Errata for the

Amiga Hardware Manual

The attached documents contain various updates to the *Amiga Hardware Manual.* The documents are:

Changes and Additions
: Read this first; it contains miscellaneous corrections and is a guide to all of the updates.

Examples
: Contains example assembly code sequences for the Copper, playfields, and sprites.

Addendum to Chapter 5: Audio Hardware
: Describes the audio state machine.

Addendum to Chapter 6: Blitter Hardware
: Describes blitter DMA and how the operation of the blitter affects the performance of the rest of the system.

Appendix F
: This replaces the present Appendix F.

Appendix G
: This replaces the present Appendix G.

Changes and Additions to the *Amiga Hardware Manual*

Changes and additions have been made to the following chapters and appendixes of the *Amiga Hardware Manual*.

Chapter 2: Coprocessor Hardware

Add the examples in the attached document called "Examples for the Amiga Hardware Manual".

Chapter 3: Playfield Hardware

Add the examples in the attached document called "Examples for the Amiga Hardware Manual".

Chapter 4: Sprite Hardware

Add the examples in the attached document called "Examples for the Amiga Hardware Manual".

On page 4-14, under section 4.3, "Displaying a Sprite", add the following:

CAUTION

> Should sprite DMA be turned off while a sprite is being displayed (that is, after VSTART but before VSTOP), the system will continue to display the line of sprite data that was most recently fetched. This causes a vertical bar to appear on the screen. It is recommended that sprite DMA only be turned off during vertical blanking or during some portion of the screen where you are ***sure*** that no sprite is being displayed.

Chapter 5: Audio Hardware

Add the attached document called "Addendum to Chapter 5: Audio Hardware".

Chapter 6: Blitter Hardware

Add the attached document called "Addendum to Chapter 6: Blitter Hardware".

On page 6-14, under section 6.11, "Line Drawing", add the following line to the table:

Register Name	Bit Number	Bit Name	State	Purpose
BLTCON1	5		0	Reserved for flabloden mode

On page 6-8, add the following to section 6.6, "Ascending and Descending Addressing":

If the source and destination data areas overlap in a blitter operation, there is a possibility of writing to a particular location as the destination before it was read as the source.

To prevent this kind of data destruction, you must take care to correctly choose ascending or descending mode, and possibly you should offset the source or destination.

Some types of overlapped blitter moves will always result in damaged data; therefore, the desired move must be done in two steps:

1. Move to a safe place.

2. Move to desired destination.

Using the blitter Memory Bus Activity Table[1], you can observe the order of operations and determine the required offset or mode. Pay careful attention to the notes. It helps to draw pictures.

[1] This table is located in the separate "Addendum to the Blitter Hardware Chapter" attached to this document.

Chapter 8: Interface Hardware

On page 8-4, under "Counter Limitations", the last line should read:

** 45-200 = -155. Because the absolute value exceeds 127, the true count must be

Appendix A

On page 4, in the table called "LINE MODE (line draw)", change the following bits for BLTCON1:

BIT#	BLTCON0	BLTCON1	
15	START3	TEXTURE3	
14	START2	TEXTURE2	
13	START1	TEXTURE1	
13	START0	TEXTURE0	
05	LF5	0	(Reserved for new mode)

On page 5, in the table called "LINE MODE (line draw)", change the line beginning "OVF ..." to:

0 Reserved for new mode

At the end of the same table, add:

The "B" source is used for texturing the drawn lines.

Appendix B

On page 2, after "BLTCDAT", "BLTBDAT", and "BLTADAT", change "&" to "%".

Appendix F

Remove Appendix F and insert the attached new Appendix F. Appendix F contains a new front section, called "Brief address map for 8520's".

Appendix G

Remove Appendix G and insert the attached new Appendix G. Appendix G has some minor changes.

Since this appendix is subject to change, anyone who is actually designing a device to interface to the Amiga should contact Amiga Third-Party Support.

Appendix I

This is a new appendix. It describes the interface to the 68000 bus connector.

Appendix F

Remove Appendix F and insert the attached new Appendix F. Appendix F contains a new front section, called "Brief address map for 8520's".

Appendix G

Remove Appendix G and insert the attached new Appendix G. Appendix G has some minor changes.

Since this appendix is subject to change, anyone who is actually designing a device to interface to the Amiga should contact Amiga Third-Party Support.

Appendix I

This is a new appendix. It describes the interface to the 68000 bus connector.

Examples for the

Amiga Hardware Manual

Chapter 2: Coprocessor Hardware

Add the following example to page 2-4, under section 2.3, "The MOVE Instruction".

The following MOVE instructions point bit-plane pointer 1 at $21000 and bitplane pointer 2 at $25000.

```
DC.W    $00E0,$0002    ;MOVE $0002 TO ADDRESS $0E0 (BPL1PTH)
DC.W    $00E2,$1000    ;MOVE $1000 TO ADDRESS $0E2 (BPL1PTL)
DC.W    $00E4,$0002    ;MOVE $0002 TO ADDRESS $0E4 (BPL2PTH)
DC.W    $00E6,$5000    ;MOVE $5000 TO ADDRESS $0E6 (BPL2PTL)
```

Add the following examples to page 2-5, under section 2.4, "The Wait Instruction".

This first WAIT instruction waits for scan line 150 (96 hex) with the horizontal position masked off.

```
DC.W    $9601,$FF00    ;WAIT FOR LINE 150, IGNORE HORIZONTAL COUNTERS
```

This second WAIT instruction waits for scan line 255 and horizontal position 254. This will never occur, so the copper stops until the next vertical blanking interval begins.

```
DC.W    $FFFF,$FFFE    ;WAIT FOR LINE 255, H = 254 (ENDS COPPER LIST).
```

Add the following example to page 2-10, under section 2.6, "Putting Together a Copper Instruction List".

This is an example of a complete Copper list. It is a Copper list for two bit-planes, one at $21000, one at $25000. At the top of the screen, the color registers are set as follows:

```
COLOR00 = WHITE
COLOR01 = RED
COLOR02 = GREEN
COLOR03 = BLUE
```

At line 150, the color registers are reloaded as follows:

```
COLOR00 = BLACK
COLOR01 = YELLOW
COLOR02 = CYAN
COLOR03 = MAGENTA

COPPERLIST:
        DC.W    $00E0,$0002     ;MOVE $0002 INTO ADDRESS $0E0 (BPL1PTH)
        DC.W    $00E2,$1000     ;MOVE $1000 INTO ADDRESS $0E2 (BPL1PTL)
        DC.W    $00E4,$0002     ;MOVE $0002 INTO ADDRESS $0E4 (BPL2PTH)
        DC.W    $00E6,$5000     ;MOVE $5000 INTO ADDRESS $0E6 (BPL2PTL)
;
; Load color registers
;
        DC.W    $0180,$0FFF     ;MOVE WHITE INTO ADDRESS $180 (COLOR00)
        DC.W    $0182,$0F00     ;MOVE RED INTO ADDRESS $182 (COLOR01)
        DC.W    $0184,$00F0     ;MOVE GREEN INTO ADDRESS $184 (COLOR02)
        DC.W    $0186,$000F     ;MOVE BLUE INTO ADDRESS $186 (COLOR03)
;
; Wait for line 150
;
        DC.W    $9601,$FF00     ;WAIT FOR LINE 150, IGNORE HORIZ. POSITION
;
; Reload color registers
;
        DC.W    $0180,$0000     ;MOVE BLACK INTO ADDRESS $0180 (COLOR00)
        DC.W    $0182,$0FF0     ;MOVE YELLOW INTO ADDRESS $0182 (COLOR01)
        DC.W    $0184,$00FF     ;MOVE CYAN INTO ADDRESS $0184 (COLOR02)
        DC.W    $0186,$0F0F     ;MOVE MAGENTA INTO ADDRESS $0186 (COLOR03)
;
; End copper list by waiting for the impossible
;
        DC.W    $FFFF,$FFFE     ;WAIT FOR LINE 255, H = 254 (NEVER HAPPENS)
```

Add the following to page 2-13, under section 2.8.1, "The SKIP Instruction".

The following SKIP instruction will skip the instruction following it if VP >= 100 ($64).

```
DC.W    $6401,$FF01     ;IF VP >= 100,SKIP NEXT INSTR (IGNORE HP).
DC.W    ....this is the instruction that will be skipped....
```

Replace the example on page 2-14, under section 2.8.2, "Copper Loops and Branches and Comparison Enable", with the following:

```
;
; Copper list to interrupt 68000 once every 16 scan lines,
; in the range VP = 80 through VP = 160.
;
    DC.W    $5001,$FFFE     ;WAIT FOR VP = $50, HP = 0
    DC.W    $0F01,$0F00     ;WAIT FOR VP = xxxx1111
;
; The following instruction writes to address $09C, the
; interrupt request register.  Writing $8010 sets the copper
; interrupt bit in the register, which will interrupt the 68000.
;
    DC.W    $009C,$8010     ;MOVE $8010 TO $09C (INTERRUPT 68000)
    DC.W    $A001,$FF01     ;SKIP NEXT INSTRUCTION IF VP >= 160
;
; The next MOVE instruction doesn't actually do a move.  It forces
; the copper to jump to the address in COP2LC.  This must have been
; previously set by either the copper or the 68000.  If VP >= 160,
; then this instruction will be skipped.
;
    DC.W    $008A,$0000     ;MOVE 0 TO COPJMP2 (COP2LC PREVIOUSLY SET)
    DC.W    ...other copper instructions
```

Chapter 3: Playfield Hardware

Add this example to page 3-10, under "Selecting Number of Bit-Planes".

This example shows how to write to the BPLCON0 register to tell the system to use 2 bitplanes.

```
;
; Writing $2200 to BPLCON0 sets the following conditions:
;
;       Low Resolution
;       Use two bitplanes
;       Hold-and-modify mode = OFF
;       Single playfield mode
;       Composite video color enabled
;       Genlock audio disabled
;       Light pen disabled
;       Interlace disabled
;       External resync disabled
;
BPLCON0 EQU $DFF100             ;BPLCON0 ADDRESS
        MOVE.W  #$2200,BPLCON0  ;WRITE TO IT
```

Add the following example to page 3-10, under section 3.2.3, "Selecting Horizontal and Vertical Resolution".

This example shows how to set the HIRES and LACE bits.

```
;
; Writing $A204 to BPLCON0 sets the following conditions:
;
;       High resolution
;       Use two bitplanes
;       Hold-and-modify mode = OFF
;       Single playfield mode
;       Composite video color enabled
;       Genlock audio disabled
;       Light pen disabled
;       Interlace enabled
;       External resync disabled
;
BPLCON0 EQU $DFF100             ;BPLCON0 ADDRESS
        MOVE.W  #$A204,BPLCON0  ;WRITE TO IT
```

Add the following example to page 3-12, under section 3.2.4, "Allocating Memory for Bit-Planes"

This example shows how to set pointers for memory allocation for bit-planes.

```
;
; Assuming two bitplanes, one at $21000 and the other at $25000, the processor
; sets BPL1PT to $21000 and BPL2PT to $25000. Normally, this is the copper's
; task.
;
BPL1PTH EQU $DFF0E0        ;HIGH THREE BITS OF BITPLANE 1 POINTER REGISTR
BPL1PTL EQU $DFF0E2        ;LOW FIFTEEN BITS
BPL2PTH EQU $DFF0E4        ;HIGH THREE BITS OF BITPLANE 2 POINTER REGISTR
BPL2PTL EQU $DFF0E6        ;LOW FIFTEEN BITS
;
; Two word writes are needed to write to a bitplane pointer, but we can use
; a single longword move. The 68000 writes the high order word to the lower-
; numbered address (BPLxPTH) and the low order word to the higher-numbered
; address (BPLxPTL).
;
     MOVE.L  #$21000,BPL1PTH      ;WRITE BITPLANE 1 POINTER
     MOVE.L  #$25000,BPL2PTH      ;WRITE BITPLANE 2 POINTER
```

Add the following examples to page 3-15, under "A One- or Two-Color Playfield"

This example shows how to define a 1-color bit-plane.

```
;
; This code fills a low resolution bit-plane with the background color (COLOR00)
; by writing all 0's into its memory area. The bitplane starts at $21000 and
; is 8000 bytes long.
;
     LEA $21000,A0           ;POINT AT BITPLANE
     MOVE.W  #2000,D0        ;WRITE 2000 LONGWORDS = 8000 BYTES
LOOP:  MOVE.L  #0,(A0)+      ;WRITE OUT A ZERO
     SUBQ.W  #1,D0           ;DECREMENT COUNTER
     BNE LOOP                ;LOOP UNTIL BITPLANE IS FILLED WITH 0'S
```

This example shows how to define a 2-color bit-plane.

```
;
; This code is identical to the last example, except the bitplane is filled
; with $FF00FF00 instead of all 0's.  This will produce two colors.
;
      LEA $21000,A0                ;POINT AT BITPLANE
      MOVE.W  #2000,D0             ;WRITE 2000 LONGWORDS = 8000 BYTES
LOOP:    MOVE.L  #$FF00FF00,(A0)+        ;WRITE OUT $FF00FF00
      SUBQ.W  #1,D0                ;DECREMENT COUNTER
      BNE LOOP                     ;LOOP UNTIL BITPLANE IS FULL
```

Add this example to page 3-18, under "Setting the Display Window Starting Position".

Setting DIWSTRT for the basic playfield:

```
;
; This code sets DIWSTRT for a basic playfield.  We write $2C for the vertical
; position and $81 for the horizontal position.
;
DIWSTRT EQU $DFF08E              ;DISPLAY WINDOW START REGISTER ADDRESS
;
      MOVE.W  #$2C81,DIWSTRT     ;WRITE IT OUT
```

Add this example to page 3-19, under "Setting the Display Window Stopping Position".

Setting DIWSTOP for the basic playfield:

```
;
; This code sets DIWSTOP for a basic playfield.  We write $F4 for
; the vertical position and $C1 for the horizontal position.
;
DIWSTOP EQU $DFF090              ;DISPLAY WINDOW STOP REGISTER ADDRESS
;
      MOVE.W  #$F4C1,DIWSTOP     ;WRITE IT OUT
```

Add the following examples to page 3-19, under Section 3.2.7, "Telling the System How to Fetch and Display Data".

Setting data fetch for the basic playfield:

```
;
; This code sets the data fetch start and stop values for a basic
; playfield.  We write $0038 to DDFSTRT and $00D0 into DDFSTOP.
;
DDFSTRT EQU $DFF092
DDFSTOP EQU $DFF094
;
        MOVE.W  #$0038,DDFSTRT    ;WRITE TO DDFSTRT
        MOVE.W  #$00D0,DDFSTOP    ;WRITE TO DDFSTOP
```

Setting modulo for the basic playfield:

```
;
; This code sets the modulo to 0 for a low resolution playfield with one
; bitplane.  For this example, the bitplane is one of the odd-numbered
; ones.  We would write to BPL2MOD for even-numbered bitplanes.
;
BPL1MOD EQU $DFF108          ;MODULO FOR ODD BITPLANES
;
        MOVE.W  #0,BPL1MOD   ;SET MODULO TO 0
```

Add the following examples to page 3-26, under a new section, 3.2.11, "Complete Example Playfields".

```
;
; This first example sets up a 320 x 200 pixel playfield with
; one bitplane.
; The bitplane lives at $21000.
; We also set up a copper list at $20000.
;
CUSTOM  EQU $DFF000
BPLCON0 EQU $100
BPLCON1 EQU $102
BPLCON2 EQU $104
BPL1MOD EQU $108
DDFSTRT EQU $092
DDFSTOP EQU $094
DIWSTRT EQU $08E
DIWSTOP EQU $090
VPOSR   EQU $004
COLOR00 EQU $180
COLOR01 EQU $182
```

```
COLOR02 EQU $184
COLOR03 EQU $186
DMACON EQU $096
COP1LCH EQU $080          ;COPPER LOCATION REGISTER 1 (HIGH 3 BITS)
;
     LEA CUSTOM,A0              ;A0 POINTS AT CUSTOM CHIPS
     MOVE.W #$1200,BPLCON0(A0)  ;ONE BITPLANE, ENABLE COMPOSITE COLOR
     MOVE.W #0,BPLCON1(A0)      ;SET HORIZONTAL SCROLL VALUE TO 0
     MOVE.W #0,BPL1MOD(A0)      ;SET MODULO TO 0 FOR ALL ODD BITPLANES
     MOVE.W #$0038,DDFSTRT(A0)  ;SET DATA FETCH START TO $38
     MOVE.W #$00D0,DDFSTOP(A0)  ;SET DATA FETCH STOP STOP TO $D0
     MOVE.W #$2C81,DIWSTRT(A0)  ;SET DIWSTRT TO $2C81
     MOVE.W #$F4C1,DIWSTOP(A0)  ;SET DISPLAY WINDOW STOP TO $F4C1
     MOVE.W #$0F00,COLOR00(A0)  ;SET BACKGROUND COLOR TO RED
     MOVE.W #$0FF0,COLOR01(A0)  ;SET COLOR REGISTER 1 TO YELLOW
;
; FILL BITPLANE WITH $FF00FF00 TO PRODUCE STRIPES
;
     MOVE.L  #$21000,A1         ;POINT AT BEGINNING OF BITPLANE
     MOVE.L  #$FF00FF00,D0      ;WE WILL WRITE $FF00FF00 LONG WORDS
     MOVE.W  #2000,D1           ;2000 LONG WORDS = 8000 BYTES
LOOP:  MOVE.L  D0,(A1)+         ;WRITE A LONG WORD
     SUBQ.W  #1,D1              ;DECREMENT COUNTER
     BNE LOOP                   ;LOOP UNTIL BITPLANE IS FILLED
;
; Set up copper list at $20000
;
     MOVE.L  #$20000,A1         ;POINT AT COPPER LIST DESTINATION
     LEA COPPERL,A2             ;POINT A2 AT COPPER LIST DATA
CLOOP:  MOVE.L  (A2),(A1)+      ;MOVE A WORD
     CMPI.L  #$FFFFFFFE,(A2)+   ;CHECK FOR LAST LONGWORD OF COPPER LIST
     BNE CLOOP                  ;LOOP UNTIL ENTIRE COPPER LIST IS MOVED
;
; Point copper at copper list
;
     MOVE.L  #$20000,COP1LCH(A0)  ;WRITE TO COPPER LOCATION REGISTER
     MOVE.W  COPJMP1(A0),D0       ;FORCE COPPER TO $20000
;
; Start DMA
;
     MOVE.W  #$8380,DMACON(A0)    ;ENABLE BITPLANE AND COPPER DMA
;
     BRA ....                   ;GO DO NEXT TASK
;
```

```
; This is the data for the copper list.
;
COPPERL:
     DC.W     $00E0,$0002          ;MOVE $0002 TO ADDRESS $0E0 (BPL1PTH)
     DC.W     $00E2,$1000          ;MOVE $1000 TO ADDRESS $0E2 (BPL1PTL)
     DC.W     $FFFF,$FFFE          ;END OF COPPER LIST
;


;
; This second example of a basic playfield sets up a hi resolution, interlaced
; display with one bitplane.
;
; The equates are the same as the previous example so they aren't repeated here.
;
     LEA CUSTOM,A0                ;ADDRESS OF CUSTOM CHIPS
     MOVE.W  #$9204,BPLCON0(A0)   ;HIRES, 1 BITPLANE, INTERLACE
     MOVE.W  #0,BPLCON1(A0)       ;HORIZONTAL SCROLL VALUE = 0
     MOVE.W  #80,BPL1MOD(A0)      ;MODULO = 80 FOR ODD BITPLANES
     MOVE.W  #80,BPL2MOD(A0)      ;DITTO FOR EVEN BITPLANES
     MOVE.W  #$003C,DDFSTRT(A0)   ;SET DATA FETCH START FOR HI RES
     MOVE.W  #$00D4,DDFSTOP(A0)   ;SET DATA FETCH STOP
     MOVE.W  #$2C81,DIWSTRT(A0)   ;SET DISPLAY WINDOW START
     MOVE.W  #$F4C1,DIWSTOP(A0)   ;SET DISPLAY WINDOW STOP
;
; Set up color registers
;
     MOVE.W  #$000F,COLOR00(A0)   ;BACKGROUND COLOR = BLUE
     MOVE.W  #$0FFF,COLOR01(A0)   ;FOREGROUND COLOR = WHITE
;
; Set up bitplane at $20000.
;
     LEA $20000,A1                ;POINT A1 AT BITPLANE
     LEA CHARLIST,A2              ;A2 POINTS AT CHARACTER DATA
     MOVE.W  #400,D1              ;WRITE 400 LINES OF DATA
     MOVE.W  #20,D0               ;WRITE 20 LONG WORDS PER LINE
L1:
     MOVE.L  (A2),(A1)+           ;WRITE A LONG WORD
     SUBQ.W  #1,D0                ;DECREMENT COUNTER
     BNE L1                       ;LOOP UNTIL LINE IS FULL
;
     MOVE.W  #20,D0               ;RESET LONG WORD COUNTER
     ADDQ.L  #4,A2                ;POINT AT NEXT WORD IN CHAR LIST
     CMPI.L  #$FFFFFFFF,(A2)      ;END OF CHAR LIST?
```

```
        BNE L2
        LEA CHARLIST,A2          ;YES, RESET A2 TO BEGINNING OF LIST
L2:
        SUBQ.W  #1,D1            ;DECREMENT LINE COUNTER
        BNE L1                   ;LOOP UNTIL ALL LINES ARE FULL
;
; Start DMA
;
        MOVE.W  #$8300,DMACON(A0)    ;ENABLE BITPLANE DMA ONLY, NO COPPER
;
; Since this example has no copper list, we sit in a loop waiting for vertical
; blank.  When it comes, we check the LOF (long frame) bit in VPOSR.  If it
; is a 0, then this is a short frame so we point the bitplane pointers to
; $200050; if LOF = 1, then this is a long frame so we point to $20000.  This
; keeps the long and short frames in the right relationship to each other.
;
VLOOP:
        MOVE.W  INTREQR(A0),D0       ;READ INTERRUPT REQUESTS
        AND.W   #$0020,D0            ;MASK OFF ALL BUT VERTICAL BLANK
        BEQ VLOOP                    ;LOOP UNTIL VERTICAL BLANK COMES
        MOVE.W  #$0020,INTREQ(A0)    ;RESET VERTICAL INTERRUPT
        MOVE.W  VPOSR(A0),D0         ;READ LOF BIT INTO D0 BIT 15
        BPL  VL1                     ;IF LOF = 0, JUMP
        MOVE.L  #$20000,BPL1PTH(A0)  ;LOF = 1, POINT TO $20000
        BRA VLOOP                    ;BACK TO TOP
VL1:
        MOVE.L  #$20050,BPL1PTH(A0)  ;LOF = 0, POINT TO $20050
        BRA VLOOP                    ;BACK TO TOP
;
; Character list
;
        DC.L $18FC3DF0,$3C6666D8,$3C66C0CC,$667CC0CC
        DC.L $7E66C0CC,$C36666D8,$C3FC3DF0,$00000000
        DC.L $FFFFFFFF
```

Add the following example to page 3-50, under "Specifying the Amount of Delay".

Setting the delay for horizontal scrolling:

```
;
; This code sets the horizontal scroll delay to 7 for both playfields.
;
BPLCON1 EQU $DFF102              ;HORIZONTAL SCROLL REGISTER
;
        MOVE.W  #$77,BPLCON1
```

Add the following example to page 3-52, under section 3.6.2, "Hold and Modify Mode".

Hold and modify example:

```
;
; This code generates a six bit-plane display with hold-and-modify mode
; turned on.  We load all 32 color registers with black to prove that
; the colors are being generated by hold-and-modify.
;
; The equates are the usual, so we won't repeat them here.
;
; First we set up the control registers.
;
        LEA CUSTOM,A0                ;POINT A0 AT CUSTOM CHIPS
        MOVE.W  #$6A00,BPLCON0(A0)   ;SIX BITPLANES, HOLD AND MODIFY MODE
        MOVE.W  #0,BPLCON1(A0)       ;HORIZONTAL SCROLL = 0
        MOVE.W  #0,BPL1MOD(A0)       ;MODULO FOR ODD BITPLANES = 0
        MOVE.W  #0,BPL2MOD(A0)       ;DITTO FOR EVEN BITPLANES
        MOVE.W  #$0038,DDFSTRT(A0)   ;SET DATA FETCH START
        MOVE.W  #$00D0,DDFSTOP(A0)   ;SET DATA FETCH STOP
        MOVE.W  #$2C81,DIWSTRT(A0)   ;SET DISPLAY WINDOW START
        MOVE.W  #$F4C1,DIWSTOP(A0)   ;SET DISPLAY WINDOW STOP
;
; Set all color registers = black to prove hold and modify mode is working.
;
        MOVE.W  #32,D0               ;INITIALIZE COUNTER
        LEA CUSTOM+COLOR00,A1        ;POINT A1 AT FIRST COLOR REGISTER
CREGLOOP:
        MOVE.W  #$0000,(A1)+         ;WRITE BLACK TO A COLOR REGISTER
        SUBQ.W  #1,D0                ;DECREMENT COUNTER
        BNE CREGLOOP                 ;LOOP UNTIL ALL COLOR REGISTERS SET.
;
; Fill six bitplanes with an easily recognizable pattern.
;
        MOVE.W  #2000,D0             ;2000 LONGWORDS PER BITPLANE
        MOVE.L  #$21000,A1           ;POINT A1 AT BIT PLANE 1
```

```
      MOVE.L   #$23000,A2          ;POINT A2 AT BIT PLANE 2
      MOVE.L   #$25000,A3          ;POINT A3 AT BIT PLANE 3
      MOVE.L   #$27000,A4          ;POINT A4 AT BIT PLANE 4
      MOVE.L   #$29000,A5          ;POINT A5 AT BIT PLANE 5
      MOVE.L   #$2B000,A6          ;POINT A6 AT BIT PLANE 6
FPLLOOP:
      MOVE.L   #$55555555,(A1)+    ;FILL BIT PLANE 1 WITH $55555555
      MOVE.L   #$33333333,(A2)+    ;FILL BIT PLANE 2 WITH $33333333
      MOVE.L   #$0F0F0F0F,(A3)+    ;FILL BIT PLANE 3 WITH $0F0F0F0F
      MOVE.L   #$00FF00FF,(A4)+    ;FILL BIT PLANE 4 WITH $00FF00FF
      MOVE.L   #$FFFFFFFF,(A5)+    ;FILL BIT PLANE 5 WITH $FFFFFFFF
      MOVE.L   #$00000000,(A6)+    ;FILL BIT PLANE 6 WITH $00000000
      SUBQ.W   #1,D0               ;DECREMENT COUNTER
      BNE FPLLOOP                  ;LOOP UNTIL ALL BIT PLANES ARE FULL.
;
; Set up a copper list at $20000.
;
      MOVE.L   #$20000,A1          ;POINT A1 AT COPPER LIST DESTINATION
      LEA COPPERL,A2               ;POINT A2 AT COPPER LIST IMAGE
CLOOP:  MOVE.L   (A2),(A1)+        ;MOVE A LONG WORD
      CMPI.L   #$FFFFFFFE,(A2)+    ;CHECK FOR END OF COPPER LIST
      BNE CLOOP                    ;LOOP UNTIL ENTIRE COPPER LIST MOVED.
;
; Point copper at copper list
;
      MOVE.L   #$20000,COP1LCH(A0)     ;LOAD COPPER JUMP REGISTER
      MOVE.W   COPJMP1(A0),D0          ;FORCE LOAD INTO COPPER P.C.
;
; Start DMA
;
      MOVE.W   #$8380,DMACON(A0)       ;ENABLE BITPLANE AND COPPER DMA
      BRA .....next stuff to do.....
;
; Copper list for six bit planes.  Bit plane 1 is at $21000, 2 is at $23000,
; 3 is at $25000, 4 is at $27000, 5 is at $29000, 6 is at $2B000.
;
COPPERL:
      DC.W     $00E0,$0002         ;BIT PLANE 1 POINTER = $21000
      DC.W     $00E2,$1000
      DC.W     $00E4,$0002         ;BIT PLANE 2 POINTER = $23000
      DC.W     $00E6,$3000
      DC.W     $00E8,$0002         ;BIT PLANE 3 POINTER = $25000
      DC.W     $00EA,$5000
      DC.W     $00EC,$0002         ;BIT PLANE 4 POINTER = $27000
```

```
DC.W    $00EE,$7000
DC.W    $00F0,$0002     ;BIT PLANE 5 POINTER = $29000
DC.W    $00F2,$9000
DC.W    $00F4,$0002     ;BIT PLANE 6 POINTER = $2B000
DC.W    $00F6,$B000
DC.W    $FFFF,$FFFE     ;WAIT FOR THE IMPOSSIBLE, I.E., QUIT
```

Chapter 4: Sprite Hardware"

Add the following example of a sprite data structure to page 4-9, under "Building the Data Structure".

```
;
; The following data is the data structure for the spaceship sprite.  It
; will be located at V = 65 and H = 128 on the screen.
;
SPRITE:
    DC.W    $6D60,$7200     ;VSTART, HSTART, VSTOP
    DC.W    $0990,$07E0     ;FIRST PAIR OF DESCRIPTOR WORDS
    DC.W    $13C8,$0FF0
    DC.W    $23C4,$1FF8
    DC.W    $13C8,$0FF0
    DC.W    $0990,$07E0
    DC.W    $0000,$0000     ;END OF SPRITE DATA
```

Add the following example to page 4-14, under section 4.3.1, "Selecting the Sprite DMA Channel and Setting the Pointers".

Initializing sprite data pointers:

```
;
; In this example the processor initializes the data pointers for sprite 0.
; Normally, this is done by the copper.  The sprite is at address $20000.
;
SPR0PTH EQU $DFF120         ;SPRITE 0 POINTER HIGH ORDER WORD
SPR0PTL EQU $DFF122         ;LOW ORDER WORD
;
    MOVE.L  #$20000,SPR0PTH ;WRITE $20000 TO SPRITE 0 POINTER
```

Add the following example to page 4-14 under Section 4.3, "Displaying a Sprite".

How to display a sprite:

```
;
; This example displays the spaceship sprite at location V=65, H=128.
;
; The equates are the usual, so they're not repeated here.
;
; First, we set up a single bit plane.
;
     LEA CUSTOM,A0                ;POINT A0 AT CUSTOM CHIPS
     MOVE.W  #$1200,BPLCON0(A0)     ;1 BIT PLANE, COLOR IS ON
     MOVE.W  #$0000,BPL1MOD(A0)     ;MODULO = 0
     MOVE.W  #$0000,BPLCON1(A0)     ;HORIZONTAL SCROLL VALUE = 0
     MOVE.W  #$0024,BPLCON2(A0)     ;SPRITES HAVE PRIORITY OVER PLAYFIELDS
     MOVE.W  #$0038,DDFSTRT(A0)     ;SET DATA FETCH START
     MOVE.W  #$00D0,DDFSTOP(A0)     ;SET DATA FETCH STOP
     MOVE.W  #$2C81,DIWSTRT(A0)     ;SET DISPLAY WINDOW START
     MOVE.W  #$F4C1,DIWSTOP(A0)     ;SET DISPLAY WINDOW STOP
;
; Set up color registers
;
     MOVE.W  #$0008,COLOR00(A0)     ;BACKGROUND COLOR = DARK BLUE
     MOVE.W  #$0000,COLOR01(A0)     ;FOREGROUND COLOR = BLACK
     MOVE.W  #$0FF0,COLOR17(A0)     ;COLOR 17 = YELLOW
     MOVE.W  #$00FF,COLOR18(A0)     ;COLOR 18 = CYAN
     MOVE.W  #$0F0F,COLOR19(A0)     ;COLOR 19 = MAGENTA
;
; Move copper list to $20000
;
     MOVE.L  #$20000,A1           ;POINT A1 AT COPPER LIST DESTINATION
     LEA COPPERL,A2               ;POINT A2 AT COPPER LIST SOURCE
CLOOP:  MOVE.L  (A2),(A1)+        ;MOVE A LONG WORD
     CMP.L   #$FFFFFFFE,(A2)+     ;CHECK FOR END OF LIST
     BNE CLOOP                    ;LOOP UNTIL ENTIRE LIST IS MOVED
;
; Move sprite to $25000
;
     MOVE.L  #$25000,A1           ;POINT A1 AT SPRITE DESTINATION
     LEA SPRITE,A2                ;POINT A2 AT SPRITE SOURCE
SPRLOOP:
     MOVE.L  (A2),(A1)+           ;MOVE A LONG WORD
     CMP.L   #$00000000,(A2)+     ;CHECK FOR END OF SPRITE
```

```
     BNE SPRLOOP                ;LOOP UNTIL ENTIRE SPRITE IS MOVED
;
; Now we write a dummy sprite to $30000, since all eight sprites are activated
; at the same time and we're only going to use one.  The remaining sprites
; will point to this dummy sprite data.
;
     MOVE.L   #$00000000,$30000    ;WRITE IT
;
; Point copper at copper list
;
     MOVE.L   #$20000,CUSTOM+COP1LC
;
; Fill bitplane with $FFFFFFFF
;
     MOVE.L   #$21000,A1           ;POINT A1 AT BIT PLANE
     MOVE.W   #2000,D0             ;2000 LONG WORDS = 8000 BYTES
FLOOP:  MOVE.L   #$FFFFFFFF,(A1)+  ;MOVE A LONG WORD OF $FFFFFFFF
     SUBQ.W   #1,D0                ;DECREMENT COUNTER
     BNE FLOOP                     ;LOOP UNTIL BITPLANE IS FULL
;
; Start DMA
;
     MOVE.W  CUSTOM+COPJMP1,D0   ;FORCE LOAD INTO COPPER
                                 ; PROGRAM COUNTER
     MOVE.W  #$83A0,(CUSTOM+DMACON)  ;BITPLANE, COPPER, AND SPRITE DMA
     BRA ....next stuff to do...

;
; This is a copper list for one bit plane, and 8 sprites.  The bit plane lives
; at $21000.  Sprite 0 lives at $25000; all others live at $30000 (the dummy
; sprite).
;
COPPERL:
     DC.W     $00E0,$0002          ;BIT PLANE 1 POINTER = $21000
     DC.W     $00E2,$1000
     DC.W     $0120,$0002          ;SPRITE 0 POINTER = $25000
     DC.W     $0122,$5000
     DC.W     $0124,$0003          ;SPRITE 1 POINTER = $30000
     DC.W     $0126,$0000
     DC.W     $0128,$0003          ;SPRITE 2 POINTER = $30000
     DC.W     $012A,$0000
     DC.W     $012C,$0003          ;SPRITE 3 POINTER = $30000
     DC.W     $012E,$0000
     DC.W     $0130,$0003          ;SPRITE 4 POINTER = $30000
```

```
        DC.W    $0132,$0000
        DC.W    $0134,$0003         ;SPRITE 5 POINTER = $30000
        DC.W    $0136,$0000
        DC.W    $0138,$0003         ;SPRITE 6 POINTER = $30000
        DC.W    $013A,$0000
        DC.W    $013C,$0003         ;SPRITE 7 POINTER = $30000
        DC.W    $013E,$0000
        DC.W    $FFFF,$FFFE         ;END OF COPPER LIST
;
; Sprite data for spaceship sprite.  It appears on the screen at V=65 and
; H=128.
;
SPRITE:
        DC.W    $6D60,$7200         ;VSTART, HSTART, VSTOP
        DC.W    $0990,$07E0         ;FIRST PAIR OF DESCRIPTOR WORDS
        DC.W    $13C8,$0FF0
        DC.W    $23C4,$1FF8
        DC.W    $13C8,$0FF0
        DC.W    $0990,$07E0
        DC.W    $0000,$0000         ;END OF SPRITE DATA
```

Add the following example to page 4-15 under Section 4.4, "Moving a Sprite".

```
;
; This is an example of moving a sprite.  We bounce the spaceship around
; on the screen, making it change direction whenever it reaches an edge.
;
; The sprite position data, containing VSTART and HSTART, lives in memory
; at $25000.  VSTOP is located at $25002.  We write to these locations to
; move the sprite.
;
; Once each frame, we increment (or decrement) VSTART by 1 and HSTART by 2.
; Then we calculate VSTOP which will be the new VSTART + 6.
;
      MOVE.B   #151,D0          ;INITIALIZE HORIZONTAL COUNT
      MOVE.B   #194,D1          ;INITIALIZE VERTICAL COUNT
      MOVE.B   #64,D2           ;INITIALIZE HORIZONTAL POSITION
      MOVE.B   #44,D3           ;INITIALIZE VERTICAL POSITION
      MOVE.B   #1,D4            ;INITIALIZE HORIZONTAL INCREMENT VALUE
      MOVE.B   #1,D5            ;INITIALIZE VERTICAL INCREMENT VALUE
;
; Here we wait for the vertical blanking bit in INTREQR to turn on.  This
; ensures a glitch free display.
;
VLOOP:  MOVE.W  CUSTOM+INTREQR,D6  ;READ INTERRUPT REQUEST WORD
      AND.W    #$0020,D6          ;MASK OFF ALL BUT VERTICAL BLANK BIT
      BEQ VLOOP                   ;LOOP UNTIL BIT IS A 1
      MOVE.W   #$0020,CUSTOM+INTREQ     ;VERTICAL BIT IS ON, SO RESET IT.
;
      ADD.B    D4,D2              ;INCREMENT HORIZONTAL VALUE
      SUBQ.B   #,D0               ;DECREMENT HORIZONTAL COUNTER
      BNE L1
      MOVE.B   #151,D0            ;COUNT EXHAUSTED, RESET TO 151
      EOR.B    #$FE,D4            ;NEGATE THE INCREMENT VALUE
L1:   MOVE.B   D2,$25001          ;WRITE NEW HSTART VALUE TO SPRITE
      ADD.B    D5,D3              ;INCREMENT VERTICAL VALUE
      SUBQ.B   #1,D1              ;DECREMENT VERTICAL COUNTER
      BNE L2
      MOVE.B   #194,D1            ;COUNT EXHAUSTED, RESET TO 194
      EOR.B    #$FE,D5            ;NEGATE THE INCREMENT VALUE
L2:   MOVE.B   D3,$25000          ;WRITE NEW VSTART VALUE TO SPRITE
      MOVE.B   D3,D6              ;MUST NOW CALCULATE NEW VSTOP
      ADD.B    #6,D6              ;VSTOP ALWAYS VSTART+6 FOR SPACESHIP
      MOVE.B   D6,$25002          ;WRITE NEW VSTOP TO SPRITE
      BRA VLOOP                   ;LOOP FOREVER.
```

Add the following example to page 4-18 under Section 4.6, "Reusing Sprite DMA Channels".

```
;
; This example displays the spaceship sprite and then redisplays it as a
; different object.  I'm not going to tell you what it is, that's a secret.
;
; Only the sprite data list is affected by this, so that's all we'll show
; here.  However, it looks best with the color registers set as follows:
;
        LEA  CUSTOM,A0
        MOVE.W  #$0F00,COLOR17(A0)      ;COLOR 17 = RED
        MOVE.W  #$0FF0,COLOR18(A0)      ;COLOR 18 = YELLOW
        MOVE.W  #$0FFF,COLOR19(A0)      ;COLOR 19 = WHITE
;
; And now for the sprite data.
;
SPRITE:
        DC.W    $6D60,$7200
        DC.W    $0990,$07E0
        DC.W    $13C8,$0FF0
        DC.W    $23C4,$1FF8
        DC.W    $13C8,$0FF0
        DC.W    $0990,$07E0
        DC.W    $8080,$8D00             ;VSTART, HSTART, VSTOP FOR NEW SPRITE
        DC.W    $1818,$0000
        DC.W    $7E7E,$0000
        DC.W    $7FFE,$0000
        DC.W    $FFFF,$2000
        DC.W    $FFFF,$2000
        DC.W    $FFFF,$3000
        DC.W    $FFFF,$3000
        DC.W    $7FFE,$1800
        DC.W    $7FFE,$0C00
        DC.W    $3FFC,$0000
        DC.W    $0FF0,$0000
        DC.W    $03C0,$0000
        DC.W    $0180,$0000
        DC.W    $0000,$0000             ;END OF SPRITE DATA
```

Add the following example to page 4-23 under section 4.8, "Attached Sprites".

```
;
; The following data structure is for the six color spaceship made with two
; attached sprites.
;
SPRITE0:
        DC.W    $6D60,$7200         ;VSTART = 65, HSTART = 128
        DC.W    $0C30,$0000         ;FIRST COLOR DESCRIPTOR WORD
        DC.W    $1818,$0420
        DC.W    $342C,$0E70
        DC.W    $1818,$0420
        DC.W    $0C30,$0000
        DC.W    $0000,$0000         ;END OF SPRITE 0
;
SPRITE1:
        DC.W    $6D60,$7280         ;SAVE AS SPRITE 0 EXCEPT ATTACH BIT ON.
        DC.W    $07E0,$0000         ;FIRST DESCRIPTOR WORD FOR SPRITE 1
        DC.W    $0FF0,$0000
        DC.W    $1FF8,$0000
        DC.W    $0FF0,$0000
        DC.W    $07E0,$0000
        DC.W    $0000,$0000         ;END OF SPRITE 1
```

Addendum to Chapter 5: Audio Hardware

Audio State Machine

There is one audio state machine for each channel. See the attached state diagram. The machine has eight states and is clocked at the system clock frequency of 3.58 MHz. Three of the states are basically unused and just transfer back to the idle (000) state. One of the paths out of the idle state is designed for interrupt-driven operation (processor provides data), and the other path is designed for DMA-driven operation (Agnus provides data).

In interrupt-driven operation, transfer to the main loop (010,011 states) is immediate upon data written by the processor. In the 010 state the upper byte is output, and in the 011 state the lower byte is output. Transitions 010→011→010 . . . occur whenever the period counter counts down to one. The period counter is reloaded at these transitions. As long as the interrupt is cleared by the processor in time, the machine remains in the main loop. Otherwise, it enters the idle state. Interrupts are generated on every word (011→010) transition.

In DMA-driven operation, transit to the 001 state occurs and DMA requests are sent to Agnus as soon as DMA is turned on. Because of pipelining in Agnus, the first data word must be thrown away. State 101 is entered as soon as this word arrives. A request for the next data word has already gone out. When the data arrives, state 010 is entered and the main loop continues until the DMA is turned off. The length counter counts down once with each word that comes in. When it finishes, a DMA restart request goes to Agnus along with the regular DMA request. This tells Agnus to reset the pointer to the beginning of the table of data. Also, the length counter is reloaded and an interrupt request goes out soon after the length counter finishes (counts to one). The request goes out just as the last word of the waveform starts its output.

DMA requests and restart requests are transferred to Agnus once each horizontal line, and the data comes back about 14 clock cycles (of 280ns) later.

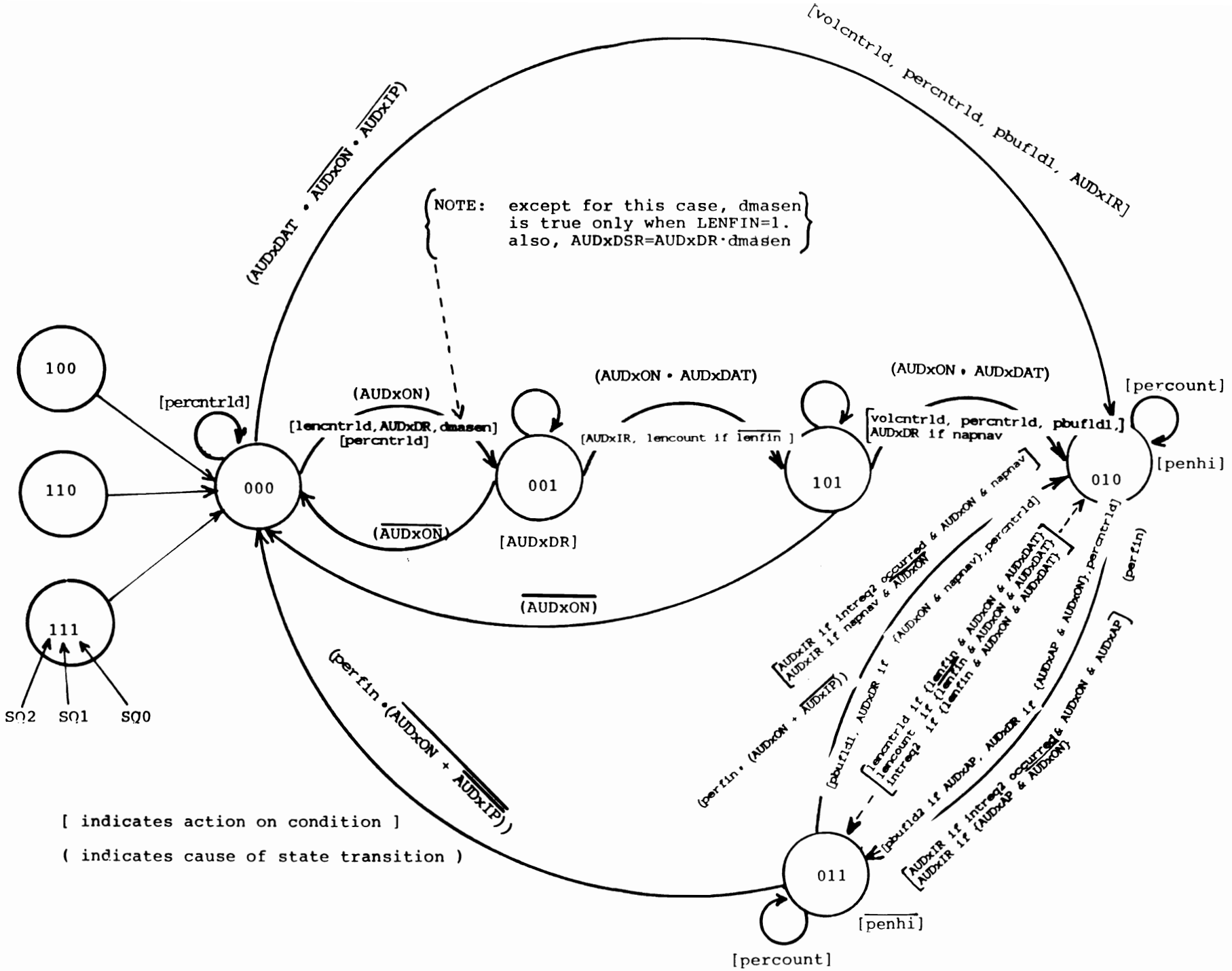
In attach mode, things run a little differently. Attach volume has requests the same as normal operation (on the 011→010) transition). In attach period, a set of requests occurs on the 010→011 transition. With both attach period and attach volume high, requests occur on both transitions.

One of the consequences of the way this machine works is that if the sampling rate is set much higher than the normal max sampling rate (~ 29 kHz), the two samples in the buffer register will be repeated. If the filter on the Amiga is bypassed and the volume is set to max (40 hex), this feature can be used to make modulated carriers up to 1.79 MHz. The modulation is placed in the memory map, with plus values in the even bytes, and minus values in the odd bytes.

The following list shows the symbols used in the state diagram. Capitals are external signals; small letters are local signals.

AUDxON	DMA on "x" indicates channel number (signal from DMACON).
AUDxIP	Audio interrupt pending (input to channel from interrupt circuitry)
AUDxIR	Audio interrupt request (output from channel to interrupt circuitry)
intreq1	Interrupt request that combines with intreq2 to form AUDxIR
intreq2	Prepare for interrupt request. Request comes out after the next 011→010 transition in normal operation.
AUDxDAT	Audio data load signal. Loads 16 bits of data to audio channel.
AUDxDR	Audio DMA request to Agnus for one word of data.
AUDxDSR	Audio DMA request to Agnus to reset pointer to start of block
dmasen	Restart request enable.
percntrld	Reload period counter from backup latch typically written by processor with AUDxPER (can also be written by attach mode).
percount	Count period counter down one latch.
perfin	Period counter finished (value = 1).
lencntrld	Reload length counter from backup latch.
lencount	Count length counter down one notch.
lenfin	Length counter finished (value = 1).
volcntrld	Reload volume counter from backup latch.
pbufld1	Load output buffer from holding latch written to by AUDxDAT.
pbufld2	Like pbufld1, but only during 010→011 with attach period.
AUDxAV	Attach volume. Send data to volume latch of next channel instead of to D→A converter.

AUDxAP	Attach period. Send data to period latch of next channel instead of to the D→A converter.
penhi	Enable the high 8 bits of data to go to the D→A converter.
napnav	/AUDxAV * /AUDxAP + AUDxAV—no attach stuff or else attach volume. Condition for normal DMA and interrupt requests.
sq2,1,0	The name of the state flip-flops, MSB to LSB.


[volcntrld, percntrld, pbufld1, AUDxIR]
(AUDxDAT • AUDxON • AUDxIP)
NOTE: except for this case, dmasen is true only when LENFIN=1. also, AUDxDSR=AUDxDR·dmasen
100
110
111
SQ2
SQ1
SQ0
000
001
101
010
011
[percntrld]
(AUDxON)
[lencntrld, AUDxDR, dmasen]
[percntrld]
[AUDxDR]
(AUDxON • AUDxDAT)
[AUDxIR, lencount if lenfin]
(AUDxON • AUDxDAT)
[volcntrld, percntrld, pbufld1,]
AUDxDR if napnav
[percount]
[penhi]
(AUDxON)
(perfin • (AUDxON + AUDxIP))
[AUDxIR if intreq2 occurred & AUDxON & napnav]
[AUDxIR if napnav & AUDxON]
(pbufld1, AUDxDR if (AUDxON & napnav), percntrld)
[lencntrld if (lenfin & AUDxON & AUDxDAT)
lencount if (lenfin & AUDxON & AUDxDAT)
intreq2 if (lenfin & AUDxON & AUDxDAT)]
(perfin)
[pbufld2 if AUDxAP, AUDxDR if (AUDxAP & AUDxON), percntrld]
[AUDxIR if intreq2 occurred & AUDxON & AUDxAP]
[AUDxIR if (AUDxAP & AUDxON)]
[penhi]
[percount]
[indicates action on condition]
(indicates cause of state transition)

Addendum to Chapter 6: Blitter Hardware

Blitter Operations and System DMA

This section explains how the operation of the blitter affects the performance of the rest of the system. The section covers the following topics:

- Blitter direct memory access (DMA) priority
- DMA time slot allocation
- Bus sharing between the 68000 and the bit-plane operations of the blitter and Copper.
- Effects of different playfield display sizes on sprite display
- Effects of blitter operation on the 68000's access to memory

1. Blitter DMA Priority

The blitter performs its various data fetch, modify, and store operations through a DMA sequence. It shares memory access with other devices in the system. In a system like the Amiga that uses a lot of DMA there must be some control over the priority of the accessing devices. A device's priority indicates its importance relative to other devices. The list below shows the order of priority for DMA operations, from highest priority to lowest.

- Disk DMA — handles communications with the disk
- Audio DMA — produces the sound
- Bit-plane DMA — produces the static display
- Sprite DMA — produces the dynamic display
- Copper — display-synchronized coprocessor
- Blitter — data copying and line drawing device

- o 68000 microprocessor—central processor

The first four devices in the list all have the same priority. If a disk DMA cycle is missed, some disk data is lost. If an audio cycle is missed, noise is added to the audio output. In displays (bit-plane or sprite), there may be flashes or other interruptions on-screen. None of these situations are desirable and are avoided by making all of these devices priority 1. Under certain circumstances, however, the bit-plane DMA will take priority over sprite DMA.

Each of the first four devices in the list is specifically allocated a group of time slots during each horizontal scan of the video beam. If the device does not specifically request to use one of its allocated time slots, the slot is open for other uses.

The Copper has the next priority because it is designed as a display-synchronized coprocessor. It has to perform its operations at the same time during each display frame to remain synchronized with the display beam sweeping across the screen.

At the bottom of the list come the blitter and the 68000, in that order. The blitter is optimized for data copying, modifying, and line drawing operations. It performs these kinds of operations much faster than the 68000 could perform them. It is, therefore, given a higher priority than the processor so that it can do its job in the most efficient manner.

2. DMA Time Slot Allocation

During a horizontal scan line (about 63 microseconds), there are 227.5 "color-clocks". A color clock is the basic timing interval for memory access within this system and amounts to approximately 280 ns.

This interval of 227.5 color clocks includes both display time and non-display time on a horizontal line. Of this total time, there are 226 possible memory access cycles at 280 ns each to be allocated to the various devices that need memory access.

Each of the first four time slots in the list below is assigned at one of the odd-numbered slots available. This assignment was made to allow maximum bus utilization by the 68000 (as shown in the section titled "Bit-Plane/Processor Bus Sharing"). The even-numbered slots can be used by the Copper, blitter, or 68000.

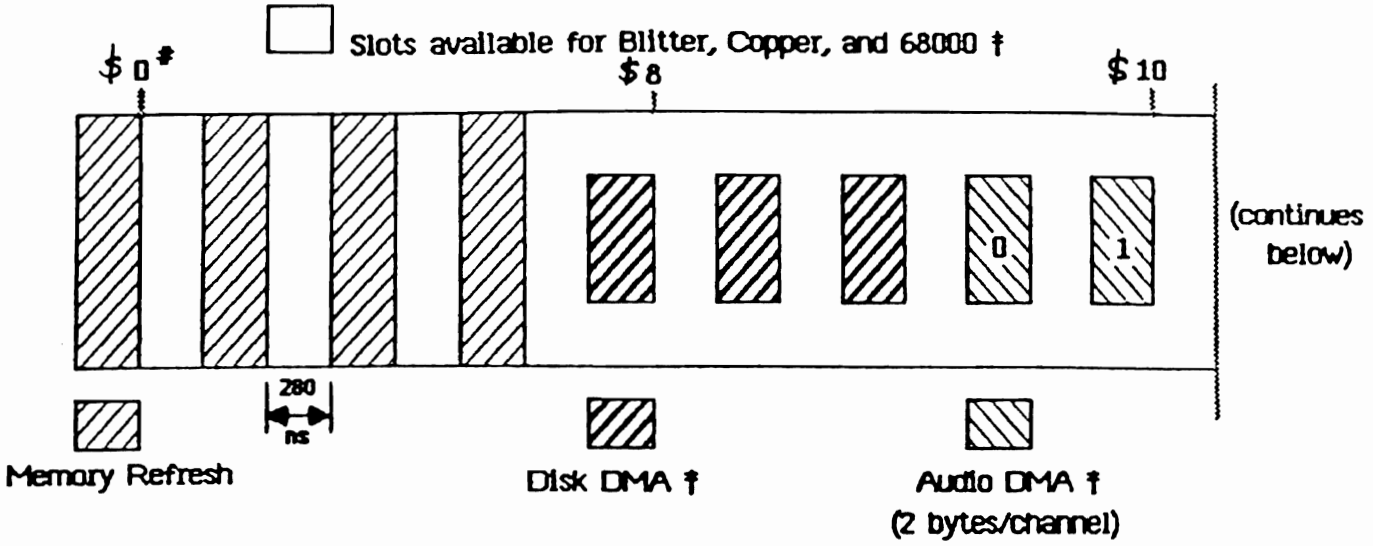
Here is the time-slot allocation per horizontal line:

- o 4 cycles for memory refresh

- 3 cycles for disk DMA
- 4 cycles for audio DMA (2 bytes per channel)
- 16 cycles for sprite DMA (2 words per channel)
- 80 cycles for bit-plane DMA (even and/or odd according to the display size used)

The figures on the following pages show one complete horizontal scan line and how the clock cycles are allocated.

DMA Time Slot Allocation / Horizontal line



† These operations only take slots if the associated operation is being performed

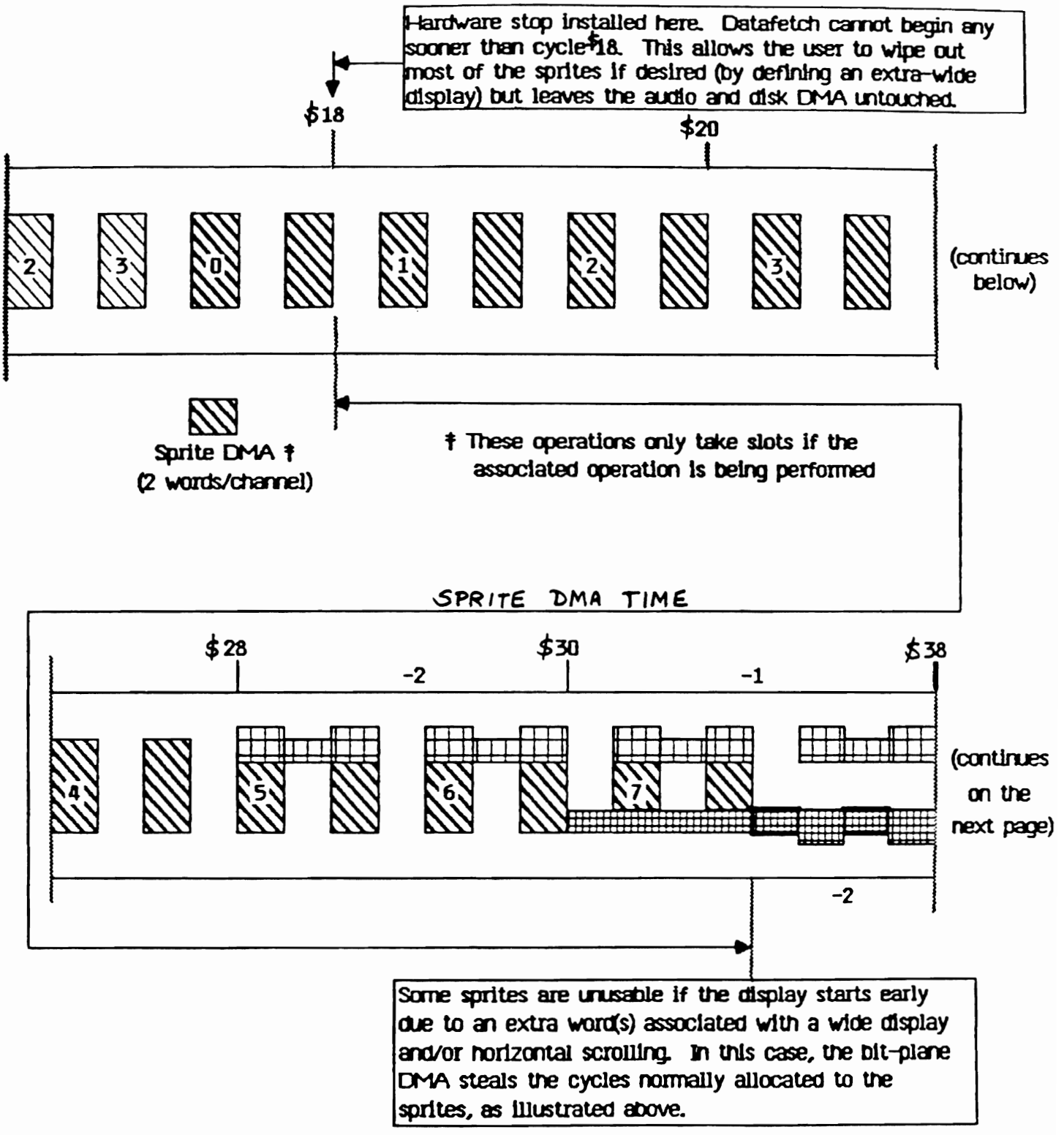
Note: Copper Data Move instructions require 4 slots.
Copper Wait instructions require 6 slots.

This cycle 0 appears to exclude one of the memory refresh cycles. This is not the case.

Actual system hardware demands certain specific values for data fetch start and display start. Therefore this timing chart has been "adjusted" to match those requirements.

$ Indicates a hex number.

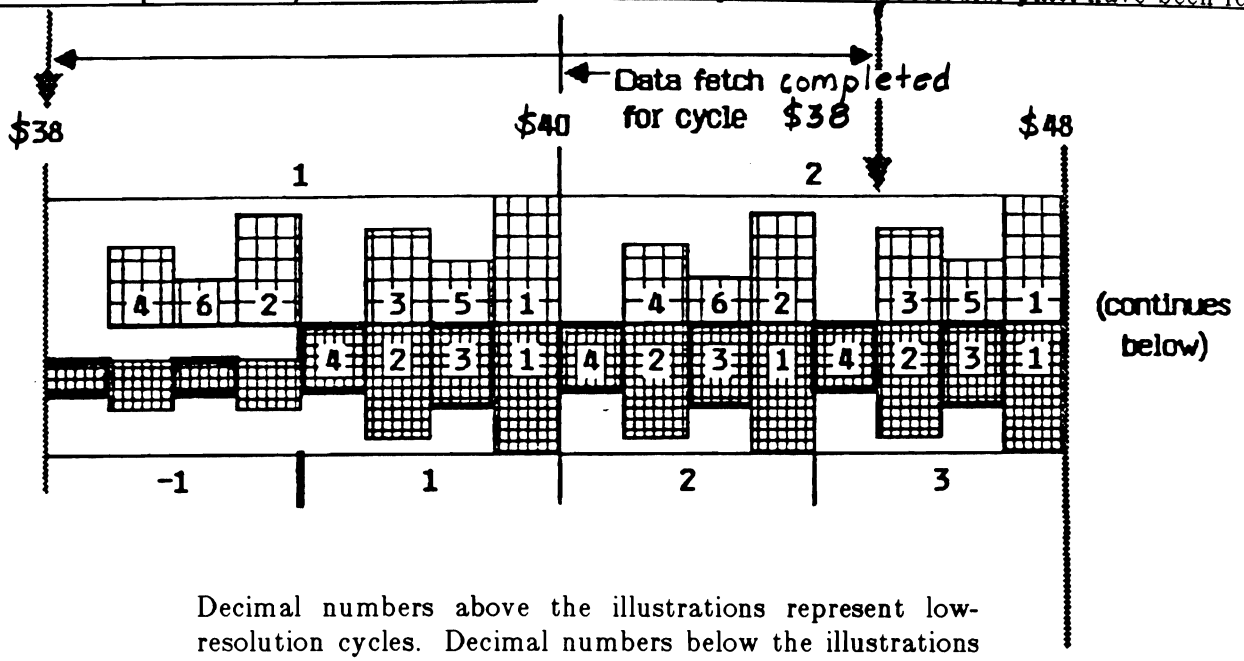
DMA Time Slot Allocation / Horizontal line (cont'd)



DMA Time Slot Allocation / Horizontal line (cont'd)

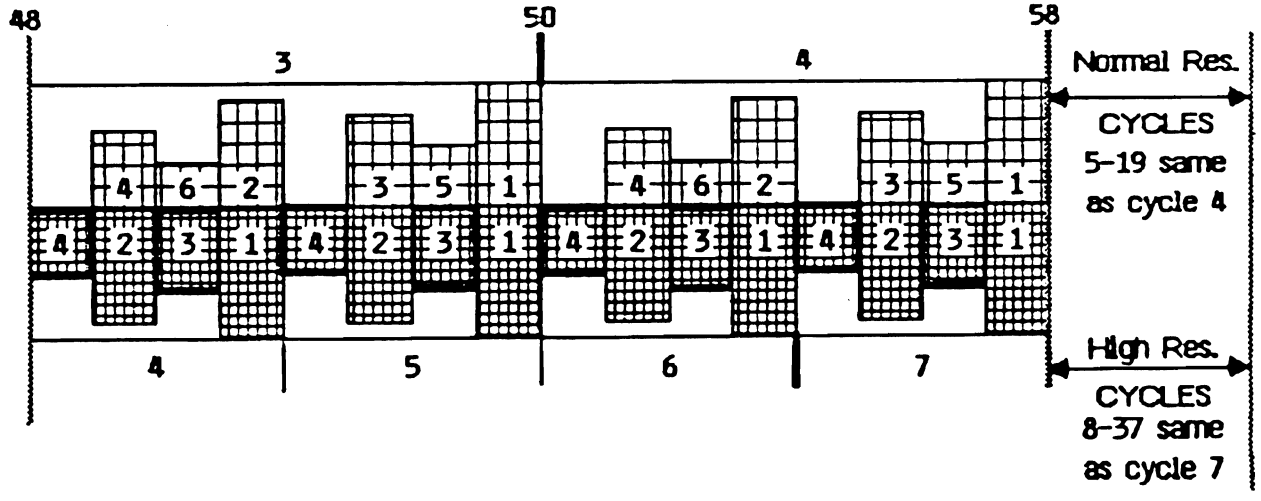
Data fetch start can only be specified at even multiples of 8 clocks. This is the clock position which should be specified for the normal width display. (20 word fetch for 320 pixel, 40 word fetch for 640 pixel width).

Five clocks must occur before the data fetched for a particular position can appear on-screen. For example, if data fetch start is $38, data will not be available for display until clock number $45. It is available at $45 because display processing does not begin until all of the bit-planes for a particular pixel have been fetched.

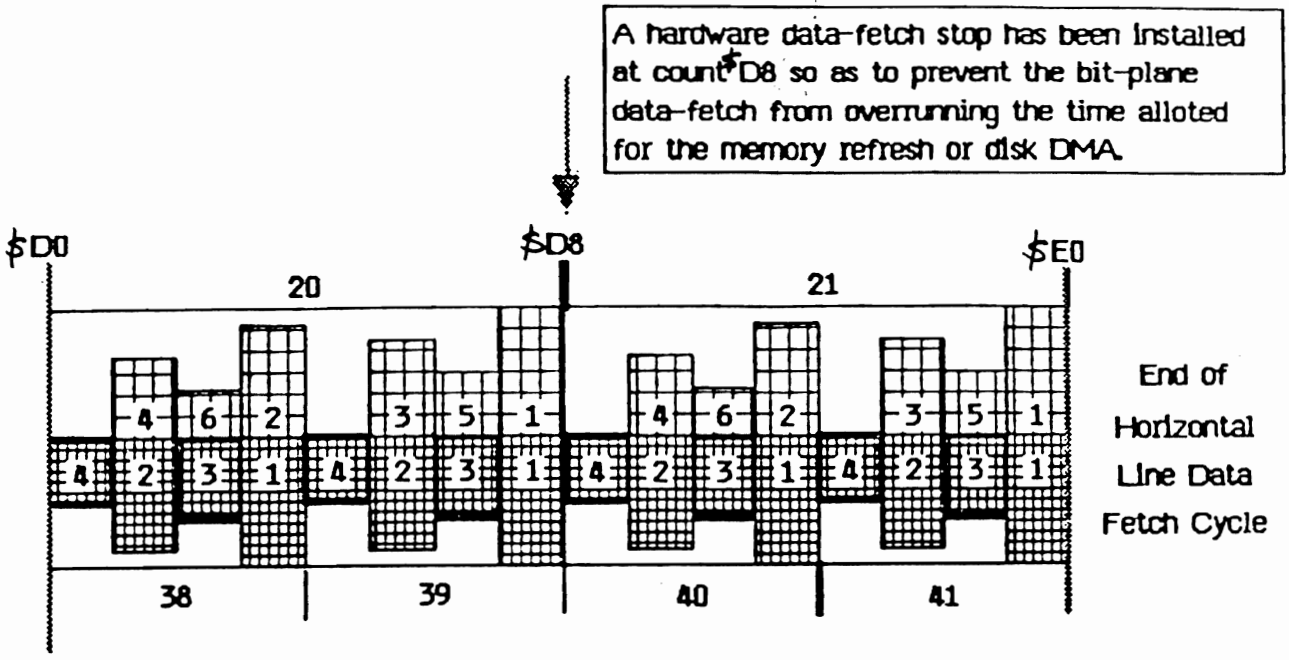


Decimal numbers above the illustrations represent low-resolution cycles. Decimal numbers below the illustrations represent high-resolution cycles. Negative numbers indicate the start of data fetch for displays that are larger than normal.

Decimal numbers inside the illustrations represent the bit-plane for which the data is being fetched.



DMA Time Slot Allocation / Horizontal line (cont'd)

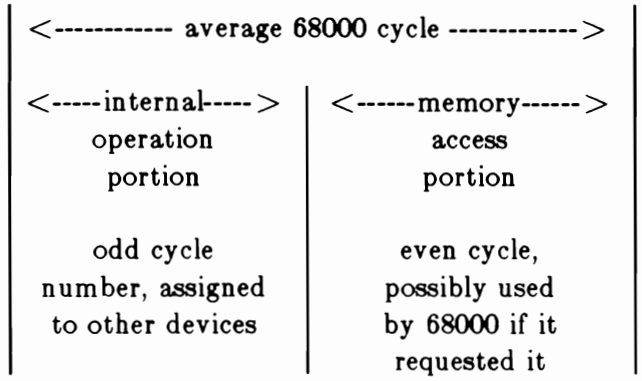


3. Bit-Plane/Processor Bus Sharing

The memory access cycles are interleaved to allow the 68000 processor to operate at close to its maximum speed. The 68000 spends about half of a complete processor instruction time doing internal operations and the other half accessing memory. Therefore, if the 68000 is given each alternate 280 ns memory cycle, it will appear to the 68000 that it has the memory all of the time and it will run at full speed.

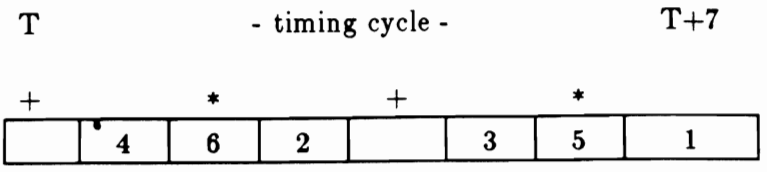
Not all of the 68000 instructions allow this even-cycle allocation to mate perfectly all of the time. If it doesn't, the processor will have to wait until its next memory slot is available before continuing. Most 68000 instructions do not cause cycles to be missed, so it will run at full speed most of the time if there is no blitter DMA interference.

The following figure illustrates the normal cycle of the 68000.



If there are 4 or less low-resolution bit-planes, then the 68000 can be granted each alternate memory cycle (if it is ready to ask for the cycle and is the highest priority item at the time). However, if there are more than 4 bit-planes, the bit-plane DMA will begin to steal cycles from the 68000 during the display time.

The following figure illustrates the time slots (each 280 ns) that will be taken by bit-plane DMA during the display time (160 slots out of 227 for each horizontal line for a 320-pixel, low-resolution display) for a 6-bit-plane display. As you will see from the figure, the bit-plane steals 50% of the open slots that the processor might have used if there were only 4 bit-planes displayed.

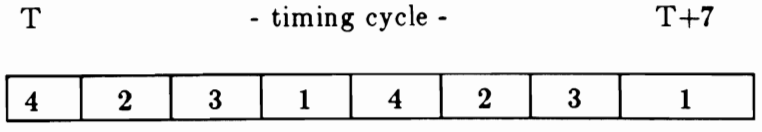


\+ an open memory slot that the 68000 might use

* a slot that cannot be used by the 68000 due to added bit-plane DMA

If you specify 4 high-resolution bit-planes (640 pixels wide), bit-plane DMA needs *all* of the available memory time slots during the display time just to grab the 40 data words for each line of the 4 bit-planes (40 X 4 = 160 time slots). This effectively locks out the 68000 (as well as the blitter or Copper) from any memory access during the actual period of the display.

The following figure shows how the time slots are allocated for high-resolution bit-planes.



4. Effects of Different Display Sizes

The normal, full-sized screen consists of 320 pixels for low-resolution mode or 640 pixels for high-resolution mode. This means that either 20 or 40 words will be fetched during the horizontal line display time.

When you want to scroll either or both playfields, one extra data word per line must be fetched from the memory. This extra word provides the pixels that will show on-screen as scrolling occurs.

Screen size is adjustable (see the descriptions of DIWSTRT, DIWSTOP, and DDFSTRT in Chapter 3, "Playfield Hardware". Bit-plane DMA takes precedence over sprite DMA. Therefore, larger screens may block out one or more of the highest-numbered sprites, especially with scrolling.

5. Effects of Blitter Operation

As mentioned above, the blitter normally has a higher priority than the processor for DMA cycles. There are certain cases, however, where the blitter and the 68000 will share memory cycles. The blitter uses every cycle. If given the chance, it would steal every available memory cycle. Display, disk, and audio DMA take precedence, so the blitter cannot block them from bus access. Depending on the state of the setting of the blitter DMA mode bit, commonly referred to as the "blitter-nasty" bit, the processor may not be so lucky. This bit is called BLTPRI (for "blitter has priority over processor") and is in register DMACONW.

If BLTPRI is a 1, the blitter will hang onto the bus for every available memory cycle. In the non-display time (horizontal blanking interval), this could potentially be every single cycle. Even in blitter-nasty mode, there may be cases when you can grant a few cycles to the 68000. Even though the blitter would seem to be acting as a bus-hog, there are data windows that occur during certain blitter operations when the blitter is doing something internally rather than on the bus. These cycles are part of the basic operating sequence of the blitter itself.

If BLTPRI is a 0, the DMA manager will monitor the 68000 cycle requests. If the 68000 is unsatisfied for three consecutive memory cycles, the blitter will release the bus for 1 cycle.

The DMA manager knows when these cycles of the blitter are about to occur and will release them to the 68000 if it is waiting for memory access. Blitter-nasty mode doesn't *prevent* the 68000 from executing instructions, but it surely slows it down.

Table 1 shows all of the possible operating modes of the blitter along with the distribution of the memory access windows within its operation. The table shows three words of a blit (the first, any middle, and the last) and how the bus activity occurs for this sequence. The following conventions are used:

- A, B, and C stand for the sources.
- D stands for the destination.
- A0 is the first memory word fetch; A1 is any middle memory word fetch; A2 is the last memory word fetch.
- An asterisk (*) following a source name indicates an internal cycle used for that source; the cycle does no data fetch. For these functions, the cycle fetch columns have blank spaces that indicate open memory cycles where some other device can access the bus. Thus functions such as item 3, ABC*->D can be read as AB->D.

- o Z stands for a no-data-stored case (no destination).
- o (f) stands for fill and (n) stands for no-fill, if the use of fill makes a difference.

Table 1: Blitter Memory Bus Activity

	Operation Type	Memory Cycle Data Fetch or Data Store Usage											
1.	A B C ---D	A0	B0	C0		A1	B1	C1	D0	A2	B2	C2	D1	D2
2.	A B C ---Z	A0	B0	C0	A1	B1	C1	A2	B2	C2				
3.	A B C*---D(f)	A0	B0			A1	B1	D0		A2	B2	D1		D2
4.	A B C*---D(n)	A0	B0	A1	B1	D0		A2	B2	D1		D2		
5.	A B C*---Z	A0	B0		A1	B1		A2	B2					
6.	A C ---D	A0	C0		A1	C1	D0	A2	C2	D1		D2		
7.	A C ---Z	A0	C0	A1	C1	A2	C2							
8.	A C* ---D(f)	A0			A1	D0	A2	D1		D2				
9.	A C* ---D(n)	A0		A1	D0	A2	D1		D2					
10.	A C* ---Z	A0		A1		A2								
11.	A* B C ---D		B0	C0			B1	C1	D0		B2	C2	D1	D2
12.	A* B C ---Z		B0	C0		B1	C1		B2	C2				
13.	A* B C* ---D(f)		B0				B1	D0			B2	D1		D2
14.	A* B C* ---D(n)		B0			B1	D0		B2	D1		D2		
15.	A* B C* ---Z		B0			B1			B2					
16.	A* C ---D		C0			C1	D0		C2	D1		D2		
17.	A* C ---Z		C0		C1		C2							
18.	A* C* ---D(f)					D0			D1			D2		
19.	A* C* ---D(n)				D0		D1		D2					
20.	A* C* ---Z													
21.	LINE DRAWING MODE		C0	*	D0		C1	*	D1		C2	*	D2	

Notes for Table 1:

a. Function 9 is a block move only when both shift values are 0. Use function 14 if either shift value is nonzero.

b. Function 10 is a zero test; function 19 is a constant fill.

c. Function 21 is for line drawing. The cycles marked with * are not used by the blitter but are not released to the system. Sorry.

d. Functions 11, 13, 14, and 16 have special restraints when source and data overlap. During some priority overrides, the order of data fetch and store will be reversed. When using these functions with overlapping source and data, the overlap must be greater than two words. Otherwise, you risk overwriting your source data.

APPENDIX F - Amiga Hardware Manual

This appendix contains information about the 8520 peripheral interface adapters.

****

QUICK REFERENCE

BRIEF ADDRESS MAP FOR 8520'S

The system hardware selects the 8520's (aka CIA's) when the upper three address bits are 101.

Furthermore, CIAA is selected when A12 is low, A13 high; CIAB is selected when A12 is high, A13 low.

You can use either byte or word addresses to access the 8520's. For byte access (seems to be the usual case), A0 must be 0 for CIAA, 1 for CIAB. For word access, CIAB communicates on data bits 15~8; CIAA communicates on data bits 7~0. (A0 is always 0 for word access, naturally).

Address bits A11, A10, A9, and A8 are used to specify which of the 16 internal registers you want to access. This is indicated by "r" in the address.

All other bits are don't cares. So, CIAA is selected by the following binary address: 101x xxxx xx01 rrrr xxxx xxx0.
CIAB address: 101x xxxx xx10 rrrr xxxx xxx1

With future expansion in mind, we have decided on the following addresses: CIAA = BFEr01; CIAB = BFDr00.

CIAB Address Map

Byte Address	Register Name	Data bits 7	6	5	4	3	2	1	0
BFD000		/DTR	/RTS	/CD	/CTS	/DSR	SEL	POUT	BUSY
BFD100		/MTR	/SEL3	/SEL2	/SEL1	/SEL0	/SIDE	DIR	/STEP
BFD200		ddr for port A (BFD000); 1 = output (set to 0xC0)							
BFD300		ddr for port B (BFD100); 1 = output (set to 0xFF)							
BFD400		CIAB Timer A low byte							
BFD500		CIAB Timer A high byte							
BFD600		CIAB Timer B low byte							
BFD700		CIAB Timer B high byte							
BFD800		Horizontal sync event counter bits 7~0							
BFD900		Horizontal sync event counter bits 15~8							
BFDA00		Horizontal sync event counter bits 23~16							
BFDB00		not used							
BFDC00		CIAB Serial data register							
BFDD00		CIAB Interrupt control register							
BFDE00		CIAB Control register A							
BFDF00		CIAB Control register B							

Note: CIAB can generate INT6.

CIAA Address Map

Byte Address	Register Name	Data bits 7	6	5	4	3	2	1	0
BFE001		/FIR1	/FIR0	/RDY	/TK0	/WPRO	/CHNG	/LED	OVL
BFE101		Parallel port							
BFE201		ddr for port A (BFE001); 1 = output (Set to 0x03)							
BFE301		ddr for port B (BFE101); 1 = output (Can be in or out)							
BFE401		CIAA Timer A low byte							
BFE501		CIAA Timer A high byte							
BFE601		CIAA Timer B low byte							
BFE701		CIAA Timer B high byte							
BFE801		60 Hz event counter bits 7~0							
BFE901		60 Hz event counter bits 15~8							
BFEA01		60 Hz event counter bits 23~16							
BFEB01		not used							
BFEC01		CIAA Serial data register (keyboard)							
BFED01		CIAA Interrupt control register							
BFEE01		CIAA Control register A							
BFEF01		CIAA Control register B							

Note: CIAA can generate INT2.

****

INTERFACE SIGNALS

Clock Input

The 02 clock is a TTL compatible input used for internal device operation and as a timing reference for communicating with the system data bus.

CS - Chip Select Input

The CS input controls the activity of the 8520. A low level on CS while 02 is high causes the device to respond to signals on the R/W and address (RS) lines. A high on CS prevents these lines from controlling the 8520. The CS line is normally activated (low) at 02 by the appropriate address combination.

R/W - Read/Write Input

The R/W signal is normally supplied by the microprocessor and controls the direction of data transfers of the 8520. A high on R/W indicates a read (data transfer out of the 8520), while a low indicates a write (data transfer into the 8520).

RS3-RS0 - Address Inputs

The address inputs select the internal registers as described by the Register Map.

DB7-DB0 - Data Bus Inputs/Outputs

The eight data bus output pins transfer information between the 8520 and the system data bus. These pins are high impedance inputs unless CS is low and R/W and 02 are high, to read the device. During this read, the data bus output buffers are enabled, driving the data from the selected register onto the system data bus.

IRQ - Interrupt Request Output

IRQ is an open drain output normally connected to the processor interrupt input. An external pull-up resistor holds the signal high, allowing multiple IRQ outputs to be connected together. The IRQ output is normally off (high impedance) and is activated low as indicated in the functional description.

RES - Reset Input

A low on the RES pin resets all internal registers. The port pins are set as inputs and port registers to zero (although a read of the ports will return all highs because of passive pull-ups). the timer control registers are set to zero and the timer latches to all ones. All other registers are reset to zero.

REGISTER MAP

Each 8520 has 16 registers which you may read or write. Here is the list of registers and the access address of each within the memory space dedicated to the 8520:

RS3	RS2	RS1	RS0	Register #(hex)	NAME	MEANING
0	0	0	0	0	PRA	Peripheral Data Register A
0	0	0	1	1	PRB	Peripheral Data Register B
0	0	1	0	2	DDRA	Data Direction Register A
0	0	1	1	3	DDRB	Direction Register B
0	1	0	0	4	TALO	Timer A Low register
0	1	0	1	5	TAHI	Timer A High register
0	1	1	0	6	TBLO	Timer B Low register
0	1	1	1	7	TBHI	Timer B High register
1	0	0	0	8		Event LSB
1	0	0	1	9		Event 8-15
1	0	1	0	A		Event MSB
1	0	1	1	B		No Connect

1	1	0	0	C	SDR	Serial Data Register
1	1	0	1	D	ICR	Interrupt Control Register
1	1	1	0	E	CRA	Control Register A
1	1	1	1	F	CRB	Control Register B

SOFTWARE NOTE:

The operating system kernel has already allocated the use of all 4 of the timers TA and TB in the 8520's. If you are running under control of the system exec, be aware of the following allocation of system resources:

8520A, timer A -- Commodore serial communications (if no serial comm happening, timer becomes available).

8520A, timer B -- Video beam follower (used when synchronizing the blitter device to the video beam, see the description of QBSBlit in the system software manual). If no beam-sync'ed blits are in process, this timer will be available.

8520B, timer A -- Keyboard (used continuously, whenever system EXEC is in control).

8520B, timer B -- Virtual timer device (used continuously, whenever system EXEC is in control, used for task switching/interrupts).

REGISTER NAMES

The names of the registers within the 8520's are as follows. The address at which each is to be accessed is given here in this list.

Address for:			
8520-A	8520-B	NAME	EXPLANATION
			(write)/(read mode)
BFE001	BFD000	PRA	Peripheral Data Register A
BFE101	BFD100	PRB	Peripheral Data Register B
BFE201	BFD200	DDRB	Data Direction Register "A"
BFE301	BFD300	DDRA	Data Direction Register "B"
BFE401	BFD400	TALO	TIMER A Low Register
BFE501	BFD500	TAHI	TIMER A High Register
BFE601	BFD600	TBLO	TIMER B Low Register
BFE701	BFD700	TBHI	TIMER B High Register
BFE801	BFD800		Event LSB
BFE901	BFD900		Event 8 - 15

BFEA01	BFDA00		Event MSB
BFEB01	BFDB00		No connect
BFEC01	BFDC00	SDR	Serial Data Register
BFED01	BFDD00	ICR	Interrupt Control Register
BFEE01	BFDE00	CRA	Control Register A
BFEF01	BFDF00	CRB	Control Register B

REGISTER FUNCTIONAL DESCRIPTION:

I/O PORTS (PRA, PRB, DDRA, DDRB)

Ports A and B each consist of an 8-bit Peripheral Data Register (PR) and an 8-bit data direction register (DDR). If a bit in the DDR is set to a 1, the corresponding bit position in the PR becomes an output. If a DDR bit is set to a 0, the corresponding PR bit is defined as an input.

When you READ a PR register, you read the actual current state of the I/O pins (PA0-PA7, PB0-PB7, regardless of whether you have set them to be inputs or outputs.

Ports A and B have passive pull-up devices as well as active pull-ups, providing both CMOS and TTL compatibility. Both ports have two TTL load drive capability.

In addition to their normal I/O operations, ports PB6 and PB7 also provide timer output functions.

HANDSHAKING

Handshaking occurs on data transfers using the PC output pin and the FLAG input pin. PC will go low on the third cycle after a Port B access. This signal can be used to indicate "data ready" at PORT B or "data accepted" from PORT B. Handshaking on 16-bit data transfers (using both ports A and B) is possible by always reading or writing PORT A first. FLAG is a negative edge sensitive input which can be used for receiving the PC output from another 8520, or as a general purpose interrupt input. Any negative transition on FLAG will set the FLAG interrupt bit.

REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0
0	PRA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
1	PRB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
2	DDRA	DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0
3	DDRB	DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0

INTERVAL TIMERS (TIMER A, TIMER B)

Each interval timer consists of a 16-bit read-only Timer Counter and a 16-bit write-only Timer Latch. Data written to the timer is latched into the Timer Latch, while data read from the timer

is the present contents of the Timer Counter.

The latch is also called a prescalar in that it represents the countdown value which must be counted before the timer reaches an underflow (no more counts) condition. This latch (prescalar) value is a divider of the input clocking frequency.

The timers can be used independently, or linked for extended operations. Various timer operating modes allow generation of long time delays, variable width pulses, pulse trains, and variable frequency waveforms. Utilizing the CNT input, the timers can count external pulses or measure frequency, pulse width, and delay times of external signals.

Each timer has an associated control register, providing independent control over each of the following functions:

START/STOP

A control bit allows the timer to be started or stopped by the microprocessor at any time.

PB On/Off

A control bit allows the timer output to appear on a PORT B output line (PB6 for timer A and PB7 for timer B). This function over-rides the DDRB control bit and forces the appropriate PB line to become an output.

Toggle/Pulse

A control bit selects the output applied to PORT B while the PB On/Off bit is ON. On every timer underflow, the output can either toggle or generate a single positive pulse of one cycle duration. The Toggle output is set high whenever the timer is started, and set low by RES.

One-Shot/Continuous

A control bit selects either timer mode. In one-shot mode, the timer will count down from the latched value to zero, generate an interrupt, reload the latched value, then stop. In continuous mode, the timer will count down from the latched value to zero, generate an interrupt, reload the latched value, and repeat the procedure continuously.

In one-shot mode, a write to Timer High (register 5 for Timer A, register 7 for Timer B) will transfer the timer latch to the counter and initiate counting regardless of the start bit.

Force Load

A strobe bit allows the timer latch to be loaded into the timer counter at any time, whether the timer is running or not.

INPUT MODES

Control bits allow selection of the clock used to decrement the timer. TIMER A can count 02 clock pulses or external pulses applied to the CNT pin. TIMER B can count 02 pulses, external CNT pulses, TIMER A underflow pulses, or TIMER A underflow pulses while the CNT pin is held high.

The timer latch is loaded into the timer on any timer underflow, on a force load, or following a write to the high byte of the pre-scalar while the timer is stopped. If the timer is running, a write to the high byte will load the timer latch, but not reload the counter.

BIT NAMES on READ-register

REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0
4	TALO	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0
5	TAHI	TAH7	TAH6	TAH5	TAH4	TAH3	TAH2	TAH1	TAH0
6	TBLO	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0
7	TBHI	TBH7	TBH6	TBH5	TBH4	TBH3	TBH2	TBH1	TBH0

BIT NAMES on WRITE-register

REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0
4	TALO	PAL7	PAL6	PAL5	PAL4	PAL3	PAL2	PAL1	PAL0
5	TAHI	PAH7	PAH6	PAH5	PAH4	PAH3	PAH2	PAH1	PAH0
6	TBLO	PBL7	PBL6	PBL5	PBL4	PBL3	PBL2	PBL1	PBL0
7	TBHI	PBH7	PBH6	PBH5	PBH4	PBH3	PBH2	PBH1	PBH0

TIME OF DAY CLOCK

TOD consists of a 24-bit binary counter. Positive edge transitions on this pin cause the binary counter to increment. The TOD pin has a passive pull-up on it.

A programmable ALARM is provided for generating an interrupt at a desired time. The ALARM registers are located at the same addresses as the corresponding TOD registers. Access to the ALARM is governed by a Control Register bit. The ALARM is write-only; any read of a TOD address will read time regardless of the state of the ALARM access bit.

A specific sequence of events must be followed for proper setting and reading of TOD. TOD is automatically stopped whenever a write to the register occurs. The clock will not start again until after a write to the LSB Event Register. This assures that TOD will always start at the desired time.

Since a carry from one stage to the next can occur at any time with respect to a read operation, a latching function is included to keep all Time Of Day information constant during a read sequence. All TOD registers latch on a read of MSB Event and remain latched until after a read of LSB Event. The TOD clock continues to count when the output registers are latched. If only one register is to be read, there is no carry problem and the register can be read "on the fly" provided that any read of MSB Event is followed by a read of LSB Event to disable the latching.

BIT NAMES for WRITE TIME/ALARM or READ TIME

REG	NAME								
8	LSB Event	E7	E6	E5	E4	E3	E2	E1	E0
9	Event 8-15	E15	E14	E13	E12	E11	E10	E9	E8
A	MSB Event	E23	E22	E21	E20	E19	E18	E17	E16

WRITE
CRB7 = 0
CRB7 = 1 ALARM

SERIAL PORT (SDR)

The serial port is a buffered, 8-bit synchronous shift register. A control bit selects input or output mode.

INPUT MODE

In input mode, data on the SP pin is shifted into the shift register on the rising edge of the signal applied to the CNT pin. After 8 CNT pulses, the data in the shift register is dumped into the Serial Data Register and an interrupt is generated.

OUTPUT MODE

In the output mode, TIMER A is used as the baud rate generator. Data is shifted out on the SP pin at 1/2 the underflow rate of TIMER A. The maximum baud rate possible is 02 divided by 4, but the maximum usable baud rate will be determined by line loading and the speed at

which the receiver responds to input data.

To begin transmission, you must first set up TIMER A in continuous mode, and start the timer. Transmission will start following a write to the Serial Data Register. The clock signal derived from TIMER A appears as an output on the CNT pin. The data in the Serial Data Register will be loaded into the shift register, then shifted out to the SP pin when a CNT pulse occurs. Data shifted out becomes valid on the next falling edge of CNT and remains valid until the next falling edge.

After 8 CNT pulses, an interrupt is generated to indicate that more data can be sent. If the Serial Data Register was reloaded with new information prior to this interrupt, the new data will automatically be loaded into the shift register and transmission will continue.

If no further data is to be transmitted after the 8th CNT pulse, CNT will return high and SP will remain at the level of the last data bit transmitted.

SDR data is shifted out MSB first. Serial input data should appear in this same format.

BIDIRECTIONAL FEATURE

The bidirectional capability of the Serial Port and CNT clock allows many 8520's to be connected to a common serial communications bus on which one 8520 acts as a master, sourcing data and shift clock, while all other 8520 chips act as slaves. Both CNT and SP outputs are open drain to allow such a common bus. Protocol for master/slave selection can be transmitted over the serial bus or via dedicated handshake lines.

REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0
C	SDR	S7	S6	S5	S4	S3	S2	S1	S0

INTERRUPT CONTROL REGISTER (ICR)

There are 5 sources of interrupts on the 8520:

- -underflow from TIMER A (timer counts down past 0)
- -underflow from TIMER B
- -TOD ALARM
- -Serial Port Full/Empty
- -FLAG

A single register provides masking and interrupt information. The Interrupt Control Register consists of a write-only MASK register and a read-only DATA register. Any interrupt will set the corresponding bit in the DATA register. Any interrupt that is enabled by a 1-bit in that position in the MASK will set the IR bit (MSB) of the DATA register,

and bring the IRQ pin low. In a multi-chip system, the IR bit can be polled to detect which chip has generated an interrupt request.

When you read the DATA register, its contents are cleared (set to 0) and the IRQ line returns to a high state. Since it is cleared on a read, you must assure that your interrupt polling or interrupt service code can preserve and respond to all bits which may have been set in the DATA register at the time it was read. With proper preservation and response, it is easily possible to intermix polled and direct interrupt service methods.

You can set or clear one or more bits of the MASK register without affecting the current state of any of the other bits in the register. This is done by setting the appropriate state of the MSBit, which is called the SET/CLEAR bit. In bits 6-0, you yourself form a mask which specifies which of the bits you wish to affect. Then, using bit 7, you specify HOW the bits in corresponding positions in the mask are to be affected.

If bit 7 is a 1, then any bit 6-0 in your own mask word which is set to a 1 SETS the corresponding bit in the MASK register. Any bit which you have set to a 0 causes the MASK register bit to remain in its current state.

If bit 7 is a 0, then any bit 6-0 in your own mask word which is set to a 1 CLEARS the corresponding bit in the MASK register. Again, any 0 bit in your own mask word causes no change in the contents of the corresponding MASK register bit.

If an interrupt is to occur based on a particular condition, then that corresponding MASK bit must be a 1.

Example: Suppose you want to SET the TIMER A interrupt bit (enable the TIMER A interrupt), but want to be sure that all other interrupts are CLEARED. Here is the sequence you can use:

```
movi.b  01111110B,A0
mov.b   A0,ICR      ;MSB is 0, means clear
                    ;any bit whose value is
                    ;1 in the rest of the byte
movi.b  10000001B,A0
mov.b   A0,ICR      ;MSB is 1, means set
                    ;any bit whose value is
                    ;1 in the rest of the byte
                    ;(do not change any values
                    ; wherein the written value
                    ; bit is a zero)
```

Read Interrupt Control Register:

REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0

D	ICR	IR	0	0	FLG	SP	ALRM	TB	TA

Write Interrupt Control MASK:

REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0
D	ICR	S/C	x	x	FLG	SP	ALRM	TB	TA

CONTROL REGISTERS

There are two control registers in the 8520, CRA and CRB. CRA is associated with TIMER A and CRB is associated with TIMER B. The format of the registers is as follows:

CONTROL REGISTER A:

BIT	NAME	FUNCTION
0	START	1 = start TIMER A, 0 = stop TIMER A. This bit is automatically reset (= 0) when underflow occurs during one-shot mode.
1	PBON	1 = TIMER A output on PB6, 0 = PB6 is normal operation.
2	OUTMODE	1 = TOGGLE, 0 = PULSE.
3	RUNMODE	1 = one-shot mode, 0 = continuous mode.
4	LOAD	1 = FORCE LOAD (this is a STROBE input, there is no data storage; bit 4 will always read back a zero and writing a 0 has no effect.)
5	INMODE	1 = TIMER A counts positive CNT transitions, 0 = TIMER A counts 02 pulses.
6	SPMODE	1 = SERIAL PORT=output (CNT is the source of the shift clock) 0 = SERIAL PORT=input (external shift clock is required)

BIT MAP OF REGISTER CRA:

REG #	NAME	TOD IN	SPMODE	INMODE	LOAD	RUNMODE	OUTMODE	PBON	START
E	CRA	0=60Hz 1=50Hz	0=input 1=output	0=02 1=CNT	1=FORCE LOAD	0=cont. 1=one-	0=pulse 1=toggle	0=PB6OFF 1=PB6ON	0=stop 1=start

(STROBE) shot

|<-------- TIMER A Variables ------------------->|

All unused register bits are unaffected by a write and forced to 0 on a read.

CONTROL REGISTER B:

BIT	NAME	FUNCTION
0	START	1 = start TIMER B, 0 = stop TIMER B. This bit is automatically reset (= 0) when underflow occurs during one-shot mode.
1	PBON	1 = TIMER B output on PB7, 0 = PB7 is normal operation.
2	OUTMODE	1 = TOGGLE, 0 = PULSE.
3	RUNMODE	1 = one-shot mode, 0 = continuous mode.
4	LOAD	1 = FORCE LOAD (this is a STROBE input, there is no data storage; bit 4 will always read back a zero and writing a 0 has no effect.)

6,5 INMODE Bits CRB6 and CRB5 select one of four possible input modes for TIMER B, as follows:

CRB6	CRB5	Mode Selected
0	0	TIMER B counts 02 pulses
0	1	TIMER B counts positive CNT transitions
1	0	TIMER B counts TIMER A underflow pulses
1	1	TIMER B counts TIMER A underflow pulses while CNT pin is held high.

7 ALARM 1 = writing to TOD registers sets ALARM,
0 = writing to TOD registers sets TOD clock.
Reading TOD registers always reads TOD clock, regardless of the state of the ALARM bit.

BIT MAP OF REGISTER CRB:

REG #	NAME	ALARM	INMODE	LOAD	RUNMODE	OUTMODE	PBON	START
F	CRB	0=TOD 1=ALARM	00=02 01=CNT 10=TIMER A 11=CNT+TIMER A	1=FORCE LOAD (STROBE)	0=cont. 1=one-shot	0=pulse 1=toggle	0=PB7OFF 1=PB7ON	0=stop 1=start

|<----------------TIMER B Variables---------------------->|

All unused register bits are unaffected by a write and forced to 0 on a read.

PORT SIGNAL ASSIGNMENTS

This part specifies how various signals relate to the available ports of the 8520. This information enables the programmer to relate the port addresses to the outside-world items (or internal control signals) which are to be affected. This part is primarily for the use of the systems programmer and should generally not be utilized by applications programmers. Systems software normally is configured to handle the setting of particular signals, no matter how the physical connections may change. In other words, if you have a version of the system software that matches the rev. level of the machine (normally a true condition), when you ask that a particular bit be set, you don't care which port that bit is connected to. Thus applications programmers should rely on system documentation rather than going directly to the ports. Note also that in this, a multi-tasking operating system, many different tasks may be competing for the use of the system resources. Applications programmers should follow the established rules for resource access in order to assure compatibility of their software with the system.

Address BFERFF data bits 7-0 (A12*) (int2)

```
PA7..game port 1, pin 6 (fire button*)
PA6..game port 0, pin 6 (fire button*)
PA5..RDY*       disk ready*
PA4..TK0*       disk track 00*
PA3..WPRO*      write protect*
PA2..CHNG*      disk change*
PA1..LED*       led light (0=bright)
PA0..OVL        memory overlay bit

SP...KDAT       keyboard data
CNT..KCLK

PB7..P7         data 7
PB6..P6         data 6
PB5..P5         data 5     Centronics parallel interface
PB4..P4         data 4          data
PB3..P3         data 3
PB2..P2         data 2
PB1..P1         data 1
PB0..P0         data 0

PC...drdy*              centronics control
F....ack*
```

Address BFDRFE data bits 15-8 (A13*) (int6)

```
PA7..com line DTR*, driven output
```

```
PA6..com line RTS*, driven output
PA5..com line carrier detect*
PA4..com line CTS*
PA3..com line DSR*
PA2..SEL         centronics control
PA1..POUT        paper out ---+
PA0..BUSY        busy    ---+ |
                             | |
SP...BUSY        commodore -+ |
CNT..POUT        commodore ---+

PB7..MTR*        motor
PB6..SEL3*       select external 3rd drive
PB5..SEL2*       select external 2nd drive
PB4..SEL1*       select external 1st drive
PB3..SEL0*       select internal drive
PB2..SIDE*       side select*
PB1..DIR         direction
PB0..STEP*       step*

PC...not used
F....INDEX*      disk index*
.fi
```

AMIGA AUTO-CONFIGURATION ARCHITECTURE

This document is is process. Please contact Commodore-Amiga for the latest available information on this topic.