2 relative to 3 and 4 doesn't apply to this mode; volume-only channels will add in any combination.

High-pass filter

Channels 1 and 2 have a high-pass filter which is enabled by bits 2 and 1 of AUDCTL, respectively. The filter works by XORing the signal against a point-sampled version of itself, which crudely blocks lower frequencies. Channels 3 and 4 control the rate at which the flip-flop is updated and thus the sampling rate for the XOR source.

When the high-pass filter is disabled, the high-pass flip-flop is forced to a 1, but the XOR still takes place. This causes the signal from channels 1 and 2 to be inverted. Normally this isn't noticeable, but it can show up when two channels play synchronized sound. If channels 1 and 2 are set to the same frequency and to pure tone mode, they will add, but if the same is done with channels 1 and 3, they will cancel. This doesn't happen in volume-only mode, however, as the gates that force volume-only mode are after the high-pass circuitry and

therefore volume-only channels always add in any combination.

Resetting the timers

Writing to the STIMER register causes all of the timers to reload and sets the output flip-flops. When high-pass filters are disabled, this turns off the output of channels 1 and 2 and turns on the output of channels 3 and 4. This is useful to synchronize the sound channels.

5.4 Serial port

The serial port is used to transfer data to and from the SIO bus. This allows for communication with disk drives, printers, cassette tape recorders, and other SIO-supporting peripherals.

Shift registers

The main programmatic interfaces to the serial port are the SERIN and SEROUT registers, which hold the last byte received or the next byte to transmit on the serial bus. POKEY automatically exchanges these registers with input and output shift registers and handles start and stop bits, requiring interaction with the CPU only on a byte basis and allowing the Atari to read and write back-to-back bytes on the serial bus.

Three IRQs notify the CPU when the serial port needs attention. The serial input data ready interrupt (IRQEN/IRQST bit 5) is asserted when a byte has been assembled and transferred to SERIN; the CPU can then store this byte while a new byte is shifting in. The serial output needed interrupt (bit 4) fires when SEROUT has been transferred to the output shift register and a new byte can be queued. Finally, the serial transmission complete interrupt (bit 3) is asserted when the last byte has finished shifting out and transmission is complete.

Polled operation

It is possible to drive the serial port in polled mode by enabling serial interrupts on POKEY, disabling interrupts on the CPU, and then polling IRQST. This can be useful if the data rate is too high to use interrupts.

Direct input

Bit 4 of SKSTAT directly reads the state of the serial input port. This is used by the kernel to measure baud rate prior to reading a block from cassette tape, since the serial input shift register cannot be used until the baud rate has been set.

Clock selection

Bits 4-6 of SKCTL control the clocks used during serial port operation. These three bits affect a number of switches and gates and interact in complex manners. For instance, bit 4 generally enables asynchronous receive using timer 4, but it also sometimes changes the output clock as well. Each setting specifies a different combination of signals to use for both the input and output clocks, as well as whether to configure the bidirectional clock line as an input or an output. Here are all of the modes:⁸

⁸ [ATA82] II.27 has the official mode chart; see also unnumbered page with serial/audio diagram for exact switch and gate layout.

| Setting | Input clock | Output clock | Bidirectional clock |
|---------|------------------------|-------------------|---------------------|
| 000 | External clock | External clock | Input |
| 001 | Channel 3+4 (async) | External clock | Input |
| 010 | Channel 4 | Channel 4 | Output channel 4 |
| 011 | Channel 3+4 (async) | Channel 4 (async) | Input |
| 100 | External clock | Channel 4 | Input |
| 101 | Channel 3+4 (async) | Channel 4 (async) | Input |
| 110 | Channel 4 ⁹ | Channel 2 | Output channel 4 |
| 111 | Channel 3+4 (async) | Channel 2 | Input |

Table 1: Serial port timing modes

The modes for standard half-duplex SIO operation are 001 for reception and 010 for transmission. The external clock is not normally used; for instance, the 810 disk drive ignores the clock lines and uses timing loops for both transmission and reception.

Serial port clocks are produced by divide-by-two flip flops driven off of the counter outputs. They are not affected by any of the audio control bits in the AUDC1-4 registers. However, the clock select and linking bits in AUDCTL – bit 0 and bits 3-6 – do affect serial port operation since they affect the countdown timers themselves.

When using timer channels to clock the serial port, the timer frequency should be set to twice the baud rate.¹⁰ Channels 3+4 should also be linked together and driven by the 1.79MHz clock for highest precision. For cassette operation at 600 baud, the divisor setting is \$05CC; for disk operation at 19200 baud, it is \$0028. Remember that there is a six cycle delay in reloading a 16-bit, 1.79MHz timer. Due to imprecision in the timer divisor at high frequencies, the actual transmission rate for the SIO bus is 19040 baud.

Timer usage during serial port operation

The serial port and audio circuitry both share the countdown timers and thus timers used for controlling the serial port are not available for audio generation. Usually channels 3 and 4 are used for clock generation; when using two-tone mode for recording to cassette, channels 1 and 2 are also occupied for FSK output.

Note that while the serial port uses the output of the counters, the audio circuitry is still active. This means that the occupied channels should normally be silenced by setting their volume to zero and the corresponding interrupt enables in IRQEN should also be disabled. However, the audio channels can be enabled for effect. The SIO library in the kernel ROM normally enables audio from channel 4 during transfers, producing the characteristic beep-beep-beep of Atari disk loads.

The asynchronous receive mode tone

Asynchronous receive mode can be enabled by setting bit 4 of SKCTL (\$D20F). Enabling asynchronous mode causes timer counters 3 and 4 to be reset either whenever the serial logic is waiting for a start bit or when a zero is received, resynchronizing the receive clock to the incoming bit stream. With the standard SIO receive routine, this disruption introduces an additional audible tone into the channel 4 output caused by the output flipping every byte. At 19200 baud, this produces a 960Hz tone during the read of each disk sector.

⁹ [ATA82] II.27 and [AHS03] p.21 appear to have the same error of showing channel 2 as the input clock for the 110 setting. This is not possible, as only channel 4 or the bidirectional clock line can be routed to the serial input shift register. The description text correctly indicates channel 4.

¹⁰ [ATA82] II.25. The output clock toggles level each time the timer expires, so the frequency of the clock is half the frequency of the timer.

A side effect of this behavior is that channels 3 and 4 can be locked in reset state if asynchronous receive mode is enabled. Therefore, bit 4 of SKCTL should be cleared before attempting to use those channels for audio.¹¹

Two-tone mode

Two-tone mode is enabled by setting bit 3 of SKCTL and replaces the normal 1 and 0 bits output to the SIO bus with the outputs of timers 1 and 2, respectively. This is used when writing data to tape, where the timers are programmed in 64KHz mode with divisors \$05 (5327Hz) and \$07 (3995Hz) to do FSK encoding. The timer output is tapped prior to the output circuitry and so the serial output is unaffected by either AUDC1 or AUDC2.

The switching between timer 1 and 2 based on serial data is done only in the serial logic and is therefore inaudible; both audio channels will play during transmission if their control registers are set appropriately. There is still an audible effect from two-tone mode, however, due to resynchronization between the timers: whenever the timer selected by the current serial data bit expires, the other timer is reset. This avoids glitching in the output when switching between 0 and 1 bits by keeping phase continuity between the timers. In the audio output, however, it has the effect of either lengthening the period of the inactive timer or silencing it, depending on whichever timer has the longer period.

The force break bit (SKCTL bit 7) can be used to enforce a known 0 output so that timer 2 is always used to reset timer 1.

5.5 Clock generation

There are three clocks that can be used to drive the counters:

- Channels 1 and 3 can use 1.78979MHz (NTSC) if bits 6 and 5 of AUDCTL (\$D208) are set, respectively.
- Otherwise, channels use a 63.9210KHz clock by default. This is exactly $1/28^{\text{th}}$ of the main clock.
- If bit 0 of AUDCTL (\$D208) is set, then channels use a 15.6999KHz clock instead. This is exactly $1/114^{\text{th}}$ of the main clock.

Both the 64KHz and 15KHz clocks are generated by polynomial counters internal to POKEY, driven off the main clock, and have no guaranteed phase relation to other clocks in the system. In particular, the 15.6999KHz clock signal is labeled as HSYNC in the schematic, presumably because it counts at the same rate as ANTIC's scan lines, but there is no connection to synchronize the two. The phase relationship between the 15KHz clock and horizontal scan timing is determined by when initialization mode is ended.

5.6 Pseudo-random number generators

POKEY contains three pseudo-random number generators, all composed of linear feedback shift registers (LFSRs, or polynomial counters) that run at 1.79MHz. These are used both for generating audio noise as well as random numbers for the CPU.

The 4- and 5-bit generators within POKEY are linear feedback shift registers with the polynomials $1+x^3+x^4$ and $1+x^3+x^5$, respectively. They are only used for noise output and are not accessible to the CPU.

POKEY also has a third shift register which is either 9 or 17 bits long, depending on bit 7 of AUDCTL. When in 9-bit (short) mode, the polynomial is $1+x^4+x^9$; when in 17-bit (long) mode, an additional eight bits are added to the shift register and the polynomial is $1+x^{12}+x^{17}$. Eight bits of the shift register are visible to the CPU via RANDOM (\$D20A); this is most commonly used for random numbers, but it can also be used to test cycle counting hypotheses. The CPU sees the top bits of the shift register such that bit 7 is closer to where bits are being

¹¹ [ATA82] II.26 states a slightly different rule, that the start bit resets channels 3+4. This must be interpreted as waiting for the start bit and not the actual reception of the start bit in order to explain why those channels become silent when asynchronous mode is enabled even when no serial data is being received.

shifted in and bit 0 is where bits are being shifted out.

If the main LFSR is in 9-bit mode and samples are taken from RANDOM (\$D20A) every scan line by STA WSYNC + LDA RANDOM, part of the sequence is as follows: 00 DF EE 16 B9.

Initialization behavior

The polynomial counters must be reset on startup in case they power up in a lock-up state, of which there is always exactly one state: either all 1s or 0s, depending on the exact formulation. Initialization mode forces bits into the register until it is reset to the opposite of the lock-up state so that it is guaranteed to count normally when the initialization state ends. Initialization mode need not be asserted for a long period of time for the polynomial counters to work, as a single bit of the right polarity is enough to prevent lock-up.

Once the 17-bit polynomial counter is fully initialized, RANDOM reads a constant \$FF until initialization mode ends.

5.7 Interrupts

Interrupt enable/status

The IRQEN register selectively enables or disables interrupts; a 1 bit enables an interrupt. When an interrupt is enabled and becomes active, the corresponding bit in IRQST is set to a 1 and the IRQ line to the 6502 CPU is asserted. POKEY will keep the IRQ line asserted until all pending interrupts are cleared by resetting the corresponding IRQEN bit; this ensures that the CPU will continue to execute its IRQ routine until all interrupts are serviced, even if an NMI intervenes temporarily.

Note that the serial transmission complete interrupt (bit 3) is special – it is not latched, so it is simply active whenever the serial output shift register is idle. The interrupt status bit and corresponding interrupt will be set in that case even if bit 3 of IRQEN is cleared. This can be useful to assert an IRQ on the CPU on demand.

5.8 Keyboard scan

The keyboard scan is triggered by the 15KHz clock. This means that keyboard IRQs occur relative to when the 15KHz clock is initialized. This typically means that the keyboard IRQ never hits the magic cycle on a scan line that can block NMIs, but just about every key can hit that cycle if POKEY is initialized at just the wrong offset. This happens if initialization mode is cleared at around cycle 32 on a scan line. The timing will vary somewhat due to variance in when the 6502 is able to acknowledge the interrupt.

5.9 Examples

Ray of Hope, Numen

Both of these demos use channels 3+4 in 16-bit mode at 1.79MHz with the 4-bit polynomial noise generator selected. The channels are set to a high frequency and the demos rely on the pattern of the noise generator to alias the frequency down to a lower range. The cycle period is therefore critical for the high notes to sound correctly instead of squeaking.

SpartaDOS X

SDX uses its own SIO routines for disk access that use polling rather than interrupts, by disabling interrupts on the CPU and waiting for bits in IRQST to change state.

5.10 Further reading

Read the Hardware Manual [ATA82] or the POKEY datasheet [AHS03] for theory and register-level specifications for POKEY. The Hardware Manual is especially useful here as it has detailed descriptions of the serial port and audio paths that are undocumented elsewhere.

6 CTIA/GTIA

6.1 Player/missile graphics

GTIA supports display of eight sprites on top of the playfield. These sprites can have distinct colors and can be moved horizontally much more quickly than the playfield for fast action. Four of the sprites are 8-bit wide players and four are two-bit wide missiles. All sprites are the height of the screen and can be as tall as desired. It is also possible to reposition sprites horizontally in the middle of the screen in order to increase the number of visible objects on screen.

Player/missile graphics DMA

The default method for GTIA to receive player/missile graphics data is for ANTIC DMA to read it on a scan line basis, thus relieving the CPU of the burden of spoon-feeding graphics data. In order for this to happen, either bits 2 or 3 of DMACTL in ANTIC must be set to enable DMA, and the corresponding bits 0 and 1 of GRAC TL must be set in GTIA to receive data.

Graphic data latches

The CPU can also load directly into the graphics data latches for players and missiles by writing to GRAFP0-3 and GRAFM directly. This allows the CPU to directly control P/M graphics data when ANTIC DMA is inconvenient. It also allows vertical bar patterns to be displayed without requiring data in memory, since the graphics latches can be loaded once and GTIA will reuse the same pattern for each scan line.

Vertical delay

Vertical delay is used to move a two-line resolution sprite with scan line resolution. Unlike the Atari 2600's TIA, the GTIA does not have a true vertical delay function with a delayed graphics latch. Instead, the "vertical delay" function works by masking DMA fetches. Setting the bit for a sprite in the VDELAY register causes GTIA to load DMA data for that sprite only on odd scan lines. In two-line resolution mode, when ANTIC repeats the same data on pairs of scan lines, this effectively moves the sprite image down by one scan line. In one-line resolution mode, this effectively reduces the sprite to two-line resolution.

VDELAY has no effect on writes from the CPU to GRAFP0-3 or GRAFM.

Player/missile positioning

The eight P/M objects are positioned via registers HPOSP0-HPOSP3 [D400-D403] and HPOSM0-HPOSM3 [D404-D407]. Position registers have color clock resolution. A player or missile begins shifting its output to the video display when the horizontal position counter matches the position register; this happens even if the object is positioned in the horizontal blank region ($\text{pos} < \$22$), as long as part of it is in the visible region.

The center of the playfield is at \$80. This means that the narrow playfield spans \$40-\$BF, the normal playfield \$30-\$CF, and the wide playfield \$22-\$DD (visible portion of \$20-\$DF).

Mid scan line positioning

Although the timing is very tricky, players and missiles can be repositioned in the middle of a scan line. Repositioning a sprite sufficiently later in the scan line causes it to re-trigger again, thus displaying the sprite multiple times. A sprite only begins displaying if it is repositioned beyond the current horizontal position counter, thus allowing its comparator to reload and re-trigger the shift register. If it is positioned too far left, it won't draw even if its right bound is greater than the current position. If the new position partially overlaps the old, the first image is truncated when the shift register is reloaded and restarted at the left side of the second image.

Size control

Each of the players and missiles can be set to one of three widths, with each bit displaying as one color clock (single width), two color clocks (double width), or four color clocks (quadruple width). Player widths are set by SIZEP0-SIZEP3; missile widths are set by SIZEM. Sprites are always positioned from their left edge, so increasing a sprite's width causes it to expand to the right.

Player/missile colors

Four color registers are reserved for player/missile graphics, COLPM0-3. Each player shares its color with the missile of the same number.

6.2 Collision detection

GTIA has 60 collision bits to indicate when players, missiles, and the playfield collide. This permits fast collision detection at pixel-exact level without the need for the CPU to do expensive bounding box or image comparison checks.

Collision detection mechanism

A collision is flagged between two objects when both objects are active at the same time during display. This means that a collision is not detected until the display logic actually processes the collision location on-screen, and the CPU must wait until the end of a frame in order for collisions to be reliably detected.

Color registers do not play a part in collision detection – the collision logic can distinguish between two objects of the same color. This is sometimes used to establish hidden collision objects for gameplay purposes, such as an invisible wall or a trigger. The collision logic can also see collisions between two objects even if a third object is displayed on top.

Playfield collisions

For collision detection purposes, the non-background playfield colors are each separate entities that can register collisions with players and missiles. 32 collision bits in eight registers, P0PF-P3PF and M0PF-M3PF, are devoted to registering P/M collisions against PF0-PF3. No collisions are detected against the background.

In high resolution mode (ANTIC modes 2, 3, and F), the areas corresponding to a 1 bit in the graphics data are considered to be PF2 for collision purposes. Each pair of high-resolution pixels is combined and a collision is detected if either pixel is set where a sprite is present. No collisions are registered against areas with a 0 bit even though those are displayed as non-background color.

No playfield collisions are detected in GTIA modes 9 and 11. In GTIA mode 10, a playfield collision will register whenever pixels using PF0-PF3 codes are present. No P/M collisions are reported for playfield pixels that use P/M color codes in a GTIA mode 10 screen.

Player/missile collisions

Twelve collision bits report collisions between players. A collision between player X and player Y sets two bits, one for player X in the PyPL register and another for player Y in the PxPL register. A player never registers a collision with itself: the bit for collision between a player and itself is always 0.

Sixteen collision bits in registers M0PL-M3PL report collisions between players and missiles. Each register indicates collisions between all four players against each missile.

There is no support for collision detection between missiles.

Horizontal and vertical blank

P/M collisions are only registered during the visible portions of the screen refresh and are ignored during horizontal and vertical blank. This means that only the portions of objects at horizontal positions 34-221 (\$22-\$DD) and in scan lines 8-247 (\$08-\$F7) can trigger collisions.

An object that is so far left or right that it is in partially in horizontal blank can still register collisions in the part that is in the visible region.

Resetting collision latches

The collision detection bits are latches and will stay set once a collision has been detected. Writing to HITCLR resets all collision latches to zero.

6.3 Priority control

Playfield/object priority

The GTIA uses a priority scheme to determine which objects to display when multiple objects overlap. Bits 0-3 of PRIOR control the relative priority between player/missiles and the playfields. The four official modes are as follows¹²:

| PRIOR[0:3] | 1000 | 0100 | 0010 | 0001 |
|------------|------|------|------|------|
| Top | PF0 | PF0 | P0 | P0 |
| | PF1 | PF1 | P1 | P1 |
| | P0 | PF2 | PF0 | P2 |
| | P1 | PF3 | PF1 | P3 |
| | P2 | P0 | PF2 | PF0 |
| | P3 | P1 | PF3 | PF1 |
| | PF2 | P2 | P2 | PF2 |
| | PF3 | P3 | P3 | PF3 |
| Bottom | BAK | BAK | BAK | BAK |

Note that the official hardware manual lists the fifth player (P5) as having the same priority as PF3. This is only partially true, as P5 actually assumes the priority of the highest priority playfield; more on this later.

As the hardware manual notes, setting more than one bit causes black to be output whenever the results of the priority orders corresponding to each set bit are in conflict. This is true for all 11 combinations that have more than one bit set. The case of all bits cleared (0000), however, has different behavior. The exact logic used by GTIA for resolving playfield and player/missile priorities is as follows:

```

PRI01 = PRI0 + PRI1
PRI12 = PRI1 + PRI2
PRI23 = PRI2 + PRI3
PRI03 = PRI0 + PRI3
SP0 = P0 * /(PF01*PRI23) * /(PRI2*PF23)
SP1 = P1 * /(PF01*PRI23) * /(PRI2*PF23) * (/P0 + MULTI)

```

¹² Hardware III.8

```

SP2 = P2 * /P01 * /(PF23*PRI12) * /(PF01*/PRI0)
SP3 = P3 * /P01 * /(PF23*PRI12) * /(PF01*/PRI0) * (/P2 + MULTI)
SF0 = PF0 * /(P23*PRI0) * /(P01*PRI01) * /SF3
SF1 = PF1 * /(P23*PRI0) * /(P01*PRI01) * /SF3
SF2 = PF2 * /(P23*PRI03) * /(P01*/PRI2) * /SF3
SF3 = PF3 * /(P23*PRI03) * /(P01*/PRI2)
SB  = /P01 * /P23 * /PF01 * /PF23

```

In this form, the priority bits enable specific signals that cause elements to suppress lower priority elements. Conflicts cause elements to cross-disable each other, resulting in black since the background also drops out. If no priority bits are set, however, then most of the cross-disable signals are shut off, resulting in the following set of reduced logic equations:

```

SP0 = P0
SP1 = P1 * (/P0 + MULTI)
SP2 = P2 * /P01 * /PF01
SP3 = P3 * /P01 * /PF01 * (/P2 + MULTI)
SF0 = PF0 * /SF3
SF1 = PF1 * /SF3
SF2 = PF2 * /P01
SF3 = PF3 * /P01

```

The effect is that playfields 0 and 1 can mix with players 0 and 1, and playfields 2 and 3 can mix with players 2 and 3. The mix result is the bitwise OR of the bit patterns of the involved color registers. P0/P1/PF0/PF1 have priority over P2/P3/PF2/PF3.

Fifth player enable

PRIOR bit 4 changes the color of all four missiles to that of PF3, thus allowing them to be used as a fifth player. No other change to the missiles occurs – in order to be used as a player they must be moved together manually. This means, however, that it is possible to take advantage of just the color change and still position the missiles in different places on screen.

For the purposes of priority versus players, the fifth player assumes the priority of playfield 3. It always wins against all other playfields. This leads to a contradiction in the priority mode set by PRIOR[0:3] = 1000, where the playfields are split by players in priority order. In this configuration, PF0-PF1 should cover P0-P3, which should in turn cover PF2-PF3. However, because PF3 actually overrides PF0-PF2 in order to accommodate the fifth player, this leads to the odd result that when all of the following are active:

- Either PF0 or PF1
- At least one of P0-P3
- The fifth player

...PF3 actually shows up from the fifth player in this case, because PF0/PF1 overrides the players, and then PF3 overrides PF0/PF1. However, if PF0/PF1 is taken away, then P0-P3 show up instead.

Enabling the fifth player does not affect collisions in any way. Even though it changes all missiles to use the PF3 color, each individual missile still registers collisions against playfields and players as usual, and no extra PF3 collisions result.

The fifth player has odd interactions with the 16 luma and 16 color modes. The logic that prevents the playfield values from being impressed onto the players only checks the inputs that contribute to player colors. The fifth player bypasses this such that when it is active in these modes, the result is the PF3 color impressed with the luminance or color specified by the playfield.

Multiple color player enable

By setting PRIOR bit 5, it is possible to blend players together in order to produce additional colors. The pairs that blend are P0+P1, P2+P3, M0+M1, and M2+M3. This works simply by disabling the priority logic between these pairs, thus allowing both colors to contribute to the output. The resultant color is the logical OR of the color registers involved.

Multiple color mode has no effect on collision detection.

6.4 High resolution mode (ANTIC modes 2, 3, and F)

At the beginning of horizontal blank, ANTIC signals to the GTIA whether high resolution mode is enabled. This mode is enabled for ANTIC modes 2, 3 and F and specifies whether the low two bits of playfield data for each color clock is to be interpreted as individual bits for high resolution mode. This produces pixels at each half color clock, or 320 pixels across for normal playfield width. However, as much of the logic in GTIA operates at color clock rate, this necessitates some logic bypassing and thus some unusual behavior.

When high resolution mode is active, the priority logic always sees PF2, and that is the color that is used unless that playfield is overlapped by players. The high resolution data bypasses the priority logic and conditionally impresses only the luminance from PF1 onto the output. This takes place regardless of whatever color register is used, so the change in luminance occurs on top of anything, including players, missiles, and the fifth player. The collision logic, however, sees a modified PF2C output that is the OR of the two pixels in each color clock, thus registering collisions against PF2 as expected.

Pseudo ANTIC mode E

High resolution mode is forced off whenever any of the GTIA special modes are active, thus preventing the PF1 luminance substitution or PF2C collision from interfering. This leads to a quirk of the GTIA whenever PRIOR[7:6] are set in the middle of a scanline. The high resolution flip-flop can only be set at horizontal blank, but it resets any time PRIOR[7:6] is activated and stays off for the rest of the scanline even if those bits are reset to 00. When this happens, ANTIC continues to encode data in high resolution mode while GTIA starts interpreting it as low-resolution data. Due to the differences in ANx bus encoding, this causes ANTIC mode F to revert to a pseudo mode E, where the bit pairs 00-11 encode PF0-PF3 instead of BAK + PF0-PF2.

Artifacting

In high-resolution mode, the pixel dot clock is high enough and just the right rate that an alternating stream of 0 and 1 bits can trick an NTSC receiver into interpreting the alternating bits as color. This is known as artifacting, and is the same trick used by the Apple II to create color. Unlike the Apple II, however, the Atari lacks the ability to do a 90° phase shift and thus only two phases are available. When the background color is black and the foreground is white, this produces green and purple colors with the GTIA.

Artifacting doesn't work with PAL or SECAM because the pixel dot clock doesn't match the color subcarrier frequency and thus it is not possible to produce a consistent color.

6.5 GTIA special modes

Setting the top two bits of PRIOR to something other than 00 enables one of the three special GTIA modes. These three modes have several features in common:

- Each pixel is elongated to occupy two color clocks, giving a resolution across of 80 pixels at normal playfield width.

- The GTIA modes only work properly with the hi-res ANTIC modes 2, 3, and F.
- They allow access to more simultaneous colors per scanline than any other documented modes.

Normally ANTIC sends either one pixel per color clock with five different values (low resolution mode), or two pixels per color clock in monochrome (high resolution mode). When the special modes are active, however, GTIA pairs inputs on alternating clocks for each pixel. One side effect of this is that the GTIA modes can only be scrolled by two color clocks at a time. Attempting to scroll by one color clock causes garbled output as GTIA pairs the wrong sets of bits for each pixel without actually shifting the pixel boundaries.

This pairing also explains why only ANTIC modes 2, 3, and F work. In order to form a four-bit pixel, GTIA extracts the two lower bits from each three-bit value sent by ANTIC per color clock. In the high resolution modes 2, 3, and F, these values are encoded as follows:

- 00 → 100
- 01 → 101
- 10 → 110
- 11 → 111

Most of the low resolution ANTIC modes, however, can only send BAK + PF0-PF2, encoded as follows:

- 00 → 000
- 01 → 100
- 10 → 101
- 11 → 110

This prevents access to any value of the form 11xx or xx11 and thus only gives nine of the possible sixteen pixel values. A few of the low-resolution character modes can output PF3 with the right character index (modes 4-7) and thus can produce the missing 11 output, but not in a way that is generally useful here.

16 luminance / 1 color mode (PRIOR[7:6] = 01)

Setting PRIOR[7:6] = 01 produces a playfield with a single color, but using sixteen luminance values. As this occurs by bypassing the color registers, this is the only mode in which the GTIA can produce 256 distinct color values instead of the usual 128. The color of the playfield comes from the background color register.

Player/missile graphics are largely unaffected by this mode. No playfield collisions register at all, and P/M graphics maintain their usual colors when displayed on top of the playfield. The one exception is that if the fifth player is enabled (PRIOR[4] = 1), it will assume the luminance of the playfield whenever it overlaps the playfield.

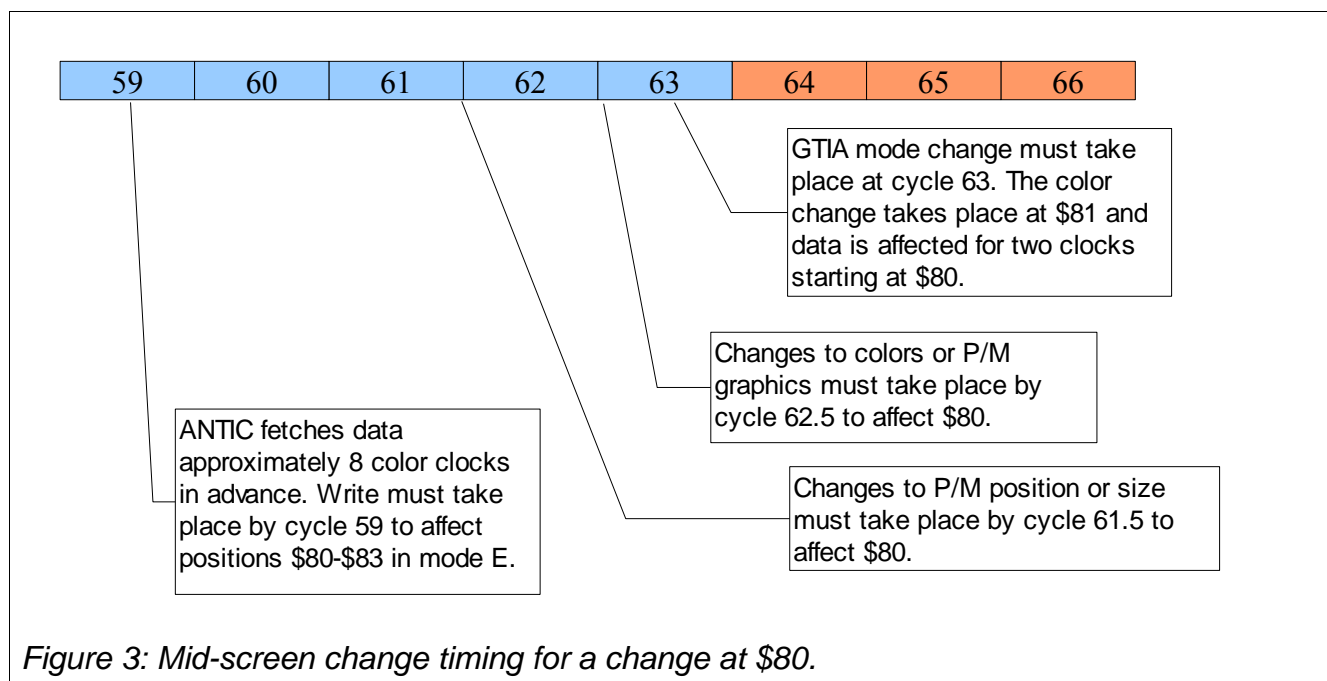
Since the playfield in this mode is activated by the absence of players, it is always on the bottom for priority purposes.

16 color / 1 luminance mode (PRIOR[7:6] = 11)

With PRIOR[7:6] = 11, the playfield is instead a single luminance, but with any of all 16 colors specified by the playfield data. The luminance comes from the background color register, with the exception of pixel 0000, which is always forced to black.

Player/missile graphics are largely unaffected by this mode. No playfield collisions register at all, and P/M graphics maintain their usual colors when displayed on top of the playfield. The one exception is that if the fifth player is enabled (PRIOR[4] = 1), it will assume the color of the playfield whenever it overlaps the playfield.

Since the playfield in this mode is activated by the absence of players, it is always on the bottom for priority purposes.



9 color mode (PRIOR[7:6] = 10)

The nine color mode, activated by PRIOR[7:6] = 10, is more unusual than the other two special modes. All of the colors come from the color registers, giving more color flexibility, and causing more interaction with the priority and collision logic.

The four bit pixel values activate color registers as follows:

- 0000-0011: P0-P3
- x100-x111: PF0-PF3
- 10xx: Background

For priority purposes, the pixel values which correspond to player colors act as though that player/missile were active and are thus modified by the priority settings in PRIOR[0:3]. They do not, however, activate player collisions. The nine color mode, however, is able to activate playfield collisions via the PF0-PF3 codes.

The nine color mode is delayed by one color clock (one half pixel) and thus appears shifted slightly right relative to all other modes.

6.6 Cycle timing

The following sections all assume that a write has taken place on cycle 63 of a scan line. In a normal width mode E line, this would be immediately before ANTIC reads data for positions \$88-\$8B.

Color register changes

A write to a color register takes place nine color clocks past when ANTIC receives graphics data. For a write on cycle 63, that means that the color change appears at position \$81.

P/M graphics changes

A write to a player/missile graphics register only takes effect when the sprite retriggers and its shift register is reloaded. For a write on cycle 63, the sprite must be at position \$81 or greater.

P/M position/size changes

A write to a player/missile position or size register must take place eleven color clocks in advance to take effect. This means that for a write on cycle 63, the new position must be at least \$83 for the shift register to reload and the player/missile to retrigger. Changes to the size register will take effect immediately, with the remaining bits in the shift register shifting out at the new width.

GTIA mode changes

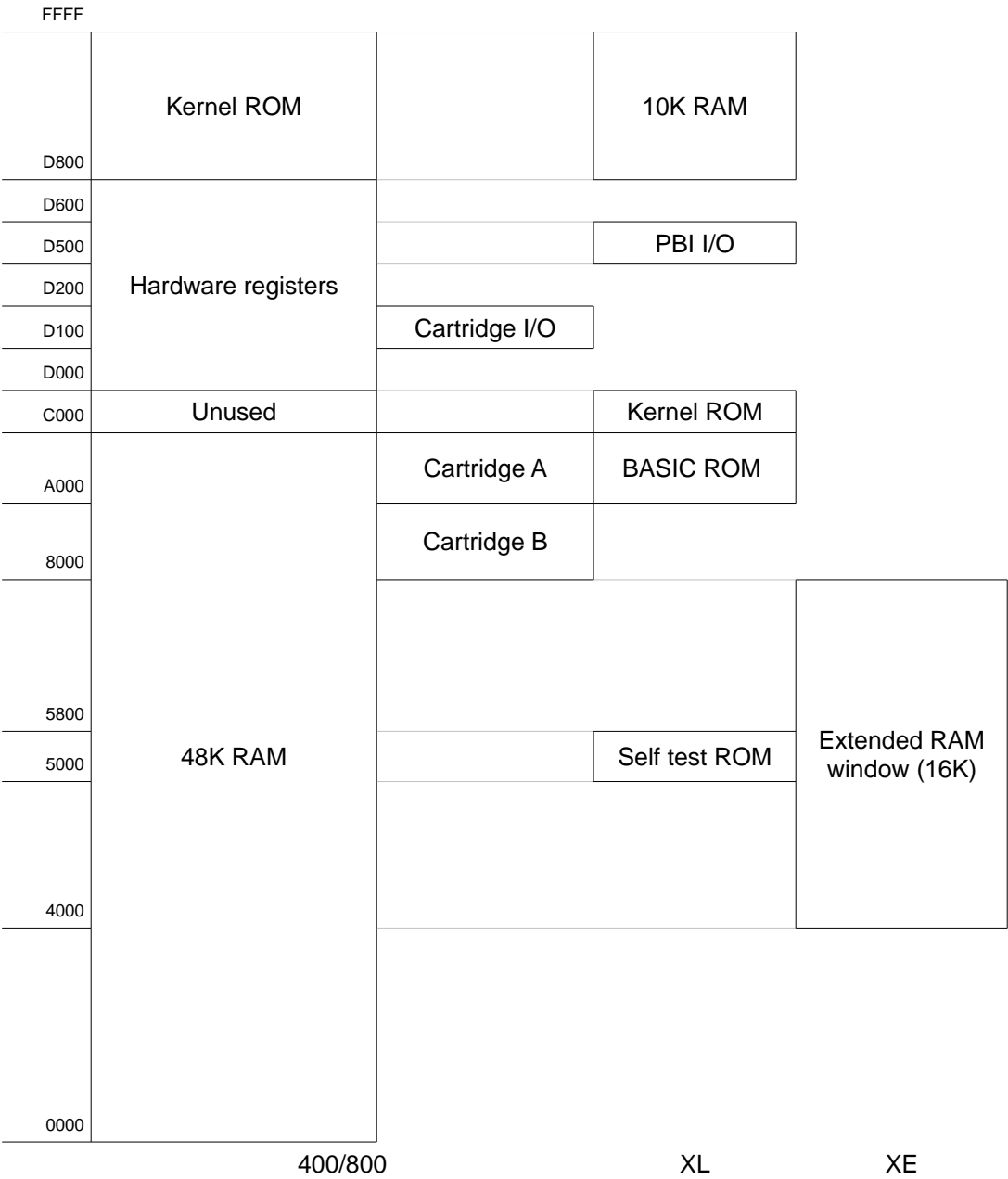
A change to bits 6-7 of PRIOR takes place between 0-2 color clocks after the write. For a write on cycle 63, the change takes place at positions \$80-\$81.

6.7 Further reading

The main source for functionality and register level descriptions for the GTIA is the Hardware Manual [ATA82] as usual, but it only covers CTIA level of functionality. Read the GTIA datasheet [AHS99a] for additional details on the GTIA modes and on communication between ANTIC and GTIA.

7 Reference

7.1 Memory map

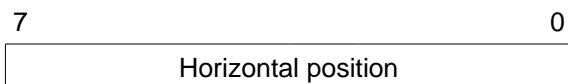


7.2 Register list

[HPOSP0-3 \[D000-D003, W\]](#)
[M0PF-M3PF \[D000-D003, R\]](#)
[HPOSM0-3 \[D004-D007, W\]](#)
[P0PF-P3PF \[D004-D007, R\]](#)
[SIZEP0-SIZEP3 \[D008-D00B, W\]](#)
[M0PL-M3PL \[D008-D00B, R\]](#)
[SIZEM \[D00C, W\]](#)
[P0PL-P3PL \[D00C-D00F, R\]](#)
[GRAFP0-3 \[D00D-D010, W\]](#)
[TRIG0-3 \[D010-D013, R\]](#)
[GRAFM \[D011, W\]](#)
[COLPM0-3 \[D012-D015, W\]](#)
[PAL \[D014, R\]](#)
[COLPF0-3 \[D016-D019, W\]](#)
[COLBK \[D01A, W\]](#)
[PRIOR \[D01B, W\]](#)
[VDELAY \[D01C, W\]](#)
[GRCTL \[D01D, W\]](#)
[HITCLR \[D01E, W\]](#)
[CONSOL \[D01F, R/W\]](#)
[AUDF1-4 \[D200/2/4/6, W\]](#)
[POT0-7 \[D200-D207, R\]](#)
[AUDC1-4 \[D201/3/5/7, W\]](#)
[AUDCTL \[D208, W\]](#)
[ALLPOT \[D208, R\]](#)
[STIMER \[D209, W\]](#)
[KBCODE \[D209, R\]](#)
[SKRES \[D20A, W\]](#)
[RANDOM \[D20A, R\]](#)
[POTGO \[D20B, W\]](#)
[SEROUT \[D20D, W\]](#)
[SERIN \[D20D, R\]](#)
[IRQEN \[D20E, W\]](#)
[IRQST \[D20E, R\]](#)
[SKCTL \[D20F, W\]](#)
[PORTB \[D301, R/W\]](#)
[DMACTL \[D400, W\]](#)
[CHACTL \[D401, W\]](#)
[DLISTL/DLISTH \[D402-3, W\]](#)
[VSCROL \[D405, W\]](#)
[PMBASE \[D407, W\]](#)
[CHBASE \[D409, W\]](#)
[WSYNC \[D40A, W\]](#)
[VCOUNT \[D40B, R\]](#)
[NMIEN \[D40E, W\]](#)
[NMIST \[D40F, R\]](#)

| Unit | Address | Description |
|------|--|--|
| GTIA | HPOSP0, HPOSP1, HPOSP2, HPOSP3 \$D000-\$D003 | Player 0-3 horizontal position (Write Only) |

Register layout



Description

HPOSP0-HPOSP3 control the position of the left edge of each of the four players, in color clocks. More precisely, they set the trigger point at which the shift register is loaded and begins shifting player graphics data through the collision and priority logic to the video output.

A position of \$80 corresponds to the center of the playfield. The narrow playfield runs from \$40-\$BF, the normal playfield from \$30-\$CF, and the wide playfield from \$22-\$DD.

| Unit | Address | Description |
|------|---|---|
| GTIA | M0PF, M1PF, M2PF, M3PF \$D000-\$D003 | Missile-to-playfield collision registers (Read Only) |

Register layout

| | | | | | | | |
|---|---|---|---|----|----|----|----|
| 7 | | | | | | | 0 |
| 0 | 0 | 0 | 0 | P3 | P2 | P1 | P0 |

D3:D0 Playfield 0-3 collision bits

| | |
|---|-----------------------|
| 0 | No collision detected |
| 1 | Collision detected |

Description

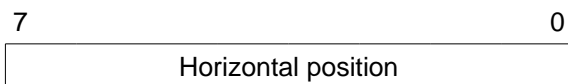
A bit is set in the M0PF, M1PF, M2PF, and M3PF registers whenever missiles 0-3 overlap a playfield in the visible region, but bit 0 being set for a collision with playfield 0. Overlaps in the horizontal or vertical blank region are not detected. Collisions are latched and stay flagged until HITCLR is written.

No playfield collisions are detected in GTIA modes 9 or 11. Playfield collisions are triggered normally for GTIA mode 10.

In high-resolution modes (ANTIC modes 2, 3, and F), the monochrome playfield is considered to be PF2. Either of the two pixels being set in the pair displayed during a color clock will signal a PF2 collision on that clock.

| Unit | Address | Description |
|------|--|---|
| GTIA | HPOSM0, HPOSM1, HPOSM2, HPOSM3 \$D004-\$D007 | Missile 0-3 horizontal position (Write Only) |

Register layout



Description

HPOSM0-HPOSM3 control the position of the left edge of each of the four missiles, in color clocks. More precisely, they set the trigger point at which the shift register is loaded and begins shifting missile graphics data through the collision and priority logic to the video output.

A position of \$80 corresponds to the center of the playfield. The narrow playfield runs from \$40-\$BF, the normal playfield from \$30-\$CF, and the wide playfield from \$22-\$DD.

| Unit | Address | Description |
|------|--|--|
| GTIA | P0PF, P1PF, P2PF, P3PF \$D004-\$D007 | Player-to-playfield collision registers (Read Only) |

Register layout

| | | | | | | | |
|---|---|---|---|-----|-----|-----|-----|
| 7 | | | | | | | 0 |
| 0 | 0 | 0 | 0 | PF3 | PF2 | PF1 | PF0 |

D3:D0 Playfield 0-3 collision bits

| | |
|---|-----------------------|
| 0 | No collision detected |
| 1 | Collision detected |

Description

A bit is set in the P0PF, P1PF, P2PF, and P3PF registers whenever players 0-3 overlap a playfield in the visible region, but bit 0 being set for a collision with playfield 0. Overlaps in the horizontal or vertical blank region are not detected. Collisions are latched and stay flagged until HITCLR is written.

No playfield collisions are detected in GTIA modes 9 or 11. Playfield collisions are triggered normally for GTIA mode 10.

In high-resolution modes (ANTIC modes 2, 3, and F), the monochrome playfield is considered to be PF2. Either of the two pixels being set in the pair displayed during a color clock will signal a PF2 collision on that clock.

| Unit | Address | Description |
|------|--|---|
| GTIA | SIZEP0, SIZEP1, SIZEP2, SIZEP3 \$D008-\$D00B | Player horizontal width control (Write Only) |

Register layout

| | |
|---------|------|
| 7 | 0 |
| Ignored | Size |

D1:D0 Player size

| | |
|----|--|
| 00 | Normal width (1 color clock per bit) |
| 01 | Double width (2 color clocks per bit) |
| 10 | Normal width (1 color clock per bit) |
| 11 | Quadruple width (4 color clocks per bit) |

Description

SIZEP0-SIZEP3 control the horizontal width of each player by specifying how many color clocks to display each bit on screen. Since the horizontal position registers control the left side of each player, increasing the width causes players to expand to the right.

A change to SIZEPx while the corresponding player is being shifted out will take place immediately.

| Unit | Address | Description |
|------|--|--|
| GTIA | M0PL, M1PL, M2PL, M3PL \$D008-\$D00B | Missile-to-player collision registers (Read Only) |

Register layout

| | | | | | | | |
|---|---|---|---|----|----|----|----|
| 7 | | | | | | | 0 |
| 0 | 0 | 0 | 0 | P3 | P2 | P1 | P0 |

D3:D0 Player 0-3 collision bits

| | |
|---|-----------------------|
| 0 | No collision detected |
| 1 | Collision detected |

Description

A bit is set in the M0PL, M1PL, M2PL, and M3PL registers whenever missiles 0-3 overlap a player in the visible region, but bit 0 being set for a collision with player 0. Overlaps in the horizontal or vertical blank region are not detected. Collisions are latched and stay flagged until HITCLR is written.

| Unit | Address | Description |
|------|-----------------|--|
| GTIA | SIZEM \$D00C | Missile horizontal width control (Write Only) |

Register layout

| | | | | |
|--------|--------|--------|--------|---|
| 7 | | | | 0 |
| Size 3 | Size 2 | Size 1 | Size 0 | |

D7:D6 Missile 3 size

D5:D4 Missile 2 size

D3:D2 Missile 1 size

D1:D0 Missile 0 size

00 Normal width (1 color clock per bit)

01 Double width (2 color clocks per bit)

10 Normal width (1 color clock per bit)

11 Quadruple width (4 color clocks per bit)

Description

SIZEM0-SIZEM3 control the horizontal width of each missile by specifying how many color clocks to display each bit on screen. Since the horizontal position registers control the left side of each missile, increasing the width causes missiles to expand to the right.

A change to SIZEM while the corresponding missile is being shifted out will take place immediately.

| Unit | Address | Description |
|------|--|---|
| GTIA | P0PL, P1PL, P2PL, P3PL \$D00C-\$D00F | Player-to-player collision registers (Read Only) |

Register layout

| | | | | | | | |
|---|---|---|---|----|----|----|----|
| 7 | | | | | | | 0 |
| 0 | 0 | 0 | 0 | P3 | P2 | P1 | P0 |

D3:D0 Player 0-3 collision bits

| | |
|---|-----------------------|
| 0 | No collision detected |
| 1 | Collision detected |

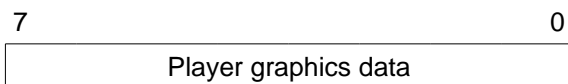
Description

A bit is set in the P0PL, P1PL, P2PL, and P3PL registers whenever two players overlap in the visible region, with bit 0 being set for a collision with player 0. Overlaps in the horizontal or vertical blank region are not detected. Collisions are latched and stay flagged until HITCLR is written.

A player never collides with itself and the corresponding collision bit is always 0.

| Unit | Address | Description |
|------|--|---|
| GTIA | GRAFP0, GRAFP1, GRAFP2, GRAFP3 \$D00D-\$D010 | Player graphics registers (Write Only) |

Register layout



Description

GRAFP0-GRAFP3 hold the graphics data that is loaded into the shift register when each player is triggered by horizontal position. Normally player DMA is enabled on ANTIC when player graphics are used, which causes GRAFP0-GRAFP3 to be loaded automatically at the start of each scan line. When disabled, GTIA uses whatever data is in the internal latches. The latches can then be updated under CPU control, or simply left alone to display the same data on every scan line.

Data is displayed MSB to LSB, with the most significant bit being displayed on the left.

| Unit | Address | Description |
|------|---|----------------------------------|
| GTIA | TRIG0, TRIG1, TRIG2, TRIG3 \$D010-\$D013 | Trigger registers (Read Only) |

Register layout

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7 | | | | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | T |

| | |
|----|------------------------|
| D0 | Trigger bit (inverted) |
| 0 | Trigger active |
| 1 | Trigger not active |

Description

TRIG0-3 reflect the state of the four joystick trigger inputs.

On the XL line, only two joystick ports are present and TRIG2 always reads as 1. TRIG3 is repurposed as the cartridge detect line, reading 1 if cartridge ROM is mapped to \$A000-BFFF and 0 otherwise.

| Unit | Address | Description |
|------|-----------------|---|
| GTIA | GRAFM \$D011 | Missile graphics register (Write Only) |

Register layout

| | | | | |
|-----------|-----------|-----------|-----------|---|
| 7 | | | | 0 |
| Missile 3 | Missile 2 | Missile 1 | Missile 0 | |

Description

GRAFM holds the graphics data that is loaded into the shift register when each missile is triggered by horizontal position. Normally missile DMA is enabled on ANTIC when missile graphics are used, which causes GRAFM to be loaded automatically at the start of each scan line. When disabled, GTIA uses whatever data is in the internal latch. The latch can then be updated under CPU control, or simply left alone to display the same data on every scan line.

Data is displayed MSB to LSB, with the most significant bit being displayed on the left.

| Unit | Address | Description |
|------|--|---|
| GTIA | COLPM0, COLPM1, COLPM2, COLPM3 \$D012-\$D015 | Player/missile 0-3 color register (Write Only) |

Register layout

| | | |
|-----|-----------|------|
| 7 | | 0 |
| Hue | Luminance | Ign. |

Description

These registers control the base colors used for players 0-3.

| Unit | Address | Description |
|------|---------------|---|
| GTIA | PAL \$D014 | NTSC/PAL detect register (Read Only) |

Register layout

| | |
|-----|---|
| 7 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| PAL | |

D3:D0 NTSC/PAL detect

0000 PAL
0111 NTSC

Description

The PAL register indicates whether the GTIA is either the NTSC or PAL model.

| Unit | Address | Description |
|------|--|--|
| GTIA | COLPF0, COLPF1, COLPF2, COLPF3 \$D016-\$D019 | Playfield 0-3 color register (Write Only) |

Register layout

| | | | |
|---|-----|-----------|------|
| 7 | | | 0 |
| | Hue | Luminance | Ign. |

Description

These registers control the base colors used for playfields 0-3.

In ANTIC modes 2, 3, and F, COLPF2 controls the color of the playfield. A 1 bit in the graphics data replaces the luminance of a pixel with that from COLPF1.

| Unit | Address | Description |
|------|-----------------|---|
| GTIA | COLBK \$D01A | Background color register (Write Only) |

Register layout

| | | | |
|---|-----|-----------|------|
| 7 | | | 0 |
| | Hue | Luminance | Ign. |

Description

This register controls the color of the background, including the horizontal and vertical blank regions.

| Unit | Address | Description |
|------|-----------------|----------------------------------|
| GTIA | PRIOR \$D01B | Priority control (Write Only) |

Register layout

| | | | | |
|------|----|----|---------------|---|
| 7 | | | | 0 |
| GTIA | MC | P5 | Priority mode | |

D3:D0 Playfield / P/M priority mode

| | |
|------|---|
| 1000 | PF0 > PF1 > P0 > P1 > P2 > P3 > PF2 > PF3 > BAK |
| 0100 | PF0 > PF1 > PF2 > PF3 > P0 > P1 > P2 > P3 > BAK |
| 0010 | P0 > P1 > PF0 > PF1 > PF2 > PF3 > P2 > P3 > BAK |
| 0001 | P0 > P1 > P2 > P3 > PF0 > PF1 > PF2 > PF3 > BAK |

D4 Fifth player enable

| | |
|---|--------------------------------|
| 0 | Missiles use player 0-3 colors |
| 1 | Missiles use playfield 3 color |

D5 Multicolor player enable

| | |
|---|----------------------------|
| 0 | Normal |
| 1 | Multicolor players enabled |

D7:D6 GTIA mode enable

| | |
|----|-------------------------|
| 00 | Normal |
| 01 | 1 color / 16 luma mode |
| 10 | 9 color mode |
| 11 | 16 colors / 1 luma mode |

Description

PRIOR controls a bunch of miscellaneous options, including player/missile priority relative to playfields. All of these options have complex interactions with the rest of the video display logic. See the CTIA/GTIA chapter for details.

| Unit | Address | Description |
|------|------------------|--------------------------------|
| GTIA | VDELAY \$D01C | Vertical delay (Write Only) |

Register layout

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 7 | | | | | | | 0 |
| P3 | P2 | P1 | P0 | M3 | M2 | M1 | M0 |

D7:D0 Vertical delay

- | | |
|---|--|
| 0 | Accept DMA data every scan line |
| 1 | Accept DMA data only on odd scan lines |

Description

VDELAY is used to vertically scroll players and missiles down by one scan line in two-line resolution mode. Contrary to its name, however, it doesn't actually delay anything. What it does is control whether GTIA loads the graphics latches from the data during DMA time on even scan lines. When a bit is set in VDELAY, the corresponding sprite only loads data on odd scan lines, which effectively moves the sprite down a scan line when two-line DMA mode is enabled. In single line mode, this has the effect of halving sprite resolution.

VDELAY has no effect on direct writes to the GRAFP0-3 or GRAFM registers.

| Unit | Address | Description |
|------|-----------------|----------------------------------|
| GTIA | GRCTL \$D01D | Graphics control (Write Only) |

Register layout

| | | | | | |
|---|---------|----|---|---|---|
| 7 | | | | | 0 |
| | Ignored | LT | P | M | |

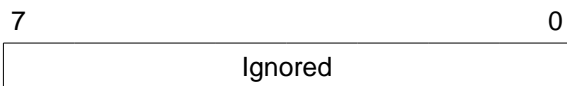
| | |
|----|------------------------------|
| D0 | Missile DMA enable |
| 0 | Disabled |
| 1 | Missile DMA enabled |
| D1 | Player DMA enable |
| 0 | Disabled |
| 1 | Player DMA enabled |
| D2 | Trigger latch enable |
| 0 | Trigger inputs are momentary |
| 1 | Trigger inputs are latched |

Description

GRCTL controls player/missile DMA on the GTIA side. DMACTL in ANTIC must be set appropriately to enable P/M data to be fetched from memory, but GRCTL in GTIA must also be set for that data to be accepted into the GRAFP0-GRAFP3 and GRAFM registers.

| Unit | Address | Description |
|------|------------------|--|
| GTIA | HITCLR \$D01E | Collision control clear strobe (Write Only) |

Register layout



Description

A write to HITCLR clears all of the collision registers.

| Unit | Address | Description |
|------|------------------|---------------------------------|
| GTIA | CONSOL \$D01F | Console control (Read/Write) |

Register layout

| | | | | | | | |
|---|---|---|---|-----|-----|-----|-----|
| 7 | | | | | | | 0 |
| 0 | 0 | 0 | 0 | SPK | OPT | SEL | STA |

D3 Loudspeaker

| | |
|---|--------|
| 0 | Source |
| 1 | Sink |

D2 OPTION key

| | |
|---|----------------------------------|
| 0 | Asserted (read) / Source (write) |
| 1 | Inactive (read) / Sink (write) |

D1 SELECT key

| | |
|---|----------------------------------|
| 0 | Asserted (read) / Source (write) |
| 1 | Inactive (read) / Sink (write) |

D0 START key

| | |
|---|----------------------------------|
| 0 | Asserted (read) / Source (write) |
| 1 | Inactive (read) / Sink (write) |

Description

CONSOL reads and writes the state of four bidirectional switch lines connected to GTIA. On the Atari, these are connected to the internal loudspeaker and the OPTION, SELECT, and START keys. Writing a 0 into a bit causes the corresponding switch line to be pulled up to +5V, and writing a 1 sinks it to ground.

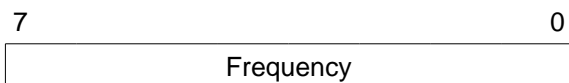
By default, the OS writes \$08 into CONSOL during vertical blank.¹³ This causes the CONSOL register to read \$07 when no keys are pressed, with bits 0-2 going low when one of the console buttons is pressed. If a 1 is written into bits 0-2, the corresponding switch is grounded and always reads as a 0.

The XL series has no internal loudspeaker and thus the speaker output is routed to the TV instead.

¹³ [ATA82] III.15

| Unit | Address | Description |
|-------|---|-----------------------------|
| POKEY | AUDF1, AUDF2, AUDF3, AUDF4 \$D200, \$D202, \$D204, (Write Only) \$D206 | Audio channel 0-3 frequency |

Register layout

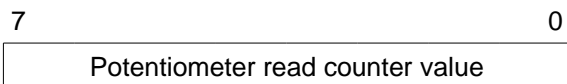


Description

AUDF1-AUDF4 control the frequency of the four audio channels.

| Unit | Address | Description |
|-------|----------------------------|---|
| POKEY | POT0-POT7 \$D200-\$D207 | Potentiometer read counter (Read Only) |

Register layout



Description

POT0-POT7 indicate the value of each of the eight potentiometer read counters. When POTGO is written, each of the counters is reset to 0 and begins counting up until either the threshold or the value 228 has been hit. The corresponding bit in ALLPOT is then set to indicate that the pot counter value is valid.

| Unit | Address | Description |
|-------|---|---------------------------|
| POKEY | AUDC1, AUDC2, AUDC3, AUDC4 \$D201, \$D203, \$D205, (Write Only) \$D207 | Audio channel 0-3 control |

Register layout

| | | | | | |
|-----|----|----|---|--------------|---|
| 7 | | | | | 0 |
| CLK | NM | NC | D | Volume level | |

D3:D0 Volume level

| | |
|------|----------------|
| 0000 | Silent |
| 0001 | Lowest volume |
| 1111 | Highest volume |

D4 Output disable

| | |
|---|------------------|
| 0 | Normal operation |
| 1 | Volume-only mode |

D5 Noise control

| | |
|---|--|
| 0 | Sample noise source |
| 1 | Output pure tone (produce square wave by toggling output on clock pulse) |

D6 Noise mode

| | |
|---|--|
| 0 | Sample 9-bit or 17-bit polynomial generator (see AUDCTL bit 7) |
| 1 | Sample 4-bit polynomial generator |

D7 Sampling clock mode

| | |
|---|--|
| 0 | Mask out clock pulses using 5-bit polynomial generator |
| 1 | Use timer output directly as clock |

Description

AUDC1-AUDC4 control the volume and timbre of the four audio channels.

See the *Audio and Serial Port Block Diagram* page of the Hardware Manual [ATA82] for a logic diagram that shows precisely how the bits in AUDCx affect the output flow.

| Unit | Address | Description |
|-------|------------------|-------------------------------|
| POKEY | AUDCTL \$D208 | Audio control (Write Only) |

Register layout

| | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|---|
| 7 | | | | | | | | 0 |
| PLY | CH1 | CH3 | L12 | L34 | HP1 | HP3 | 15K | |

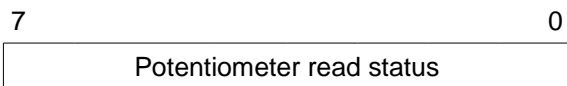
| | |
|----|---|
| D7 | Polynomial select |
| 0 | RANDOM and audio channels use 17-bit polynomial generator |
| 1 | RANDOM and audio channels use 9-bit polynomial generator |
| D6 | Channel 1 fast clock enable |
| D5 | Channel 3 fast clock enable |
| 0 | Clock channel with 15KHz or 64KHz clock |
| 1 | Clock channel with 1.79MHz clock |
| D4 | Channel 1+2 link enable |
| D3 | Channel 3+4 link enable |
| 0 | Independent 8-bit counters |
| 1 | Linked 16-bit counter (clock 2 with 1 or 4 with 3). |
| D2 | Channel 1 high pass filter enable |
| D1 | Channel 2 high pass filter enable |
| 0 | Normal operation |
| 1 | High pass enabled (filter channel 1/2 with channel 3/4) |
| D0 | Clock select |
| 0 | Use 64KHz as slow audio clock |
| 1 | Use 15KHz as slow audio clock |

Description

AUDCTL controls a number of miscellaneous sound parameters.

| Unit | Address | Description |
|-------|------------------|--|
| POKEY | ALLPOT \$D208 | Potentiometer read status (Read Only) |

Register layout



D7:D0 Pot 0-7 read status

| | |
|---|--------------------------------|
| 0 | Potentiometer read complete |
| 1 | Potentiometer still being read |

Description

ALLPOT indicates when each of the eight potentiometers have been read and the counter values are valid.

| Unit | Address | Description |
|-------|------------------|------------------------------------|
| POKEY | STIMER \$D209 | Start timer strobe (Write Only) |

Register layout

| | |
|---------|---|
| 7 | 0 |
| Ignored | |

Description

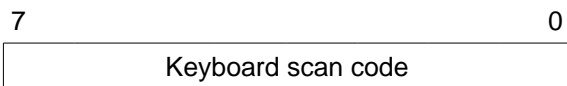
Writing to STIMER causes all timers to restart from their set period values, sets the output flip-flops for all channels to 0 (1 after inversion). When high-pass filters are disabled, this silences channels 1 and 2 and enables output for channels 3 and 4.

Errata

The POKEY datasheet [AHS03] states that STIMER forces channels 1 and 2 to logic high and channels 3 and 4 to logic low; this is backwards if logic high is the state that produces sound.

| Unit | Address | Description |
|-------|------------------|---------------------------------------|
| POKEY | KBCODE \$D209 | Keyboard code register (Read Only) |

Register layout



Description

Contains the scan code of the most recently pressed key.

| Unit | Address | Description |
|-------|-----------------|--|
| POKEY | SKRES \$D20A | Serial/keyboard reset strobe (Write Only) |

Register layout

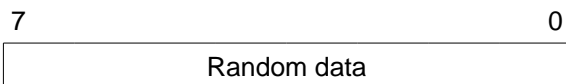
| | |
|---------|---|
| 7 | 0 |
| Ignored | |

Description

Writing to SKRES resets the serial port and keyboard status bits in SKSTAT.

| Unit | Address | Description |
|-------|------------------|--|
| POKEY | RANDOM \$D20A | Random number generator (Read Only) |

Register layout



Description

Reads the state of the top eight bits of the 17-bit polynomial noise generator. This generator counts at 1.79MHz and thus changes every cycle.

If bit 7 of AUDCTL is set, the 17-bit polynomial noise generator is shortened to 9 bits. This is reflected in the values read from RANDOM. Because the noise generator is a linear feedback shift register (LFSR) of the XOR variety, a state of all zeroes is invalid and therefore a RANDOM value of 00 is a unique LFSR state (all other values can be one of two states). From this state, the progression is as follows:

```

0: 00  43: 48  86: 94  129: ff  172: 70  215: e7  258: aa  301: a0  344: 45  387: cc  430: 3f  473: 86
1: 80  44: a4  87: 4a  130: ff  173: 38  216: 73  259: 55  302: d0  345: a2  388: 66  431: 9f  474: c3
2: 40  45: 52  88: 25  131: 7f  174: 9c  217: 39  260: aa  303: e8  346: d1  389: 33  432: 4f  475: 61
3: 20  46: a9  89: 12  132: 3f  175: ce  218: 1c  261: d5  304: 74  347: e8  390: 99  433: a7  476: b0
4: 10  47: 54  90: 09  133: 1f  176: 67  219: 0e  262: ea  305: ba  348: f4  391: 4c  434: d3  477: 58
5: 88  48: 2a  91: 04  134: 0f  177: 33  220: 07  263: f5  306: dd  349: fa  392: a6  435: 69  478: ac
6: 44  49: 15  92: 82  135: 87  178: 19  221: 03  264: fa  307: ee  350: fd  393: 53  436: b4  479: 56
7: 22  50: 8a  93: 41  136: c3  179: 0c  222: 81  265: 7d  308: f7  351: fe  394: a9  437: 5a  480: ab
8: 11  51: c5  94: 20  137: e1  180: 86  223: c0  266: be  309: fb  352: 7f  395: d4  438: ad  481: 55
9: 88  52: 62  95: 90  138: f0  181: 43  224: e0  267: 5f  310: 7d  353: bf  396: 6a  439: 56  482: 2a
10: c4  53: b1  96: c8  139: 78  182: 21  225: 70  268: af  311: 3e  354: 5f  397: 35  440: 2b  483: 95
11: 62  54: d8  97: 64  140: bc  183: 90  226: b8  269: d7  312: 1f  355: 2f  398: 9a  441: 15  484: ca
12: 31  55: 6c  98: 32  141: de  184: 48  227: dc  270: 6b  313: 8f  356: 97  399: 4d  442: 0a  485: e5
13: 98  56: 36  99: 99  142: ef  185: 24  228: ee  271: b5  314: c7  357: 4b  400: 26  443: 85  486: 72
14: 4c  57: 9b  100: cc  143: 77  186: 12  229: 77  272: 5a  315: e3  358: a5  401: 93  444: 42  487: 39
15: 26  58: cd  101: e6  144: 3b  187: 89  230: bb  273: 2d  316: f1  359: d2  402: c9  445: a1  488: 9c
16: 13  59: e6  102: 73  145: 1d  188: 44  231: 5d  274: 16  317: 78  360: 69  403: e4  446: 50  489: 4e
17: 89  60: f3  103: b9  146: 0e  189: a2  232: 2e  275: 0b  318: 3c  361: 34  404: f2  447: 28  490: 27
18: c4  61: f9  104: 5c  147: 87  190: 51  233: 97  276: 05  319: 9e  362: 1a  405: f9  448: 14  491: 13
19: e2  62: 7c  105: 2e  148: 43  191: a8  234: cb  277: 82  320: cf  363: 8d  406: fc  449: 8a  492: 09
20: 71  63: 3e  106: 17  149: a1  192: d4  235: e5  278: c1  321: 67  364: 46  407: 7e  450: 45  493: 84
21: b8  64: 9f  107: 8b  150: d0  193: ea  236: f2  279: 60  322: b3  365: a3  408: bf  451: 22  494: c2
22: 5c  65: cf  108: c5  151: 68  194: 75  237: 79  280: b0  323: 59  366: 51  409: df  452: 91  495: 61
23: ae  66: e7  109: e2  152: 34  195: ba  238: bc  281: d8  324: 2c  367: 28  410: 6f  453: c8  496: 30
24: 57  67: f3  110: f1  153: 9a  196: 5d  239: 5e  282: ec  325: 96  368: 94  411: b7  454: e4  497: 18
25: ab  68: 79  111: f8  154: cd  197: ae  240: af  283: 76  326: cb  369: ca  412: 5b  455: 72  498: 8c
26: d5  69: 3c  112: 7c  155: 66  198: d7  241: 57  284: bb  327: 65  370: 65  413: 2d  456: b9  499: 46
27: 6a  70: 1e  113: be  156: b3  199: eb  242: 2b  285: dd  328: b2  371: 32  414: 96  457: dc  500: 23
28: b5  71: 8f  114: df  157: d9  200: f5  243: 95  286: 6e  329: 59  372: 19  415: 4b  458: 6e  501: 11
29: da  72: 47  115: ef  158: 6c  201: 7a  244: 4a  287: b7  330: ac  373: 8c  416: 25  459: 37  502: 08
30: 6d  73: a3  116: f7  159: b6  202: 3d  245: a5  288: db  331: d6  374: c6  417: 92  460: 9b  503: 84
31: 36  74: d1  117: 7b  160: db  203: 9e  246: 52  289: 6d  332: eb  375: 63  418: 49  461: 4d  504: 42
32: 1b  75: 68  118: 3d  161: ed  204: 4f  247: 29  290: b6  333: 75  376: 31  419: 24  462: a6  505: 21
33: 8d  76: b4  119: 1e  162: f6  205: 27  248: 14  291: 5b  334: 3a  377: 18  420: 92  463: d3  506: 10
34: c6  77: da  120: 0f  163: 7b  206: 93  249: 0a  292: ad  335: 1d  378: 0c  421: c9  464: e9  507: 08
35: e3  78: ed  121: 07  164: bd  207: 49  250: 05  293: d6  336: 8e  379: 06  422: 64  465: f4  508: 04
36: 71  79: 76  122: 83  165: 5e  208: a4  251: 02  294: 6b  337: c7  380: 03  423: b2  466: 7a  509: 02
37: 38  80: 3b  123: c1  166: 2f  209: d2  252: 81  295: 35  338: 63  381: 01  424: d9  467: bd  510: 01
38: 1c  81: 9d  124: e0  167: 17  210: e9  253: 40  296: 1a  339: b1  382: 80  425: ec  468: de
39: 8e  82: 4e  125: f0  168: 0b  211: 74  254: a0  297: 0d  340: 58  383: c0  426: f6  469: 6f
40: 47  83: a7  126: f8  169: 85  212: 3a  255: 50  298: 06  341: 2c  384: 60  427: fb  470: 37
41: 23  84: 53  127: fc  170: c2  213: 9d  256: a8  299: 83  342: 16  385: 30  428: fd  471: 1b
42: 91  85: 29  128: fe  171: e1  214: ce  257: 54  300: 41  343: 8b  386: 98  429: 7e  472: 0d

```

| Unit | Address | Description |
|-------|-----------------|---|
| POKEY | POTGO \$D20B | Potentiometer read start strobe (Write Only) |

Register layout

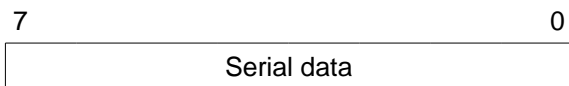
| | |
|---------|---|
| 7 | 0 |
| Ignored | |

Description

Writing to POTGO dumps the potentiometer read capacitors and resets the pot counters, restarting the pot read process. This causes all POT0-POT7 registers to reset to 0 and ALLPOT becomes \$FF until each pot is measured. In fast pot scan mode, the pots can be used as cycle timers, although the read values appear to be slightly unreliable while counting.

| Unit | Address | Description |
|-------|------------------|--|
| POKEY | SEROUT \$D20D | Serial output register (Write Only) |

Register layout

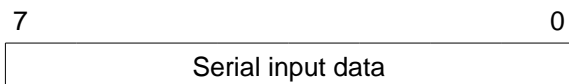


Description

SEROUT is written by the CPU to specify the data that should be copied to the serial output shift register and sent out to the SIO bus.

| Unit | Address | Description |
|-------|-----------------|--------------------------------------|
| POKEY | SERIN \$D20D | Serial input register (Read Only) |

Register layout



Description

Reads the data that was most recently shifted into POKEY from the SIO bus and clears the internal data-ready state. If two consecutive bytes are shifted in without SERIN being read in between, the second byte overwrites the first and the serial input overrun bit in SKSTAT is set.

| Unit | Address | Description |
|-------|-----------------|-------------------------------------|
| POKEY | IRQEN \$D20E | IRQ enable register (Write Only) |

Register layout

| | | | | | | | | |
|-----|-----|-----|-----|-----|----|----|----|---|
| 7 | | | | | | | | 0 |
| BRK | KBD | SIN | SOT | STR | T4 | T2 | T1 | |

| | |
|----|--|
| D7 | Break key interrupt |
| D6 | Keyboard interrupt |
| D5 | Serial input data ready interrupt |
| D4 | Serial output data needed ready interrupt |
| D3 | Serial output transmission completed interrupt |
| D2 | Timer 4 expired interrupt |
| D1 | Timer 2 expired interrupt |
| D0 | Timer 1 expired interrupt |
| 0 | Disabled, reset associated status bit |
| 1 | Enabled |

Description

IRQEN selectively enables or disables various IRQ sources within POKEY. Disabling an IRQ source via IRQEN also resets the associated status bit in IRQST and clears the interrupt if it is currently pending. The exception is the transmission complete bit in IRQST (bit 3), which is not reset by writes to IRQEN. As long as the serial output hardware is idle, setting bit 3 will immediately cause the serial output transmission interrupt to fire.

| Unit | Address | Description |
|-------|-----------------|------------------------------------|
| POKEY | IRQST \$D20E | IRQ status register (Read Only) |

Register layout

| | | | | | | | | |
|-----|-----|-----|-----|-----|----|----|----|---|
| 7 | | | | | | | | 0 |
| BRK | KBD | SIN | SOT | STR | T4 | T2 | T1 | |

| | |
|----|--|
| D7 | Break key interrupt |
| D6 | Keyboard interrupt |
| D5 | Serial input data ready interrupt |
| D4 | Serial output data needed ready interrupt |
| D2 | Timer 4 expired interrupt |
| D1 | Timer 2 expired interrupt |
| D0 | Timer 1 expired interrupt |
| | 0 Interrupt pending |
| | 1 Not active or interrupt disabled |
| D3 | Serial output transmission completed interrupt |
| | 0 Serial transmission completed |
| | 1 Serial transmission in progress |

Description

IRQST indicates when various interrupts are pending from POKEY. These interrupts remain active and trigger at the end of the next instruction if the 6502 processor status bit I is cleared unless reset via IRQEN.

Most bits in IRQST are reset and stay low when the corresponding interrupt is cleared via IRQEN. The exception is the serial output transmission bit (bit 3), which is not latched and always indicates the current state.

| Unit | Address | Description |
|-------|-----------------|---|
| POKEY | SKCTL \$D20F | Serial/keyboard control (Write Only) |

Register layout

| | | | | | | |
|----|-----------------|----|----|----|----|---|
| 7 | | | | | | 0 |
| FB | Serial clk mode | 2T | FP | KS | KD | |

D7 Force break

- 0 Serial data is output normally
- 1 Serial output line is forced to 0

D6:D5 Serial clock mode

D4 Asynchronous receive mode

- 0 Disabled
- 1 Enabled – use timer 4 as input clock and reset timers 3+4 when waiting for start bit or a zero is received

D3 Two-tone mode

- 0 Disabled – serial data is output directly on bus
- 1 Enabled – audio channels 1 and 2 output on bus for a 1 and a 0, respectively

D2 Fast pot scan

- 0 Slow pot scan: counters increment every 114 cycles
- 1 Fast pot scan: counters increment every cycle

D1 Enable keyboard scan

D0 Enable keyboard debounce

- 0 Disabled
- 1 Enabled
- 00* Special case – initialize

Description

SKCTL controls a number of miscellaneous serial port, keyboard, and pot scan functions in POKEY. See chapter 5.4, Serial port for more details.

| Unit | Address | Description |
|-------|-----------------|--|
| ANTIC | PORTB \$D301 | Port B data/direction register (Read/Write) |

Register layout

| | | | | | | | | |
|----------------|--------|---|---|--------|----|---|--------------------------------|-------------|
| 7 | | | | | | | 0 | |
| Direction bits | | | | | | | Direction (PORTBCTL bit 2 = 0) | |
| Jack 4 | | | | Jack 3 | | | 400/800 only | |
| S | Unused | | | L2 | L1 | B | K | 1200XL only |
| S | Unused | | | | | B | K | 600XL/800XL |
| S | Un. | A | C | Bank | | B | K | 130XE only |

| | | | | | | | |
|-------|--|--|--|--|--|--|--|
| D0 | Kernel ROM enable (XL/XE) | | | | | | |
| 0 | Map RAM at \$D800-FFFF | | | | | | |
| 1 | Map Kernel ROM at \$D800-FFFF | | | | | | |
| D1 | BASIC ROM enable (XL/XE) | | | | | | |
| 0 | BASIC ROM enabled at \$A000-BFFF | | | | | | |
| 1 | BASIC ROM disabled | | | | | | |
| D3:D2 | Console LED 1 and 2 states (1200XL only) | | | | | | |
| 0 | LED on | | | | | | |
| 1 | LED off | | | | | | |
| D3:D2 | Extended bank select (130XE only) | | | | | | |
| 00 | Map \$10000-\$13FFF as extended bank | | | | | | |
| 01 | Map \$14000-\$17FFF as extended bank | | | | | | |
| 10 | Map \$18000-\$1BFFF as extended bank | | | | | | |
| 11 | Map \$1C000-\$1FFFF as extended bank | | | | | | |
| D4 | CPU extended memory access enable (130XE only) | | | | | | |
| 0 | CPU sees extended bank at \$4000-7FFF | | | | | | |
| 1 | CPU sees primary bank at \$4000-7FFF | | | | | | |
| D5 | ANTIC extended memory access enable (130XE only) | | | | | | |
| 0 | ANTIC sees extended bank at \$4000-7FFF | | | | | | |
| 1 | ANTIC sees primary bank at \$4000-7FFF | | | | | | |
| D7 | Self-test ROM enabled (XL/XE) | | | | | | |
| 0 | Map self-test ROM from \$D000-\$D7FF to \$5000-57FF if kernel ROM is enabled | | | | | | |
| 1 | Disable self-test ROM | | | | | | |

Description

PORTB originally accessed joystick ports 3 and 4 on the 800, but in later models with only two joystick ports it was repurposed for various other features.

| Unit | Address | Description |
|-------|------------------|-----------------------------|
| ANTIC | DMACTL \$D400 | DMA control (Write Only) |

Register layout

| | | | | | | |
|---------|----|----|----|----|-------|---|
| 7 | | | | | | 0 |
| Ignored | D5 | D4 | D3 | D2 | D1:D0 | |

D1:D0 Playfield width

| | |
|----|-------------------------------------|
| 00 | Disabled |
| 01 | Narrow playfield (128 color clocks) |
| 10 | Normal playfield (160 color clocks) |
| 11 | Wide playfield (192 color clocks) |

D2 Missile DMA enable

| | |
|---|---|
| 0 | Disabled (ignored if player DMA is enabled) |
| 1 | Enabled |

D3 Player DMA enable

| | |
|---|----------|
| 0 | Disabled |
| 1 | Enabled |

D4 Player/missile vertical resolution

| | |
|---|---------------------|
| 0 | Two-line resolution |
| 1 | One-line resolution |

D5 Display list DMA enable

| | |
|---|----------|
| 0 | Disabled |
| 1 | Enabled |

Description

The DMACTL register selectively enables DMA from ANTIC for various display items. For players and missiles, DMA mode must also be enabled in GTIA for it to take effect; otherwise, ANTIC will run DMA cycles but the object graphics will not be updated.

Missile DMA is enabled whenever player DMA is enabled, even if bit 2 is cleared. This is needed since GTIA interprets bus data depending on the number of cycles since the first time `HALT` is asserted during horizontal blank, and thus the timing of the missile DMA cycle determines which data is used for players.

| Unit | Address | Description |
|-------|------------------|-----------------------------------|
| ANTIC | CHACTL \$D401 | Character control (Write Only) |

Register layout

| | | | | |
|---|---------|----|----|----|
| 7 | | | | 0 |
| | Ignored | D2 | D1 | D0 |

| | |
|----|--|
| D0 | Character blink enable |
| 0 | Disabled |
| 1 | Hide characters with name bit 7 set |
| D1 | Character invert |
| 0 | Disabled |
| 1 | Invert image of characters with name bit 7 set |
| D2 | Vertical reflect |
| 0 | Display rows 0 through 7 (normal) |
| 1 | Display rows 7 through 0 (reflected) |

Description

CHACTL controls various features of 40 column text modes (ANTIC modes 2 and 3).

The blink bit does not actually cause characters to blink – it only selectively hides or shows some characters. To actually blink text, the blink bit must be periodically toggled.

Vertical reflection is performed by inverting the bits of the row counter used to fetch character data. This means that reflection may not work as expected for ANTIC mode 3 since the special case mapping for the descendant rows is not affected.

| Unit | Address | Description |
|-------|--------------------------------|--------------------------------------|
| ANTIC | DLISTL/DLISTH \$D402/\$D403 | Display list address (Write Only) |

Register layout

15

0

Display list address

Description

Set the current display list fetch address. Any writes to this register immediately redirect the display list, so it is recommended that it only be changed during vertical blank.¹⁴

The display list hardware only has a ten-bit counter. Display lists may be located anywhere in memory, but may not cross a 1K boundary without a jump instruction.¹⁵

¹⁴ [ATA82] III.6

¹⁵ [ATA82] III.5

| Unit | Address | Description |
|-------|------------------|--|
| ANTIC | HSCROL \$D404 | Horizontal scroll offset (Write Only) |

Register layout

| | |
|---------|-------------------|
| 7 | 0 |
| Ignored | Horizontal scroll |

D3:D0 Horizontal delay in color clocks

Description

This register adjusts the horizontal scroll amount for mode lines that have display list mode bit 4 set. Data display can be delayed by up to 15 color clocks, scrolling the playfield to the right. This does not affect the timing of the displayed window, so the left and right displayed margins for narrow and normal width playfields are not affected.

Playfield fetch timing is delayed by one cycle for every two color clocks of scroll. Odd values have the same fetch timing as even values, with the additional delay coming from an internal one-clock delay.

Odd delay values will give unexpected results for GTIA modes since the boundaries of the pixels are not adjusted to match the fetch delay. This causes pairs of bits to be pulled from adjacent pixels to form the four-bit values used for display.

| Unit | Address | Description |
|-------|------------------|--|
| ANTIC | VSCROL \$D405 | Vertical scroll offset (Write Only) |

Register layout

| | |
|---------|-----------------|
| 7 | 0 |
| Ignored | Vertical scroll |

D3:D0 Vertical delay in color clocks

Description

This register adjusts the vertical scroll amount for mode lines in a vertical scroll region. This includes any mode line with display list instruction bit 5 set and the next mode line after that.

For the first mode line in a vertically scrolled region, the VSCROL register sets the index of the first row displayed in the mode line. Increasing the scroll amount therefore shortens the first mode line by removing scan lines from the top. For the last mode line in a vertically scrolled region, increasing scroll values extends the last mode line by adding scan lines from the bottom.

It is possible to set VSCROL such that the row counter counts through values not normally valid for a mode line. When this happens, the mode line is extended as the row counter counts up to 15 and wraps around to 0. For text modes, only the low three bits are used to fetch data and thus rows 8-15 display the same data as rows 0-7.

VSCROL must be written by cycle 0 at the beginning of a mode line to affect the start of a scrolling region and by cycle 108 to determine whether the next scan line is the last scan line of an scroll-ending mode line.

Errata

The hardware manual [ATA82] shows only the lowest three bits being significant for 8-line display modes, but all four bits are significant in all display modes.

| Unit | Address | Description |
|-------|------------------|---|
| ANTIC | PMBASE \$D407 | Player/missile base address (Write Only) |

Register layout

| | |
|------------------|---------|
| 7 | 0 |
| P/M base address | Ignored |

D7:D2 Bits 10-15 of P/M base address (two-line resolution)

D7:D3 Bits 11-15 of P/M base address (one-line resolution)

Description

PMBASE sets the base address for fetching player/missile graphics. For one-line resolution, only the top five bits can be set, and therefore the P/M data must be aligned to a 2K boundary. For two-line resolution, the top six bits are settable and 1K alignment is required.

| Unit | Address | Description |
|-------|------------------|---|
| ANTIC | CHBASE \$D409 | Character data base address (Write Only) |

Register layout

| | |
|-----------------------------|------|
| 7 | 0 |
| Character data base address | Ign. |

D7:D1 Bits 9-15 of character data base address (ANTIC modes 2, 3, 4 and 5)

D7:D2 Bits 10-15 of character data base address (ANTIC modes 6 and 7)

Description

CHBASE sets the base address for fetching character data. Each character consists of an 8x8 block of monochrome data and occupies eight contiguous bytes. For ANTIC modes 2-5, CHBASE points to 128 characters starting at a 1K boundary, and for ANTIC modes 6-7, 64 characters starting at a 512 byte boundary.

| Unit | Address | Description |
|-------|-----------------|--|
| ANTIC | WSYNC \$D40A | Wait for Horizontal Sync (Write Only) |

Register layout

| | |
|---------|---|
| 7 | 0 |
| Ignored | |

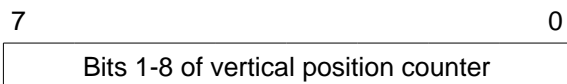
Description

A write to WSYNC causes the CPU to halt execution until the start of horizontal blank. One more cycle passes before the CPU is halted until cycle 105 on the current scan line. If the next cycle is free, the CPU executes the first cycle of the next instruction; otherwise, the next instruction starts at cycle 105. DMA contention at cycles 105 and 106 may cause the CPU restart to be delayed until as late as cycle 107.

Because the 6502 can only service an interrupt at the end of an instruction, use of WSYNC can cause excessively long delays in servicing interrupts. This is most serious with display list interrupts, where the delay can cause DLIs to occur on the wrong scan line or to be missed entirely.

| Unit | Address | Description |
|-------|------------------|-------------------------------|
| ANTIC | VCOUNT \$D40B | Vertical count (Read Only) |

Register layout



Description

VCOUNT allows the vertical position counter to be read to two-line resolution. For NTSC, VCOUNT runs from 0 to 130; for PAL, it runs from 0 to 155.

The VCOUNT register increments on cycle 111 of a scan line.

| Unit | Address | Description |
|-------|-----------------|---|
| ANTIC | NMIEN \$D40E | Non-maskable interrupt enable (Write Only) |

Register layout

| | | |
|-----|-----|---------|
| 7 | | 0 |
| DLI | VBI | Ignored |

D7 Display list interrupt enable

0 Disabled
1 Enabled

D6 Vertical blank interrupt enable

0 Disabled
1 Enabled

Description

NMIEN enables and disables NMI interrupts issued by ANTIC. This is required since the 6502 itself does not allow masking the NMI. Both interrupts are disabled automatically on system reset.¹⁶

The reset interrupt cannot be masked through NMIEN.¹⁷

¹⁶ Hardware II.28

¹⁷ Hardware III.1

| Unit | Address | Description |
|-------|-----------------|--|
| ANTIC | NMIST \$D40F | Non-maskable interrupt status (Read Only) |

Register layout

| | | | | | | | | |
|-----|-----|-----|---|---|---|---|---|---|
| 7 | | | | | | | | 0 |
| DLI | VBI | RES | 1 | 1 | 1 | 1 | 1 | |

D7 Display list interrupt status

0 Inactive
1 Active

D6 Vertical blank interrupt status

0 Inactive
1 Active

D5 System reset interrupt status (400/800 only)

0 Inactive
1 Active

Description

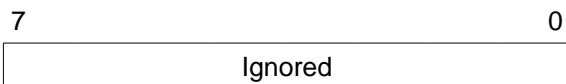
NMIST indicates which interrupt source in ANTIC triggered an NMI. The register layout is arranged so that a single BIT instruction can be used to very quickly check the DLI and VBI sources. A write to NMIRES is then used to clear status bits once the interrupt is serviced.

The DLI bit is automatically cleared when the VBI bit is set at scan line 248. Therefore, it is ordinarily never necessary to strobe NMIRES for either interrupt, as testing the DLI bit is sufficient to distinguish the two.

On the XL/XE series, the system reset button is hooked up to the RESET line rather than ANTIC's RNMI line, and thus the system reset NMI never occurs.

| Unit | Address | Description |
|-------|------------------|--|
| ANTIC | NMIRES \$D40F | Non-maskable interrupt reset (Write Only) |

Register layout



Description

A write to NMIRES resets the interrupt status bits in the NMIST register. This is only necessary for the NMI service routine to continue to identify the source of each interrupt – unlike for IRQs, the NMI is edge-triggered and therefore NMIRES does not need to be written to clear the interrupt itself.

Typically, NMIRES is only written when handling the vertical blank interrupt and not display list interrupts, because DLIs handlers are time critical. ANTIC assists this by automatically clearing the DLI bit in NMIST at the start of scan line 248.

8 Bibliography

- [LAN84] Lancaster, Don, Assembly Cookbook for the Apple II/Ile, 1984.
- [MOS76] MOS Technology, MCS6500 Microcomputer Family Hardware Manual, 1976.
- [MOS76a] MOS Technology, MCS6500 Microcomputer Family Programming Manual, 1976.
- [EYE86] Eyes, David and Lichty, Ron, Programming the 65816, 1986.
- [VIC09] VICE team, Documentation for the NMOS 65xx/85xx Instruction Set, 2009. Retrieved on July 19, 2009.
- [CHA85] Chadwick, Ian, Mapping the Atari, Revised Edition, 1985.
- [ATAXL] Atari, Atari Home Computer System Operating System Manual, XL Addendum, .
- [ATA82] Atari, Atari Home Computer System Hardware Manual, 1982.
- [AHS99] Atari, ANTIC CO12296 (NTSC) Rev. D, 1999.
- [AHS00] Atari, CGIA CO20577 (NTSC), 2000.
- [CRA82] Crawford, Chris, De Re Atari, 1982.
- [AHS03] Atari, POKEY CO12294, 2003.
- [AHS99a] Atari, GTIA CO14805 (NTSC), 1999.