mach64
Accelerator
Programmer's Guide

SECOND DRAFT

Information in this document is
proprietary and **confidential.**

Technical Reference Manuals

P/N: PRG888GX0-01

ATI Technologies Inc.
33 Commerce Valley Drive East
Thornhill, Ontario
Canada L3T 7N6

User Support: 905-882-2626
User Support Fax: 905-882-0546
Offices: 905-882-2600
Fax: 905-882-2620
BBS: 905-764-9404

P/N: PRG888GX0-01

PRELIMINARY Release 1

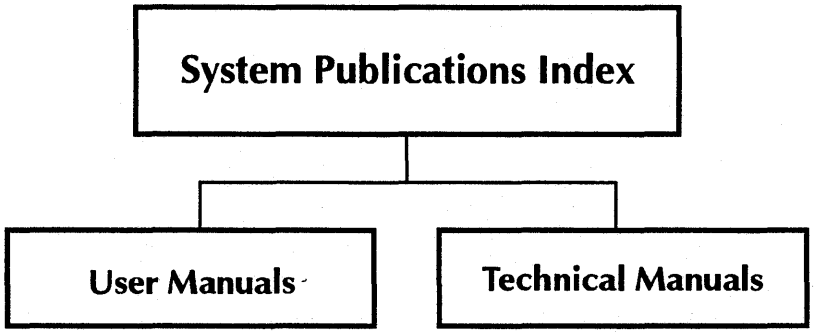
© Copyright 1993, 1994

ATI Technologies, Inc.

The information contained in this document has been carefully checked and is believed to be entirely reliable. No responsibility is assumed for inaccuracies. ATI reserves the right to make changes at any time to improve design and supply the best product possible.

All rights reserved. This document is subject to change without notice and is not to be reproduced or distributed in any form or by any means without prior permission in writing from ATI Technologies Inc.

ATI, ***VGAWonder, mach8, mach32, mach64,*** **8514ULTRA, GRAPHICS ULTRA, GRAPHICS VANTAGE, GRAPHICS ULTRA+,** and **GRAPHICS ULTRA PRO** are trademarks of ATI Technologies Inc. All other trademarks and product names are properties of their respective owners.



- TBA
- TBA
- TBA

- *mach64* Graphics Controller Specifications (GCS888GX-01)
- *mach64* Accelerator Programmer's Guide (PRG888GX0-01
- *mach64* VGA Programmer's Guide (VGA888GX0-01)
- *mach64* Accelerator Register Reference (EXT888GX0-01)

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Record of Revisions

Release	Date	Description of Changes
1.0	93 Dec	PRELIMINARY

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Contents

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Chapter 1
Overview

INTRODUCTION

This manual is a guide to understanding and programming the *mach64* accelerator. The *mach64* accelerator is a fixed-function, 2D graphics acelerator. It is function-compatible but not register-compatibl with the *mach32*.

Those seeking a general understanding of the features and functions of the *mach64*, or experienced mach64 programmers seeking a function summary, need only read *Chapter 2, Programming Model.*

Very specific examples and techniques are described in *Chapter 3, Simple Draw operations, Chapter 4, Advanced Topics I,* and *Chapter 5, Advanced Topics II.*

Major Features

- Full draw capability at 1, 4, 8, 15, 16, and 32 bit per pixel color resolutions. Hardware-assisted draw functions available for packed 24 bit per pixel draw modes.
- Spatial resolution of 640x480, 800x600, 1024x768, and 1280x1024.
- Full read/writeable memory-mapped registers.
- Up to 8M of memory.
- 16x32 command FIFO.
- Four-color (two fixed colors, complement, and transparent) hardware cursor of size up to 64x64.
- Overscan.
- Linear frame buffer locatable on 4M boundaries anywhere in a 4G system memory address space, and sizeable to 4M or 8M.
- Paged frame buffer with two, 32k pages, pageable on 32k boundaries anywhere in the 8M video memory address space.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Major Features (cont'd)

- Draw functions include rectangle fill, line draw, bitblt, polygon boundary lines, polygon fill.
- Generalized 2D patterns with rotation.
- A linear memory mode for efficient memory management.
- Efficient monochrome expansion.
- Bit masking and scissoring capabilities.
- Seventeen-function ALU for ful suite of logical ROPs.
- Source compare logic suitable for alpha channel mixing.
- Draw engine context loads for fast context switching.
- Context chaining for grouping complex task lists.
- Scrolling and panning on a virtual desktop.
- EEPROM hardware support for non-volatile storage.
- Four-level hardware Display Power Management System (DPMS) mode support.
- DAC power-down support.
- Diagnostic test modes.

Functional Enhancements to mach32

- Full draw capability in 1bpp and 32bpp modes, and hardware assist in packed 24bpp mode have been added.
- Full 32-bit registers. Some register pairs may be written in a single 32-bit write.
- Device coordinates have been expanded to -4096 to +4095 in the X direction, and -16384 to +16383 in the Y direction.
- Bresenham parameters have been expanded from 12 bits to 18 bits.
- Packed monochrome expansion.
- The paged frame buffer is now pageable on 32k boundaries instead of 64k.
- The source trajectory types, strictly-linear, general-pattern, and general-pattern-with-rotation, have been added.
- Source compare.
- Contexts and context chaining.
- Four-level hardware Display Power Management System (DPMS) mode support.
- DAC power-down support.
- Diagnostic test modes.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Deletions From mach32

- Point-to-point line draw
- Line clip exception handling
- VNIB and VPIX type rectangles
- Short-stroke vectors
- Scan line draw
- Four compare functions
- Bounds accumulators
- CRTC shadow sets
- Host reads; screen-to-host transfers can still be accomplished by aperture reads
- Degree mode lines; Bresenham lines are still supported.

All the deleted functions listed above are redundant, and may still be accomplished by other means.

Functional Differences From mach32

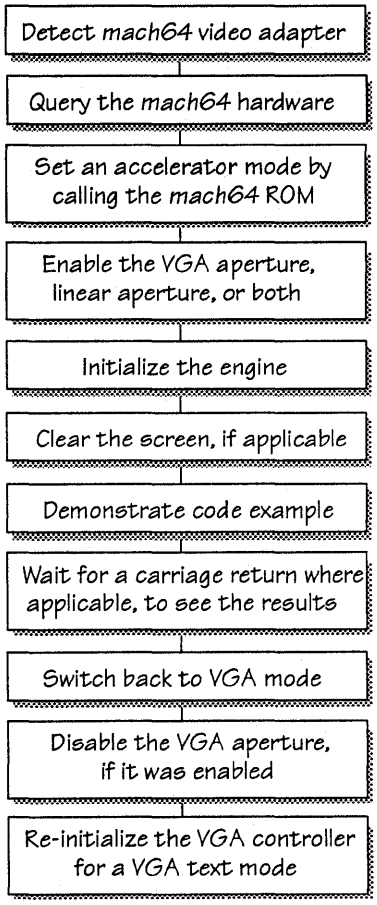
- Monochrome blits are now packed instead of sparse.
- Host writes are packed to 32 bits. 1bpp and 4bpp modes may be optionally aligned to a byte.
- Pixel consumption order from the host data register is only programmable in 1bpp and 4bpp modes.
- Polygons are always inclusive on both edges.
- Polygons derive their boundary data from an implicit polygon source instead of an explicit monochrome source.
- Rectangular trajectories are specified in width and height instead of start and end.
- The ALU carry chain mask is set explicitly instead of implicitly from the pixel depth.
- Line drawing options do not affect rectangular trajectories and rectangle options do not affect line drawing trajectories.
- Destination side effects (tiling) are now programmable.
- Source pointer always returns to the original SRC_X, SRC_Y position after draw completion.
- Pixel depths, pitches, and offsets are independently specified for CRTC, source, destination, and host.
- Bresenham parameters have been expanded from 12 bits to 18 bits.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SAMPLE CODE ORGANIZATION

C source code and assembler source code

All C source sample code is compatible with Microsoft's C compilers (version 5.1 and newer). All assembler source sample code is compatible with Microsoft's MASM compiler version 5.1. Each sample code example has its own program, based on the following template:



Constants

All constants used in the printed sample code are included in *Appendix F, Sample Code*, under the following header file names:

ATIM64.INC
ATIM64.H
SAMPLE.H.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Utility functions

Any utility functions not included with the printed sample code are listed in *Appendix F, Sample Code,* under the following source file names:

ATTR.C
DRAW.C
GLOB.C
HWCURSOR.C
INIT.C
MEMREG.C
PALETTE.C
WAIT.C
MOVEMEM.ASM
ROMCALLS.ASM

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Chapter 2
Programming Model

INTRODUCTION

The *mach64* has two, distinct operating modes:

- **VGA mode**
- **Accelerator mode.**

For more information on VGA programming, see the *mach64 VGA Programmer's Guide*. The accelerator provides the ability to draw into screen memory concurrently with the operation of the host CPU.

mach64 DETECTION

To detect the presence of a mach64:

1. Check for the ATI product signature.
2. Read SCRATCH_REG0 and save its contents.
3. Write the value 0x55555555 to SCRATCH_REG0.
4. Read back SCRATCH_REG0. If the value is not 0x55555555, a *mach64* is not present.
5. Repeat steps 3 and 4, using the value 0xAAAAAAAA.
6. Restore the saved value of SCRATCH_REG0.
7. Read the CHIP_ID register for additional information on chip type, class, and revision.

Additional configuration information can be obtained with a BIOS query call (functions 6, 7, 8, 9, and 0xA). See *Appendix A, BIOS Services*, for more information.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Sample code

This is sample code that detects if a MACH 64 based video adapter is installed. See INIT.C and ROMCALLS.ASM for more details.

```
/* ------------------------------------------------------------------------
  detect_mach64 - determine if a mach64 based video adapter is installed.

  This routine identifies if a mach64 based video adapter is installed. This
  is done by writing and reading the SCRATCH_REG0 or SCRATCH_REG1 io based
  registers. These registers must be saved and restored since the mach64
  BIOS uses their contents.

  Returns YES_MACH64 if detected, NO_MACH64 if not.
------------------------------------------------------------------------ */
int detect_mach64(void)
{
    unsigned long save_value;
    int result;

    // assume failure
    result = NO_MACH64;

    // check for ATI rom signature
    if (is_ati_rom() == 0)
    {
        return (NO_MACH64);
    }

    // save old value
    save_value = ior(ioSCRATCH_REG0);

    // test odd bits for readability
    iow(ioSCRATCH_REG0, 0x55555555);
    if (ior(ioSCRATCH_REG0) == 0x55555555)
    {
        // test even bits for readability
        iow(ioSCRATCH_REG0, 0xaaaaaaaa);
        if (ior(ioSCRATCH_REG0) == 0xaaaaaaaa)
        {
            result = YES_MACH64;
        }
    }

    // restore old value
    iow(ioSCRATCH_REG0, save_value);

    return (result);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

MODE SWITCHING

It is highly recommended that all mode switching be done by a BIOS service function call, rather than by manually setting the CRTC. The main reasons for doing this are:

- Simplicity
- The characteristics of the non-volatile storage device that stores mode and monitor information may not be known. Without monitor information, the only mode guaranteed to work on all analog monitors is 640x480 60 Hz non-interlaced.
- CRTC compatibility with future devices is not guaranteed.

CRT controllers

Note that there are **two logical CRT controllers** (CRTCs) on the *mach64*. CRT parameters may be set independently on the VGA and accelerator CRTCs, and flipped from one to the other with a simple BIOS call. Screen memory may be shared to give one or the other device more screen memory, or it may be divided into two areas so that both logical devices may operate concurrently without having to save and restore screen data.

The default memory boundary setting is set from the install program, but may be overridden by the application program by setting the appropriate bits in the MEM_CNTL register. In addition, some configurations of the *mach64* may have the VGA disabled.

BIOS Interface

VGA modes are initialized with the standard INT 10h interface, as described in the *mach64 VGA Programmer's Guide*. See *Appendix A, BIOS Services*, for a complete definition of all BIOS services.

To set the accelerator mode using the BIOS:

1. Calculate the BIOS segment by reading SCRATCH_REG1 and computing:
   segment = (SCRATCH_REG1 & 0x7F) * 0x80 + 0xC000
2. Call function 2 (load accelerator CRT parameters and set display mode) of the BIOS.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Sample code BMODE.DOC

```
This is sample code to set an accelerator mode using the Mach 64 ROM
services. See ROMCALLS.ASM for more details.


; Data section

rom_addr    dw      64h
            dw      0c000h
            db      0

----

; Code section

PARM        equ     6    ; passed parameters start at bp+6 for large model

; ---------------------------------------------------------------------------
; LOAD_AND_SET_MODE
;
; Load accelerator mode parameters and set accelerator mode. Input parameters
; are fetched from the stack.
;
; Inputs : WORD resolution code
;            CH = 12h - 640x480
;            CH = 6Ah - 800x600
;            CH = 55h - 1024x768
;            CH = 83h - 1280x1024
;
;          WORD pitch code
;            CL [bits 7-6] = 0 - 1024
;            CL [bits 7-6] = 1 - don't change
;            CL [bits 7-6] = 2 - pitch size = resolution width
;
;          WORD deep color code
;            CL [bits 3-0] = 1 - 4 bpp
;            CL [bits 3-0] = 2 - 8 bpp
;            CL [bits 3-0] = 3 - 15 bpp (555)
;            CL [bits 3-0] = 4 - 16 bpp (565)
;            CL [bits 3-0] = 5 - 24 bpp
;            CL [bits 3-0] = 6 - 32 bpp
;
; Outputs: Returns error code in ax
;            AX = 0 - no error
;            AX = 1 - function complete with error
;            AX = 2 - function not supported
;
; ---------------------------------------------------------------------------
            public  load_and_set_mode
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
load_and_set_mode proc far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    cx

            ; setup parameters for call to ATI rom
            call    rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     ax, WORD PTR [bp+PARM]      ; get resolution code
            mov     ch, al
            mov     ax, WORD PTR [bp+PARM+2]    ; get pitch code
            shl     al, 6
            mov     cl, al
            mov     ax, WORD PTR [bp+PARM+4]    ; get deep color code
            and     al, 7
            or      cl, al
            mov     ax, 2                       ; function code 2
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; setup error code in AL
            mov     al, ah
            xor     ah, ah

            ; restore saved registers
            pop     cx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

load_and_set_mode endp

; ---------------------------------------------------------------------------
; SET_DISPLAY_MODE
;
; Set display to accelerator (after a call to LOAD_MODE_PARMS()) or VGA mode.
; Input parameters are fetched from the stack.
;
; Inputs : WORD display mode
;            CL = 0 - VGA
;            CL = 1 - Accelerator
;
; Outputs: Returns error code in ax
;            AX = 0 - no error
;            AX = 1 - function complete with error
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
;            AX = 2 - function not supported
;
; ---------------------------------------------------------------------------
            public  set_display_mode

set_display_mode proc far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    cx

            ; setup parameters for call to ATI rom
            call    rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     ax, WORD PTR [bp+PARM]      ; get display mode flag
            mov     cl, al
            mov     ax, 1                       ; function code 1
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; setup error code in AL
            mov     al, ah
            xor     ah, ah

            ; restore saved registers
            pop     cx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

set_display_mode endp

; ---------------------------------------------------------------------------
; ROM_BASE
;
; Retrieve base segment of MACH64 ROM.
;
; Inputs : none
;
; Outputs: Returns base rom segment in ax (usually C000h or C800h)
; ---------------------------------------------------------------------------
            public  rom_base

rom_base    proc    far

            ; save registers used
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            push    cx
            push    dx

            ; retrieve rom segment address from MACH64 register
            mov     dx, ioSCRATCH_REG1
            in      al, dx
            and     al, 7Fh
            mov     ah, 0
            mov     cl, 7
            shl     ax, cl
            add     ax, 0C000h

            ; restore saved registers
            pop     dx
            pop     cx

            ret

rom_base    endp
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Manual Mode Switching

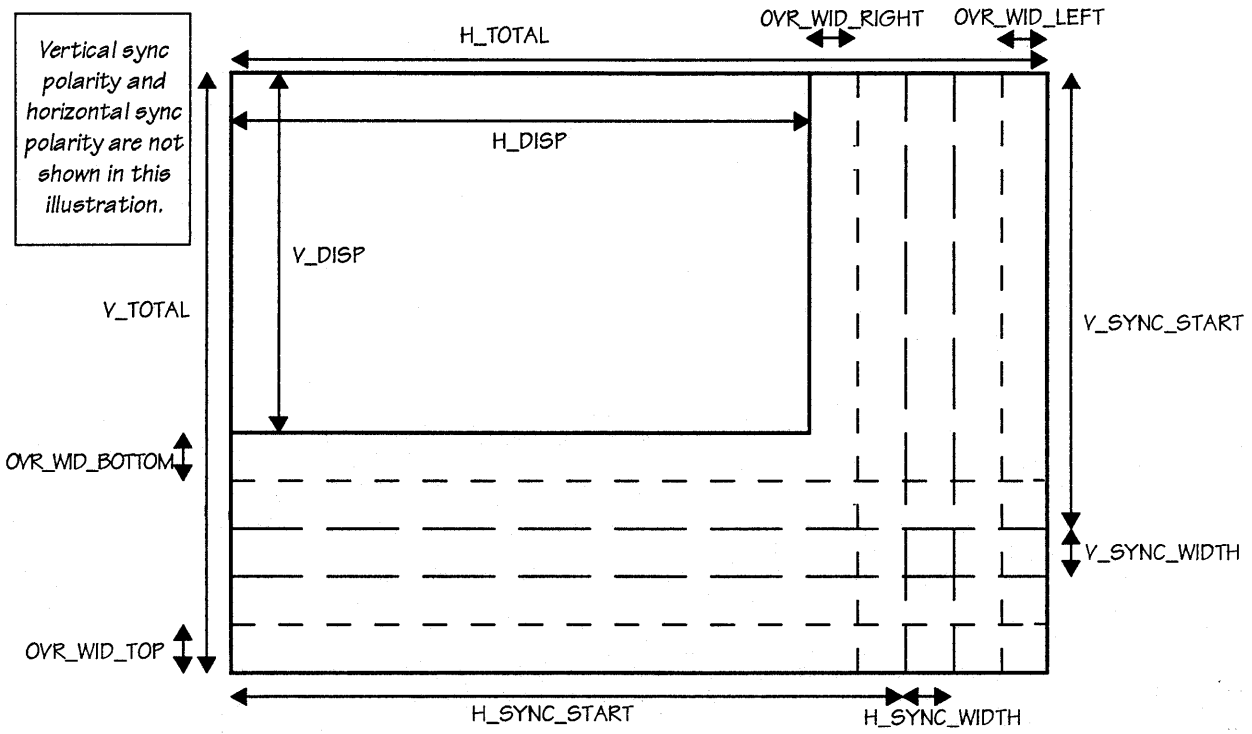
Mode switching by manual means is not recommended. If for some reason this cannot be avoided, here's how to do it:

1. *mach64* subsystems must always be configured with a non-volatile storage system for storing mode and monitor information. The application programmer must detect what kind of non-volatile storage is on board and access it appropriately to retrieve mode information. The most common configuration uses an EEPROM. See *Appendix B, EEPROM Map,* for the storage mapping of the EEPROM, and *Chapter 5, Advanced Topics II* for information on how to access the EEPROM.
2. Set the accelerator CRT controller using the information retrieved in step 1.
3. Detect the type of DAC used by reading the CONFIG_STAT0 register. Additional detection may be required for DACs that are upward-compatible with the supported DAC types. Consult the manufacturer's DAC data sheet.
4. Initialize the DAC to the appropriate pixel depth and mode using DAC_CNTL and DAC_REGS. Consult the appropriate manufacturer's DAC data sheet. Information on how to program standard DACs to standard modes is provided in *Chapter 5, Advanced Topics II.*
5. Switch from VGA mode to accelerator mode by setting the CRTC_EXT_DISP_EN bit in the CRTC_GEN_CNTL register.

Designing a Custom CRT Mode

The following illustration shows how the CRTC and overscan registers correspond to an actual video mode. The actual addressable displayed area is bounded by H_DISP and V_DISP. All registers are referenced to the upper left corner of the display area.

© 1993 ATI Technologies Inc.
Proprietary and Confidential



The relationships between CRT parameters and monitor parameters are listed in the following table:

PCLK	=	pixel clock rate (Hz)
T_{PCLK}	=	pixel clock period (sec)
H_{RES}	=	horizontal displayed resolution (pixels)
H_{SYNC}	=	horizontal sync rate (Hz)
H_{FP}	=	horizontal front porch (sec)
H_{BP}	=	horizontal back porch (sec)
H_{SWID}	=	horizontal sync width (sec)
H_{ACTIVE}	=	horizontal active time (sec)
H_{BLANK}	=	horizontal blank time (sec)
V_{RES}	=	vertical displayed resolution (pixels)
V_{SYNC}	=	vertical sync rate (Hz)
V_{FP}	=	vertical front porch (sec)
V_{BP}	=	vertical back porch (sec)
V_{SWID}	=	vertical sync width (sec)
V_{ACTIVE}	=	vertical active time (sec)
V_{BLANK}	=	vertical blank time (sec)

© 1993 ATI Technologies Inc.
Proprietary and Confidential

H_DISP	=	H_{RES} / 8 - 1
H_TOTAL	=	PCLK / H_{SYNC} / 8 - 0.5
H_SYNC_WID	=	H_{SWID} * PCLK / 8 + 0.5
H_SYNC_STRT	=	(H_{RES} + H_{FP} * PCLK + 0.5) / 8 - 1
V_DISP	=	V_{RES} - 1
V_TOTAL	=	H_{SYNC} / V_{SYNC} - 0.5
V_SYNC_WID	=	V_{SWID} * H_{SYNC} + 0.5
V_SYNC_STRT	=	V_{RES} + V_{FP} * H_{SYNC} + 0.5

H_{RES}	=	(H_DISP + 1) * 8
H_{SYNC}	=	PCLK / (H_TOTAL + 1) / 8
H_{SWID}	=	H_SYNC_WID * 8 / PCLK
H_{FP}	=	(H_SYNC_STRT - H_DISP) * 8 / PCLK
H_{BP}	=	(H_TOTAL - H_SYNC_STRT - H_SYNC_WID) * 8 / PCLK
H_{BLANK}	=	(H_TOTAL - H_DISP) * 8 / PCLK
H_{ACTIVE}	=	(H_DISP + 1) * 8 / PCLK
V_{RES}	=	V_DISP + 1
V_{SYNC}	=	HSYNC / (V_TOTAL + 1)
V_{SWID}	=	V_SYNC_WID / HSYNC
V_{FP}	=	(V_SYNC_STRT - V_DISP) / HSYNC
V_{BP}	=	(V_TOTAL - V_SYNC_STRT - V_SYNC_WID) / HSYNC
V_{BLANK}	=	(V_TOTAL - V_DISP / HSYNC
V_{ACTIVE}	=	(V_DISP + 1) / HSYNC

Note that PCLK, H_DISP, H_TOTAL, H_SYNC_WID, H_SYNC_STRT, V_DISP, V_TOTAL, V_SYNC_WID, V_SYNC_STRT, HRES, and VRES are **discrete** values, and all other parameters are **real.**

Pixel Clocks

Pixel clocks may be chosen from the ATI1881X clock chip. Refer to *Appendix D, Clock Chip Reference,* for more details.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Example CRTC calculation for 640x480 60 Hz non-interlaced: CRTC.DOC

```
Example CRTC calculation for 640x480 60 Hz non-interlaced:
==========================================================


Given parameters:
---------------

Hres  = 640
Hsync = 31.469 KHz
Hswid = 3.813 usec
Hfp   = 0.953 usec

Vres  = 480
Vsync = 59.94 Hz
Vswid = 0.064 msec
Vfp   = 0.350 msec

Pclk  = 50.35 / 2 = 25.18 MHz (ATI1881X clock chip selection 4)

Hpol  = negative polarity
Vpol  = negative polarity


CRTC calculations:
-----------------

H_TOTAL      = (Pclk / Hsync / 8) + 0.5
             = (25.18 MHz / 31.469 KHz  / 8) - 0.5
             = 99.52 = 63h

H_DISP       = Hres / 8 - 1 = 640 / 8 - 1
             = 79 = 4fh

H_SYNC_STRT = (Hres + Hfp * Pclk + 0.5) / 8 - 1
             = (640 + 0.953 usec * 25.18 MHz + 0.5) / 8 - 1
             = 82.06 = 52h

H_SYNC_WID  = (Hswid * Pclk) / 8 + 0.5
             = (3.813 usec * 25.18 MHz) / 8 + 0.5
             = 12.50 = 0ch -> 0ch + 20h (- polarity) = 2ch

V_TOTAL      = (Hsync / Vsync) - 0.5
             = (31.469 KHz / 59.94 Hz) - 0.5
             = 524.51 = 20ch

V_DISP       = Vres - 1 = 479 = 1dfh

V_SYNC_STRT = Vres + Vfp * Hsync + 0.5
             = 480 + 0.350 msec * 31.469 KHz - 0.5
             = 490.51 = 1eah
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
V_SYNC_WID  = (Vswid * Hsync) + 0.5
            = (0.064 msec * 31.469 KHz) + 0.5
            = 2.51 = 02h -> 02h + 20h (- polarity) = 22h

CLOCK_CNTL  = 14h (clock chip selection 4, divide by 2)
```

ACCELERATOR MODE

In accelerator mode, there are two ways of accessing screen memory:

- **Memory aperture**
- **Draw engine.**

Memory Aperture

The host application may read or write directly to screen memory through a memory aperture (an **aperture** is an address space that maps directly to on-board memory). Accesses through the aperture provide no acceleration, and the speed of these accesses are generally bound by the speed of the host expansion bus.

Draw Engine

The second way of accessing the memory is to use the draw engine to write to it. The draw engine can do two things:

- **Rectangle fills**
- **Lines.**

Destination trajectories and blits

These are known as **destination trajectories**. These trajectories may be filled with pixel data from various sources. If the source data comes from graphics memory, this is called a **bitblt** (or **blit**) and follows one of four different flavors of source trajectory.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Engine Initialization

Initializing or setting up a standard context for the *Mach64* engine is done using the following attributes:

- Engine reset and enabling
- FIFO control, including clearing FIFO overflow errors
- Source and destination pitch
- Source and destination offset into video memory
- Color depth
- Host data control
- Pattern data control
- Line and rectangle control
- Color source, mix, and compare control
- Scissor settings
- Write mask

Sample code

```
This is sample code to initialize the Mach 64 engine to a known context. The
values here may be changed to suit a specific application. The engine
registers are memory mapped and therefore require an enabled aperture to
access them. See discussion on apertures. For more details, see INIT.C.


/* ------------------------------------------------------------------------
  INIT_ENGINE - set the MACH64 engine to a standard context.

  This routine configures the MACH64 engine to a known typical context.
  The context consists of:

     -engine reset and enabling

     -fifo control including clearing fifo overflow errors

     -source and destination pitch

     -source and destination offset into video memory

     -color depth

     -host data control

     -pattern data control

     -line and rectangle control
```

```
     -color source, mix, and compare control

     -scissor settings

     -write mask

------------------------------------------------------------------------ */
void init_engine(void)
{
    unsigned long pitch_value, xres, yres;

    // Determine modal information from global mode structure - this
    // structure is initialized in function OPEN_MODE().

    xres = (unsigned long)(modeinfo.xres);
    yres = (unsigned long)(modeinfo.yres);
    pitch_value = (unsigned long)(modeinfo.pitch);

    // Adjust pitch if the selected mode has a color depth of 24 bpp.
    if (modeinfo.bpp == 24)
    {
        // In 24 bpp, the engine is actually in 8 bpp - the pitch is adjusted
        // by multiplying its value by 3
        pitch_value = pitch_value * 3;
    }

    // Reset engine and clear any errors
    reset_engine();

    // Ensure that vga page pointers are set to zero - the upper page
    // pointers are set to 1 to handle overflows in the lower page
    iow(ioMEM_VGA_WP_SEL, 0x00010000);
    iow(ioMEM_VGA_RP_SEL, 0x00010000);

    // Setup standard engine context
    wait_for_fifo(14);

    regw(CONTEXT_MASK, 0xFFFFFFFF);

    regw(DST_OFF_PITCH, (pitch_value / 8) << 22);
    regw(DST_Y_X, 0);
    regw(DST_HEIGHT, 0);
    regw(DST_BRES_ERR, 0);
    regw(DST_BRES_INC, 0);
    regw(DST_BRES_DEC, 0);
    regw(DST_CNTL, DST_LAST_PEL | DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

    regw(SRC_OFF_PITCH, (pitch_value / 8) << 22);
    regw(SRC_Y_X, 0);
    regw(SRC_HEIGHT1_WIDTH1, 0);
    regw(SRC_Y_X_START, 0);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    regw(SRC_HEIGHT2_WIDTH2, 0);
    regw(SRC_CNTL, SRC_LINE_X_LEFT_TO_RIGHT);

    wait_for_fifo(13);

    regw(HOST_CNTL, 0);

    regw(PAT_REG0, 0);
    regw(PAT_REG1, 0);
    regw(PAT_CNTL, 0);

    regw(SC_LEFT, 0);
    regw(SC_TOP, 0);
    regw(SC_BOTTOM, yres-1);
    regw(SC_RIGHT, pitch_value-1);

    regw(DP_BKGD_CLR, 0);
    regw(DP_FRGD_CLR, 0xFFFFFFFF);
    regw(DP_WRITE_MASK, 0xFFFFFFFF);
    regw(DP_MIX, FRGD_MIX_S | BKGD_MIX_D);
    regw(DP_SRC, FRGD_SRC_FRGD_CLR);

    wait_for_fifo(3);

    regw(CLR_CMP_CLR, 0);
    regw(CLR_CMP_MASK, 0xFFFFFFFF);
    regw(CLR_CMP_CNTL, 0);

    switch(modeinfo.bpp)
    {
        case 4 : init_4bpp();  break;
        case 8 : init_8bpp();  break;
        case 16:
            if (modeinfo.depth == 555)
            {
                init_15bpp();   // 555 color weighting
            }
            else
            {
                init_16bpp();   // 565 color weighting
            }
            break;
        case 24: init_24bpp(); break;
        case 32: init_32bpp(); break;
    }

    wait_for_idle();      // insure engine is idle before leaving
}

/* ------------------------------------------------------------------------
  RESET_ENGINE - reset engine and clear any FIFO errors
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
  This function resets the GUI engine and clears any FIFO errors.
------------------------------------------------------------------------ */
void reset_engine(void)
{
    // reset engine
    iow(ioGEN_TEST_CNTL, 0);
    iow(ioGEN_TEST_CNTL, GUI_ENGINE_ENABLE);

    // Ensure engine is not locked up by clearing any FIFO errors
    iow(ioBUS_CNTL, (ior(ioBUS_CNTL) & 0xff00ffff) | 0x00ae0000);
}

/* ------------------------------------------------------------------------
  INIT_4BPP - set the MACH64 engine to a 4bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_4bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_4BPP | SRC_4BPP | DST_4BPP |
                       BYTE_ORDER_MSB_TO_LSB);
    regw(DP_CHAIN_MASK, 0x8888);
}

/* ------------------------------------------------------------------------
  INIT_8BPP - set the MACH64 engine to a 8bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_8bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_8BPP | SRC_8BPP | DST_8BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x8080);
}

/* ------------------------------------------------------------------------
  INIT_15BPP - set the MACH64 engine to a 15bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_15bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_15BPP | SRC_15BPP | DST_15BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x4210);
}

/* ------------------------------------------------------------------------
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
  INIT_16BPP - set the MACH64 engine to a 16bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_16bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_16BPP | SRC_16BPP | DST_16BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x0410);
}

/* ------------------------------------------------------------------------
  INIT_24BPP - set the MACH64 engine to a 24bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_24bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_8BPP | SRC_8BPP | DST_8BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x8080);
}

/* ------------------------------------------------------------------------
  INIT_32BPP - set the MACH64 engine to a 32bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_32bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_32BPP | SRC_32BPP | DST_32BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x8080);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

THE LINEAR AND PAGED MEMORY APERTURES

Memory on the *mach64* may be directly accessed in one of three ways:

- **Standard paged 64k VGA aperture**
- **Small dual paged apertures**
- **Big aperture.**

Standard Paged 64k VGA Aperture

If the VGA is enabled, and the *mach64* is in VGA mode, the lower 1MB of memory may be accessed through the standard paged 64k VGA aperture. The segment base address of this aperture is either 0xA000 or 0xB000, depending on the video mode. Any memory writes via the VGA aperture are inhibited when the memory boundary is enabled. Read and write pages are set independently on 64k boundaries with this aperture type.

For more information on how to page the 64k aperture, see the *mach64 VGA Programmer's Guide.*

Small Dual Paged Apertures

If the *mach64* is in accelerator mode, **two small 32k apertures** may optionally be enabled at segment base addresses 0xA000 and 0xA800. The read and write pages are set independently on 32k boundaries for each of the two apertures with the MEM_VGA_WP_SEL and MEM_VGA_RP_SEL registers. This aperture mode is a type of VGA aperture configuration that is not available in standard VGA modes. If the memory boundary is enabled, writes to these apertures are inhibited.

- These small apertures can access the full 8MB.
- These small apertures may be enabled only if the VGA is enabled on the chip; otherwise, a memory address conflict would exist between the accelerator and the existing VGA.

PCI bus implementation

- These small apertures are not supported in the Peripheral Component Interconnect (PCI) bus implementation.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Sample code ABLIT.DOC

This is sample code to show usage of the VGA 32K apertures for transferring pixel data from one region to another. This particular example uses the apertures to transfer one line at a time to show the paging and page offset calculations. There are two write and two read 32K apertures. They may be used separately or combined to effectively make one 64K write aperture and one 64K read aperture. The combined approach has the advantage to handle 'overflows' if the lower 32K aperture size is exceeded.

```
/*=============================================================================
  ABLIT.C

  Example code to perform a screen to screen blit using the small VGA 32K
  apertures. A multi-colored filled rectangle is drawn on screen as the
  source to be blited. This source is blited by copying the pixel data
  to another screen region using the VGA 32K apertures.

  This example assumes that the VGA controller is NOT disabled.

  Copyright (c) 1994 ATI Technologies Inc. All rights reserved
 ============================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include "..\util\atim64.h"
#include "..\util\sample.h"

/* ---------------------------------------------------------------------------
  GETPAGEPTR - calculate 32K read/write pointer from a given memory address.
--------------------------------------------------------------------------- */
unsigned long getpageptr(unsigned long addr)
{
    unsigned long pageptr;

    pageptr = addr / 0x8000;

    return (pageptr);
}

/* ---------------------------------------------------------------------------
  GETADDR - calculate the memory address from a given set of coordinates.

  A byte address is returned.
--------------------------------------------------------------------------- */
unsigned long getaddr(int x, int y)
{
```

```
    unsigned long addr, xadd;

    // calculate byte address from given coordinates (x, y)
    xadd = (unsigned long)x;
    addr = (unsigned long)y;
    addr = (unsigned long)(addr * modeinfo.pitch);
    if (modeinfo.bpp == 4)
    {
        addr = addr / 2;
        xadd = xadd / 2;
    }
    else
    {
        addr = (unsigned long)(addr * (modeinfo.bpp / 8));
        xadd = (unsigned long)(xadd * (modeinfo.bpp / 8));

    }
    addr = addr + xadd;

    return (addr);
}

/* ---------------------------------------------------------------------------
  GETOFFSET - calculate offset into 32K VGA aperture from the physical
              address and page pointer.
--------------------------------------------------------------------------- */
unsigned long getoffset(unsigned long phyaddr, unsigned long pageptr)
{
    unsigned long ptraddr, offset;

    ptraddr = pageptr * 0x8000;
    offset = phyaddr - ptraddr;

    return (offset);
}

/* ---------------------------------------------------------------------------
  APPBLIT - perform a screen to screen blit using small 32K VGA apertures.

  This routine uses the lower read 32K VGA aperture as the source buffer to
  read a line of screen memory. This is transferred to the lower write 32K
  VGA aperture which represents the destination buffer. The upper 32K VGA
  aperture is used large block transfers or for overflows from the lower
  32K aperture.

  It is assumed that the VGA aperture is enabled.
--------------------------------------------------------------------------- */
void appblit(int x1, int y1, int x2, int y2, int width, int height)
{
    unsigned long readaddr, writeaddr;
    unsigned long readpage, writepage;
    unsigned long readoffset, writeoffset;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    int ySrc, yDst, index;
    int linesize;

    // calculate linesize in bytes
    if (modeinfo.bpp == 4)
    {
        linesize = width / 2;
    }
    else
    {
        linesize = width * (modeinfo.bpp / 8);
    }

    // transfer source to destination
    for (ySrc = y1, yDst = y2; ySrc < (y1 + height); ySrc++, yDst++)
    {
        // set read pointer to next source line address
        readaddr = getaddr(x1, ySrc);
        readpage = getpageptr(readaddr);
        readoffset = getoffset(readaddr, readpage);

        // upper 32k VGA aperture pointer is used for overflow
        regw(MEM_VGA_RP_SEL, ((readpage+1) << 16) | readpage);

        // set write pointer to next destination line address
        writeaddr = getaddr(x2, yDst);
        writepage = getpageptr(writeaddr);
        writeoffset = getoffset(writeaddr, writepage);

        // upper 32k VGA aperture pointer is used for overflow
        regw(MEM_VGA_WP_SEL, ((writepage+1) << 16) | writepage);

        // perform screen memory copy of one line
        for (index = 0; index < linesize; index++)
        {
            *((unsigned char far *)(LOW_APERTURE_BASE + writeoffset + index)) =
            *((unsigned char far *)(LOW_APERTURE_BASE + readoffset + index));
        }
    }
}


/* Main C program */

int main(void)
{
    // check if Mach64 adapter is installed
    if (detect_mach64() != YES_MACH64)
    {
        printf("Mach64 based adapter was not found.\n");
        return (1);
    }
```

```
    // fill global query structure by calling Mach 64 ROM
    if (query_hardware() != NO_ERROR)
    {
        printf("Failed ROM call to query Mach64 hardware.\n");
        return (1);
    }

    // check if Mach 64 VGA controller is enabled
    if (querydata.vga_type != VGA_ENABLE)
    {
        printf("This sample code example requires an enabled Mach 64 VGA
controller.\n");
        return (1);
    }

    // The VGA controller is normally set in planar mode. Data transfer
    // through the VGA aperture (low and high 32K pages) requires that the
    // VGA controller be set in a packed pixel mode where the pixel data
    // is arranged contigiously.
    set_packed_pixel();

    // set an accelerator mode
    if (open_mode(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8) != NO_ERROR)
    {
        printf("Error in setting display mode.\n");
        return (1);
    }

    // Initialize standard engine context
    init_engine();
    clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

    // Draw some rectangles to act as a blit source
    set_fg_color(get_color_code(YELLOW));
    draw_rectangle(0, 0, modeinfo.xres / 2, modeinfo.yres / 2);

    set_fg_color(get_color_code(LIGHTRED));
    draw_rectangle((modeinfo.xres / 32),
                   (modeinfo.yres / 24),
                   (modeinfo.xres / 2) - (2 * (modeinfo.xres / 32)),
                   (modeinfo.yres / 2) - (2 * (modeinfo.yres / 24)));

    set_fg_color(get_color_code(LIGHTGREEN));
    draw_rectangle((2 * (modeinfo.xres / 32)),
                   (2 * (modeinfo.yres / 24)),
                   (modeinfo.xres / 2) - (4 * (modeinfo.xres / 32)),
                   (modeinfo.yres / 2) - (4 * (modeinfo.yres / 24)));

    set_fg_color(get_color_code(LIGHTBLUE));
    draw_rectangle((3 * (modeinfo.xres / 32)),
                   (3 * (modeinfo.yres / 24)),
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                    (modeinfo.xres / 2) - (6 * (modeinfo.xres / 32)),
                    (modeinfo.yres / 2) - (6 * (modeinfo.yres / 24)));

    // Draw a DARKGRAY rectangle to show where the destination blit will be
    // drawn
    set_fg_color(get_color_code(DARKGRAY));
    draw_rectangle(modeinfo.xres / 2, modeinfo.yres / 2,
                   modeinfo.xres / 2, modeinfo.yres / 2);

    // wait for a carriage return
    getch();

    // perform blit using small 32K VGA apertures
    appblit(0, 0, modeinfo.xres / 2, modeinfo.yres / 2,
            modeinfo.xres / 2, modeinfo.yres / 2);

    // wait for a carriage return
    getch();

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Big Aperture

If the *mach64* is in accelerator mode, a **big linear aperture** may be optionally enabled to access the entire frame buffer. This aperture may be 4M or 8M in size, and is locatable on 4M or 8M boundaries, respectively, anywhere in the 4G address space. The memory boundary does not affect memory accesses through this aperture.

If the memory boundary is enabled, the lower address space of the big aperture will be occupied by VGA memory.

The availability of this aperture is assured on all board configurations except ISA bus configurations. On an ISA system, the following two conditions must be met in order to use the big aperture:

- The aperture must fit within a 16MB address space.
- The aperture must not overlap host CPU memory.

A typical set would be a system with 8MB of host CPU memory, with a big aperture of 4MB or 8MB, starting at an address of 8MB.

An ISA system with 16MB or more of host CPU memory cannot use a big aperture.

Sample code LINAPP.DOC

```
See source file LINAPP.C for more details.

unsigned int buffer[200];

int main(void)
{
    int config, memcntl, i, j;
    unsigned long appaddr;
    union REGS regs;

    // check if Mach64 adapter is installed
    if (detectmach64() != YES_MACH64)
    {
        printf("Mach64 based adapter was not found.\n");
        return (1);
    }

    // set an accelerator mode
    if (load_and_set_mode(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8) != NO_ERROR)
    {
        printf("Error in setting display mode.\n");
        return (1);
    }

    // enable the linear memory aperture
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
outpw(ioCONFIG_CNTL, inpw(ioCONFIG_CNTL) | APERTURE_4M_ENABLE);

// set engine context
init_engine(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8);
clear_screen(0, 0, 640, 480);

// get linear memory aperture base address
config = inpw(ioCONFIG_CNTL);
appaddr = (unsigned long)((config & 0x3ff0) >> 4);
appaddr = appaddr << 22;
if (appaddr == 0)
{
    // go back to VGA mode
    set_display_mode(VGA_MODE);

    // reinitialize VGA
    regs.x.ax = 3;
    int86(0x10, &regs, &regs);

    printf("Error - linear memory aperture has a zero address.\n");
    return (1);
}

// insure that memory is shared between VGA and acelerator for full
//  memory access
memcntl = inpw(ioMEM_CNTL+2);
outpw(ioMEM_CNTL+2, 0);

// fill buffer with data and write to screen memory through linear aperture
for (i = 0; i < 200; i++)
{
    buffer[i] = i;
}
movemem((void far *)buffer, appaddr, 200, MEM_WRITE);

// clear buffer and read screen memory back
for (i = 0; i < 200; i++)
{
    buffer[i] = 0;
}
movemem((void far *)buffer, appaddr, 200, MEM_READ);

// restore MEM_CNTL register
outpw(ioMEM_CNTL+2, memcntl);

// go back to VGA mode
set_display_mode(VGA_MODE);

// Since the accelerator and VGA controllers share the same video memory,
// it is necessary to reinitialize the VGA text mode before exiting. Text
// mode 3 is used here.
regs.x.ax = 3;
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    int86(0x10, &regs, &regs);

    // dump buffer to VGA screen
    printf("\nLINEAR APERTURE read dump - values should be 0 to C7h\n");
    j = 0;
    for (i = 0; i < 200; i++)
    {
        printf("%04X ", buffer[i]);
        j++;
        if (j > 9)
        {
            j = 0;
            printf("\n");
        }
    }
    printf("\n");

    return (0);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

REGISTER MAPPING

All registers not associated with the draw engine are I/O mapped, and all except CONFIG_CNTL have memory mapped register aliases. All I/O mapped addresses have a base of 0x2EC (in the ISA style of sparse I/O decoding, the base address is the lower ten bits, and the upper six bits are used for modifier bits — in this case, they are register selects) and are 32 bits wide.

All registers, except for CONFIG_CNTL, are memory-mapped. All registers, except for DAC_REGS (they are four, eight-bit registers), are 32 bits wide.

- **If the small apertures are enabled,** the registers may be accessed through a 1K area at segment:offset of 0xB000:0xFC00.
- **If the big aperture is enabled,** the registers occupy the address space located at the base address of the aperture, plus an offset of 0x3FFC00 for a 4M aperture or 0x7FFC00 for a 8M aperture configuration.

Sample code REGMAP.DOC

```
This is sample code to show how to access the memory mapped registers using
the VGA and linear apertures.


/*=========================================================================
  REGMAP.C

  Example code to write and read a memory mapped register using the VGA
  aperture, 4M linear aperture size, or 8M linear aperture size. For more
  details, see functions REGW() and REGR() in MEMREG.C.

  Note that for GUI register access with the linear aperture, the aperture
  address must be valid. The INSTALL program should be used to ENABLE the
  linear aperture.

  Failures:

    - VGA aperture register access will fail if the VGA controller is disabled

    - Linear aperture register access will fail if the memory mapped
      register area is overlapping extended memory or resides at an invalid
      address


  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
 =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
```

```
#include "..\util\atim64.h"
#include "..\util\sample.h"

int main(void)
{
    int config;
    unsigned long regval;

    // save config value
    config = inpw(ioCONFIG_CNTL);

    // fill global query structure by calling Mach 64 ROM
    if (query_hardware() != NO_ERROR)
    {
        printf("Failed ROM call to query Mach64 hardware.\n");
        return (1);
    }

    // Enable GUI engine to turn on GUI memory mapped register access
    reset_engine();

    // Insure that fifo is empty before starting - enable linear aperture
    // to insure that the wait_for_idle() can read the FIFO_STAT and
    // GUI_STAT registers. The VGA aperture may not be available. This is
    // necessary since the register under test (PAT_REG0) is a GUI memory
    // mapped register.

    enable_linear_aperture(APERTURE_4M_ENABLE);
    wait_for_idle();

    // ------------------------------------------------------------------
    // Use the VGA aperture to access the GUI registers. The base address
    // is B000:FC00h. Each register occupies 4 bytes.

    // Use VGA aperture if the VGA controller is enabled
    if (querydata.vga_type == VGA_ENABLE)
    {
        // enable the VGA aperture for register access
        disable_linear_aperture();
        enable_vga_aperture();

        // Use this function to insure regw() and regr() work correctly
        update_aperture_status();

        // write and read the PAT_REG0 register using the VGA aperture
        regval = 0x555aa5aa;
     printf("VGA aperture enabled    : write value = %08lX, read value = ", regval);

        regw(PAT_REG0, regval);

        regval = regr(PAT_REG0);
        printf("%08lX\n", regval);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    }
    else
    {
        printf("VGA controller is disabled.\n");
    }

    // ------------------------------------------------------------------
    // Use the 4Meg linear aperture size to access the GUI registers. The
    // base address is the aperture address + 0x3FFC00h. Each register
    // occupies
    // 4 bytes.

    // enable the 4M linear memory aperture without distribing address
    disable_vga_aperture();
    enable_linear_aperture(APERTURE_4M_ENABLE);

    // Use this function to insure regw() and regr() work correctly
    update_aperture_status();

    // write and read the PAT_REG0 register using the 4Meg linear aperture
    regval = 0xaaa55a55;
  printf("4M Linear aperture enabled: write value = %08lX, read value = ", regval);

    regw(PAT_REG0, regval);

    regval = regr(PAT_REG0);
    printf("%08lX\n", regval);

    // ------------------------------------------------------------------
    // Use the 8Meg linear aperture size to access the GUI registers. The
    // base address is the aperture address + 0x7FFC00h. Each register
    // occupies 4 bytes.

    // enable the 8M linear memory aperture without distribing address
    enable_linear_aperture(APERTURE_8M_ENABLE);

    // Use this function to insure regw() and regr() work correctly
    update_aperture_status();

    // write and read the PAT_REG0 register using the 8Meg linear aperture
    regval = 0x5a55aaa5;
  printf("8M Linear aperture enabled: write value = %08lX, read value = ", regval);

    regw(PAT_REG0, regval);

    regval = regr(PAT_REG0);
    printf("%08lX\n", regval);

    // restore configuration
    outpw(ioCONFIG_CNTL, config);

    return (0);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

THE COMMAND FIFO

All draw engine registers are memory-mapped with DWORD offsets greater than or equal to 0x40. All writes to draw engine registers are automatically routed through a 32-bit-wide, 16-entry-deep, command FIFO. All entries are consumed in the same order as written.

- Note that host data registers do not generate extra wait states as did the *mach32*, and complete FIFO discipline is required for these registers as well.
- Register reads are not FIFOed in any fashion.
- Register writes to registers with DWORD offsets less than 0x40 are not FIFOed.

Waiting for Sufficient FIFO Entries

Prior to any writes to any draw engine register, it is essential to check the state of the command FIFO to ensure that enough FIFO entries are available. Failure to do so may cause the draw engine to lock. C source code that waits for *n* free entries is shown below:

Sample code CHKFIFO.DOC

```
This is sample code to wait for a specific number of empty fifo entries. When
a GUI memory mapped register is written to, it goes through the FIFO. If the
FIFO fills up (16 or more writes without waiting for empty FIFO entries), the
FIFO error bit will be set and all subsequent writes will be ignored until
the condition is cleared (see RESET_ENGINE() in INIT.C). It should be noted
that since the GUI memory mapped registers are FIFOed during a write
operation, the write width must be 32 bits. It is not necessary to check for
empty fifo entries when reading GUI memory mapped registers or accessing
non-GUI registers.


/* ------------------------------------------------------------------------
  WAIT_FOR_FIFO - wait for n empty FIFO entries.

  The FIFO is 16 entries by 32 bits wide.
------------------------------------------------------------------------ */
void wait_for_fifo(int entries)
{
    while ((regr(FIFO_STAT) & 0xffff) > ((unsigned int)(0x8000 >> entries)));
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Waiting for Engine Idle

There are two cases where the application must wait for the draw engine to become idle:

- The first case occurs when the application is depending on the draw engine to update a register or bit field (such as DST_X, or the scissor status bits in the GUI_STAT register). The application must ensure idleness so that those registers will not be read back while in an intermediate state.
- The second case occurs when some draw operations are being performed by the draw engine, and memory is being accessed through an aperture, and spatial mutual exclusivity cannot be guaranteed. If an engine write and an aperture write are occurring in the same region, the pixel that lands on top will not be deterministic. If an engine write and an aperture read are occurring in the same region, the pixel that is read back may or may not be the pixel just drawn.

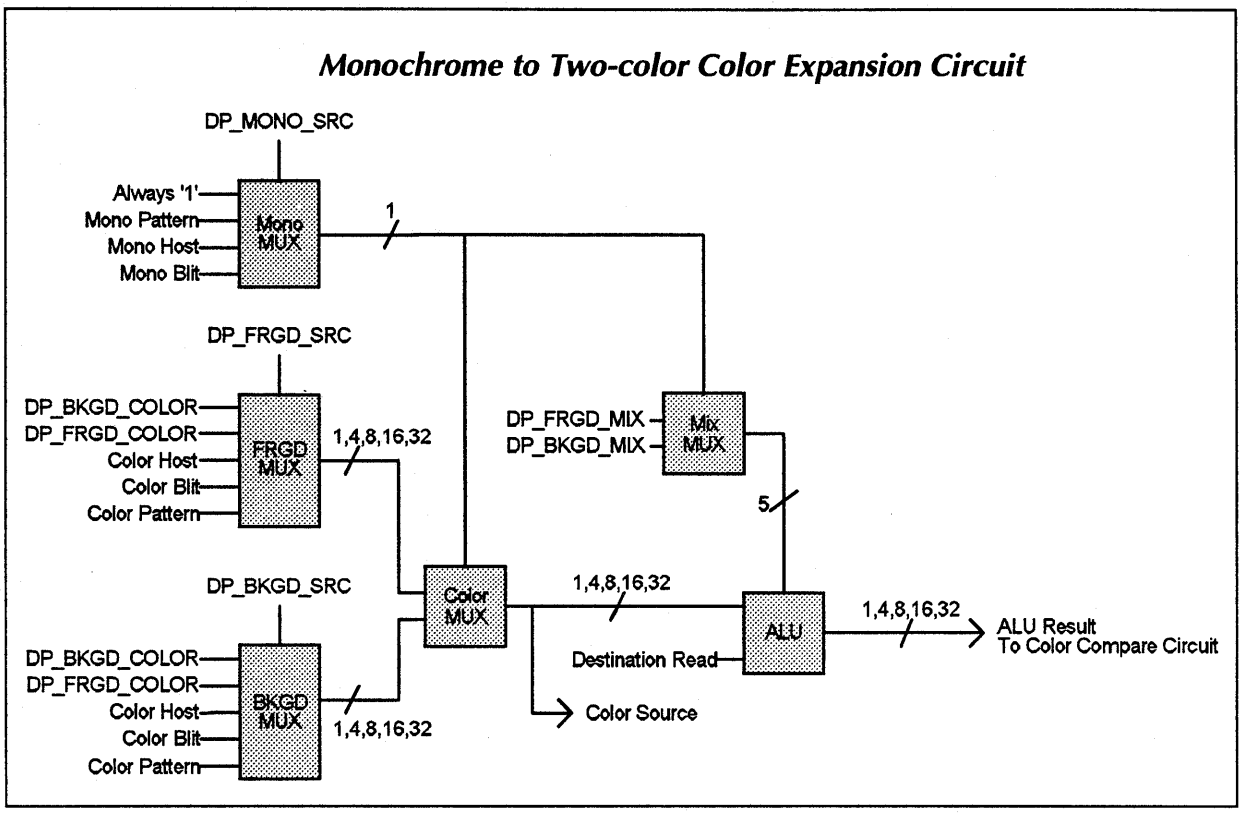
Sample code CHKIDLE.DOC

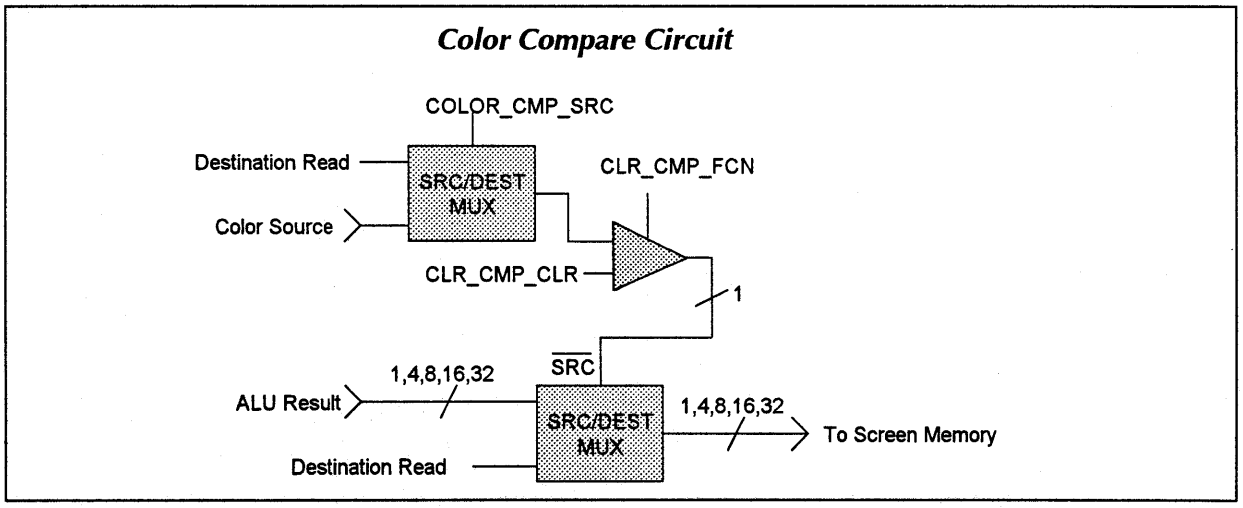
```
This is sample code to wait until the engine is idle. If the engine FIFO is
empty, this does not imply that the engine is idle. An idle engine indicates
that all operations have completed. When a GUI engine register is read, the
engine should be idle to insure that the register is updated before the read
takes place.


/* ------------------------------------------------------------------------
  WAIT_FOR_IDLE - wait until the engine is idle.
------------------------------------------------------------------------ */
void wait_for_idle(void)
{
    // wait for empty fifo
    wait_for_fifo(16);

    // wait until the engine is idle
    while ((regr(GUI_STAT) & 1) != 0) ;
}
```

A state of idleness implies 16 free FIFO entries, but 16 free FIFO entries do **not** imply a state of idleness.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

LOGICAL PIXEL DATA PATH





Note: These illustrations are VERY important in understanding the *mach64*.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

- For each pixel, a mono source bit is consumed. If it is '1', the foreground color source and foreground mix are used; otherwise, the background color source and background mix are used.
- The color source is mixed with the destination in the ALU, and passed on to the compare logic. If the color compare source is set for destination, the destination pixel is compared against the compare color; otherwise, the source color is compared against the compare color.
- If the result of the comparison is FALSE, the mixed color is written to the destination; otherwise, the destination color is written to the destination.

The **pixel data path** has inherent monochrome-to-color expansion characteristics. For instance, text operations — whose source data is monochrome — can be expanded to two colors by writing the text data to graphics memory, setting DP_MONO_SRC to Mono_Blit, DP_FRGD_SRC to DP_FRGD_CLR, and DP_BKGD_SRC to DP_BKGD_CLR and doing a blit operation. If monochrome expansion is not desired, simply set the DP_MONO_SRC to Always_'1' and only the foreground color source will be used.

The **color compare circuit** can be used for functions such as transparent blits, where a color source can be compared to a color that is used to determine whether a pixel is written to the screen or not. If no color comparison features are desired, simply set the color compare function to FALSE.

All **pattern sources** in the above diagram refer to fixed patterns. All **blit sources** refer to a blit source trajectory, as described in *Programming Model, Trajectories*.

The illustration represents the *logical* data path. The *physical* data path is actually 64 bits wide for all color data. Therefore, in 8bpp modes, eight pixels are processed simultaneously; in 16bpp modes, four pixels are processed simultaneously, and so on.

Every draw engine operation must color-expand a monochrome source into two color sources. If only a single color source is desired, the monochrome source is set to Always_'1' and only the foreground source and foreground mix are used.

The terms **foreground source** and **background source** just mean **color source 1** and **color source 2**, and that foregrounds and backgrounds are just a convenient conceptualization.

DP_FRGD_CLR is the foreground color register.

DP_BKGD_CLR is the background color register.

Color_Host means that data is to be written through the HOST_DATA register. If Color_Host is selected for one of the color sources, host data will be consumed for every pixel, regardless of whether the color MUX selects that color source. Note that the host pixel depth must be set to the same pixel depth as the destination pixel depth with the DP_PIX_WIDTH register.

Also, the HOST_CNTL register controls consumption of host data on 1bpp and 4bpp data. If the HOST_BYTE_ALIGN bit is set, host consumption advances to the nearest byte boundary whenever the destination trajectory advances in the Y direction.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Color_Blit source selects one of the four possible source trajectories (see *Trajectories* on page 2-37). The option bits in the SRC_CNTL register determine which of the four trajectories is to be used. If Color_Blit is selected as one of the color sources, pixels from this source will always be consumed for each destination pixel consumed, regardless of whether the color MUX selects that source.

Color_Pattern source are the 4x2 and 8x1 fixed color patterns. These patterns are only useful in 8bpp draw modes. The PAT_CNTL register is used to select which of the color patterns is to be used and the PAT_REG registers define the pattern itself.

Always_'1' monochrome source forces the foreground source and foreground mix to always be used.

Mono_Pattern monochrome source is the 8x8 fixed mono pattern. PAT_CNTL enables the mono pattern and the PAT_REG registers define it.

Mono_Host is much the same as Color_Host data, except that the host pixel depth must be set to monochrome with DP_PIX_WIDTH, and that monochrome data is color expanded into foreground and background color sources.

Mono_Blit is much the same as Color_Blit except that the source pixel depth must be set to monochrome with the DP_PIX_WIDTH register, and the monochrome data is color-expanded into foreground and background color sources.

1bpp means a color of zero or one; **monochrome** means two colors, which is not necessarily zero and one. In relation to the architecture, monochrome implies a two-color color expansion of a monochrome source. In some cases, these two terms may be used interchangeably.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Host Data Consumption

The following tables illustrate the order in which pixels are consumed from the HOST_DATA register. The shaded numbers indicate the bit position within the HOST_DATA register. The numbers in the table indicate the order of pixel consumption, starting from zero.

	HOST_DATA															
	31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Monochrome or 1bpp, left-to-right, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=0	24	25	26	27	28	29	30	31	16	17	18	19	20	21	22	23
	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
Monochrome or 1bpp, right-to-left, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=0	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
	23	22	21	20	19	18	17	16	31	30	29	28	27	26	25	24
Monochrome or 1bpp, left-to-right, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=1	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Monochrome or 1bpp, right-to-left, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

	HOST_DATA							
	31-28	**27-24**	**23-20**	**19-16**	**15-12**	**11-8**	**7-4**	**3-0**
4bpp, left-to-right, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=0	6	7	4	5	2	3	0	1
4bpp, right-to-left, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=0	1	0	3	2	5	4	7	6
bpp, left-to-right, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=1	7	6	5	4	3	2	1	0
4bpp, right-to-left, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH=1	0	1	2	3	4	5	6	7
8bpp, left-to-right	3		2		1		0	
8bpp, right to left	0		1		2		3	
16bpp, left-to-right	1				0			
16bpp, right-to-left	0				1			

Notes:

- Host data consumption for 32 bits per pixel is self-evident.
- Host data consumption for line draws is the same as for left-to-right trajectories.
- Pixel consumption in 15bpp modes is the same as 16bpp modes.
- Packed 24bpp mode is essentially 8bpp mode. R, G, and B component data must be fed in individually in 8-bit chunks.
- The HOST_BYTE_ALIGN@HOST_CNTL bit may affect pixel consumption for 1bpp and 4bpp modes. When set, pixel consumption advances to the next nearest byte boundary whenever the destination advances in the Y direction. Line draws are unaffected.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Pattern Consumption

Pattern consumption for the various fixed patterns is shown in the tables below. P0 and P1 indicate PAT_REG0 and PAT_REG1, respectively. The number(s) in parentheses are the bits within the pattern registers, which are used according to the destination pixel location.

Monochrome 8x8 fixed pattern, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH = 0								
(DST_Y mod 8)	(DST_X mod 8)							
	0	1	2	3	4	5	6	7
0	P0(7)	P0(6)	P0(5)	P0(4)	P0(3)	P0(2)	P0(1)	P0(0)
1	P0(15)	P0(14)	P0(13)	P0(12)	P0(11)	P0(10)	P0(9)	P0(8)
2	P0(23)	P0(22)	P0(21)	P0(20)	P0(19)	P0(18)	P0(17)	P0(16)
3	P0(31)	P0(30)	P0(29)	P0(28)	P0(27)	P0(26)	P0(25)	P0(24)
4	P1(7)	P1(6)	P1(5)	P1(4)	P1(3)	P1(2)	P1(1)	P1(0)
5	P1(15)	P1(14)	P1(13)	P1(12)	P1(11)	P1(10)	P1(9)	P1(8)
6	P1(23)	P1(22)	P1(21)	P1(20)	P1(19)	P1(18)	P1(17)	P1(16)
7	P1(31)	P1(30)	P1(29)	P1(28)	P1(27)	P1(26)	P1(25)	P1(24)

Monochrome 8x8 fixed pattern, DP_BYTE_PIX_ORDER@DP_PIX_WIDTH = 1								
(DST_Y mod 8)	(DST_X mod 8)							
	0	1	2	3	4	5	6	7
0	P0(0)	P0(1)	P0(2)	P0(3)	P0(4)	P0(5)	P0(6)	P0(7)
1	P0(8)	P0(9)	P0(10)	P0(11)	P0(12)	P0(13)	P0(14)	P0(15)
2	P0(16)	P0(17)	P0(18)	P0(19)	P0(20)	P0(21)	P0(22)	P0(23)
3	P0(24)	P0(25)	P0(26)	P0(27)	P0(28)	P0(29)	P0(30)	P0(31)
4	P1(0)	P1(1)	P1(2)	P1(3)	P1(4)	P1(5)	P1(6)	P1(7)
5	P1(8)	P1(9)	P1(10)	P1(11)	P1(12)	P1(13)	P1(14)	P1(15)
6	P1(16)	P1(17)	P1(18)	P1(19)	P1(20)	P1(21)	P1(22)	P1(23)
7	P1(24)	P1(25)	P1(26)	P1(27)	P1(28)	P1(29)	P1(30)	P1(31)

8bpp, 4x2 fixed pattern				
(DST_Y mod 2)	(DST_X mod 4)			
	0	1	2	3
0	P0(7:0)	P0(15:8)	P0(23:16)	P0(31:24)
1	P1(7:0)	P1(15:8)	P1(23:16)	P1(31:24)

8bpp, 8x1 fixed pattern							
(DST_X mod 8)							
0	1	2	3	4	5	6	7
P0(7:0)	P0(15:8)	P0(23:16)	P0(31:17)	P1(7:0)	P1(15:8)	P1(23:16)	P1(31:24)

© 1993 ATI Technologies Inc.
Proprietary and Confidential

TRAJECTORIES

This table summarizes all possible trajectories. Source and destination trajectories are consumed independently, with the exception of source direction, which tracks the destination direction for blits.

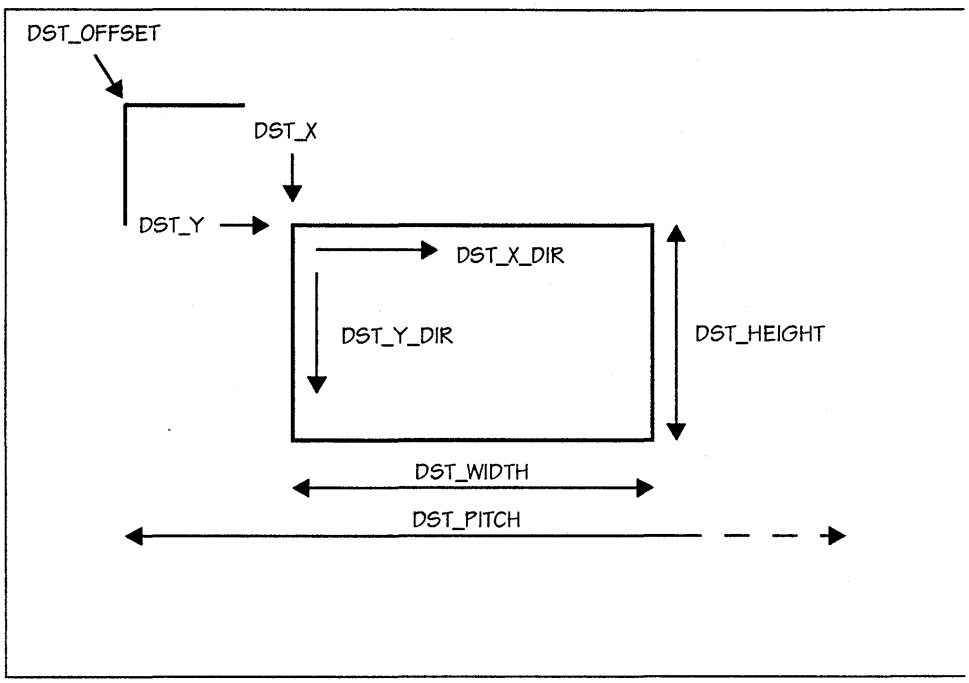
The **Trajectory Registers** column specifies all the registers that need to be initialized for the desired trajectory.

The **Enable/Initiate** column indicates the bits that must be set to achieve the desired trajectory, or the register to write to in order to initiate the draw operation. All rectangular trajectories are X major.

	Trajectory	Trajectory Registers	Enable/Initiate	
Source	Strictly Linear	SRC_OFFSET If destination is line, then SRC_LINE_X_DIR@SRC_CNTL, else DST_X_DIR@DST_CNTL	SRC_LINEAR@SRC_CNTL==1	Enables
	Unbounded Y	SRC_OFFSET, SRC_PITCH, SRC_X, SRC_Y, SRC_WIDTH1 If destination is line, then also SRC_LINE_X_DIR@SRC_CNTL	SRC_PATT_ENA@SRC_CNTL==0 SRC_PATT_ROT@SRC_CNTL==0 SRC_LINEAR@SRC_CNTL==0	
	General Pattern	SRC_OFFSET, SRC_PITCH, SRC_X, SRC_Y, SRC_WIDTH1, SRC_HEIGHT1 If destination is line, then also SRC_LINE_X_DIR@SRC_CNTL	SRC_PATT_ENA@SRC_CNTL==1 SRC_PATT_ROT@SRC_CNTL==0 SRC_LINEAR@SRC_CNTL==0	
	General Pattern with Rotation	SRC_OFFSET, SRC_PITCH, SRC_X, SRC_Y, SRC_WIDTH1, SRC_HEIGHT1, SRC_X_START, SRC_Y_START, SRC_WIDTH2, SRC_HEIGHT2 If destination is line, then also SRC_LINE_X_DIR@SRC_CNTL	SRC_PATT_ENA@SRC_CNTL==1 SRC_PATT_ROT@SRC_CNTL==1 SRC_LINEAR@SRC_CNTL==0	
Destination	Rectangle	DST_OFFSET, DST_PITCH, DST_X, DST_Y, DST_WIDTH, DST_HEIGHT, DST_X_DIR@DST_CNTL, DST_Y_DIR@DST_CNTL	DST_WIDTH or DST_HEIGHT_WIDTH or DST_X_WIDTH	Initiators
	Line	DST_OFFSET, DST_PITCH, DST_X, DST_Y, DST_BRES_LNTH, DST_BRES_ERR, DST_BRES_INC, DST_BRES_DEC, DST_X_DIR@DST_CNTL, DST_Y_DIR@DST_CNTL, DST_Y_MAJOR@DST_CNTL	DST_BRES_LNTH	

© 1993 ATI Technologies Inc.
Proprietary and Confidential

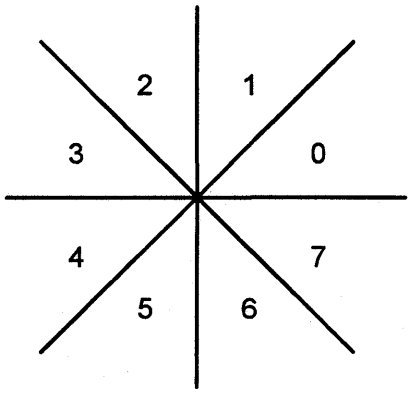
Destination Trajectory 1, Rectangular



Initiator: DST_WIDTH | | DST_HEIGHT_WIDTH | | DST_X_WIDTH

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Destination Trajectory 2, Line



Octant	DST_X_DIR	DST_Y_MAJOR	DST_Y_DIR
0	1	0	0
1	1	1	0
2	0	1	0
3	0	0	0
4	0	0	1
5	0	1	1
6	1	1	1
7	1	0	1

Initiator: DST_BRES_LNTH

Comments: The bresenham parameters are calculated as follows...

DST_BRES_ERR = 2 * min(|dx|,|dy|) - max(|dx|,|dy|)

DST_BRES_INC = 2 * min(|dx|,|dy|)

DST_BRES_DEC = 2 * [min(|dx|,|dy|) - max(|dx|,|dy|)]

DST_BRES_LNTH = max(|dx|,|dy|) + 1

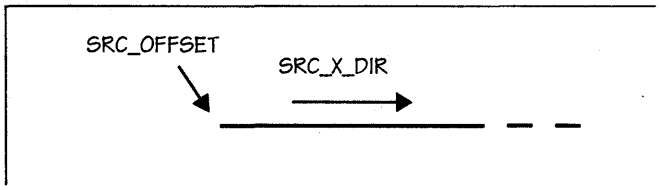
The line drawing pseudocode is:

```
if (DST_BRES_ERR < 0) {
        do axial step
        DST_BRES_ERR += DST_BRES_INC
} else {
        do diagonal step
        DST_BRES_ERR += DST_BRES_DEC
}
```

where the axial and diagonal step directions are determined by the three octant bits. The octant bits reside in DST_CNTL.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

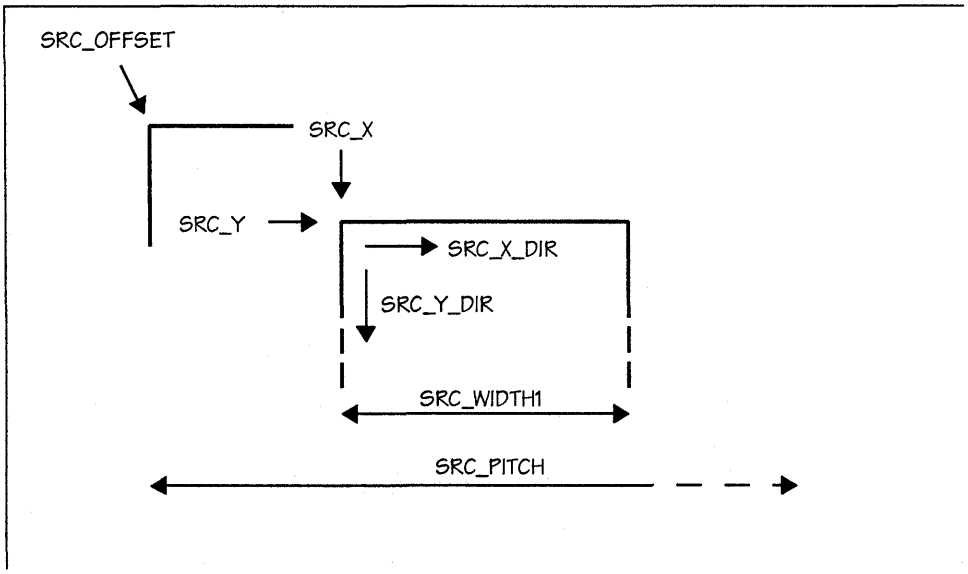
Source Trajectory 1, Strictly Linear



Criterion: SRC_LINEAR@SRC_CNTL==1

Comments: Source offset and SRC_X_DIR are the only parameters used to set up the source trajectory. Pixels are consumed in a strictly linear fashion in memory. SRC_X_DIR tracks DST_X_DIR@DST_CNTL in the case of blits. For lines, SRC_X_DIR equals SRC_LINE_X_DIR@SRC_CNTL. SRC_X_DIR should be set to go from left-to-right.

Source Trajectory 2, Unbounded Y

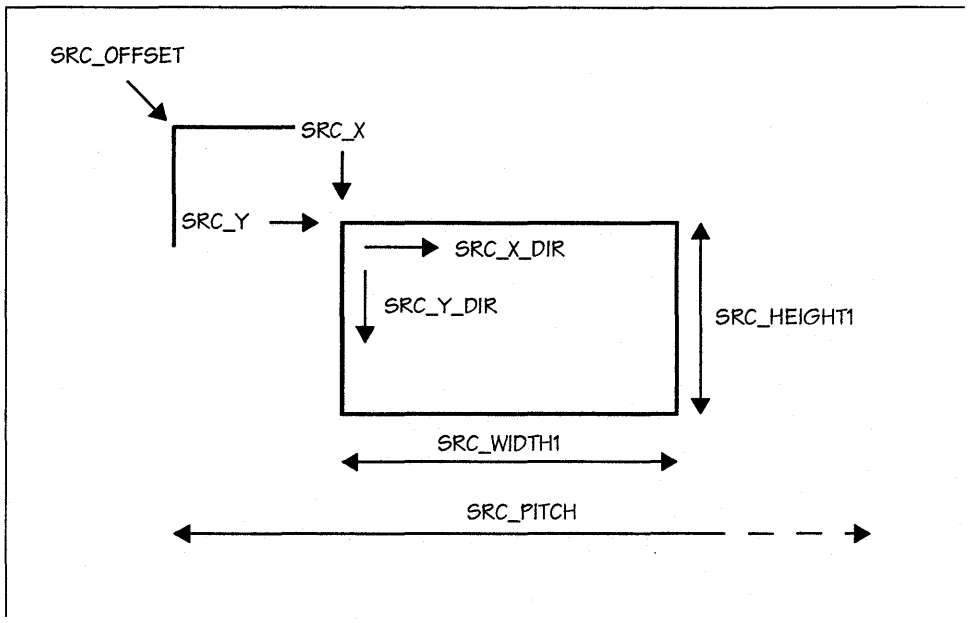


Criterion: SRC_PATT_EN@SRC_CNTL==0 && SRC_PATT_ROT@SRC_CNTL=0 && SRC_LINEAR@SRC_CNTL==0

Comments: If the destination trajectory is rectangular, SRC_X_DIR and SRC_Y_DIR track DST_X_DIR@DST_CNTL and DST_Y_DIR@DST_CNTL. For lines, SRC_LINE_X_DIR@SRC_CNTL is used and the source trajectory does not advance in the Y direction.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Source Trajectory 3, General Pattern

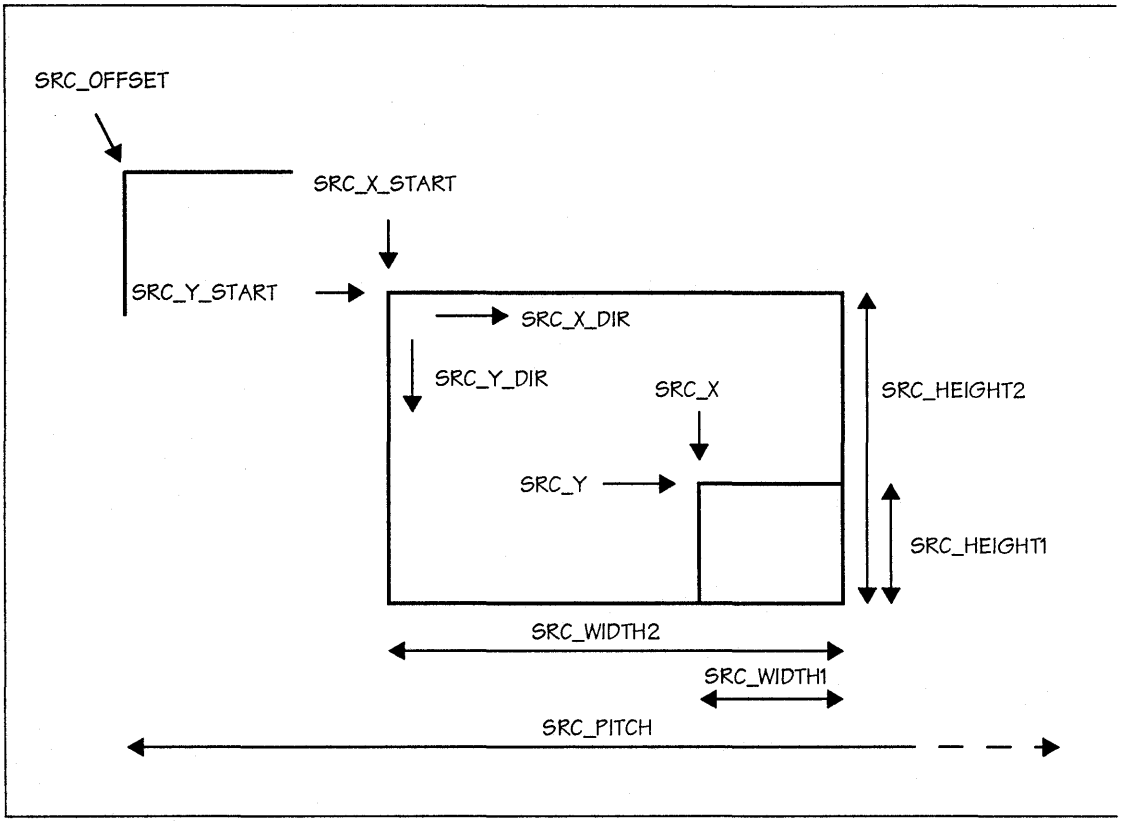


Criterion: SRC_PATT_EN@SRC_CNTL==1 && SRC_PATT_ROT@SRC_CNTL==0 && SRC_LINEAR@SRC_CNTL==0

Comments: If the destination trajectory is rectangular, SRC_X_DIR and SRC_Y_DIR track DST_X_DIR@DST_CNTL and DST_Y_DIR@DST_CNTL. For lines, SRC_LINE_X_DIR@SRC_CNTL is used and the source trajectory does not advance in the Y direction.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Source Trajectory 4, General Pattern With Rotation



Criterion: SRC_PATT_ROT@SRC_CNTL==1 && SRC_LINEAR@SRC_CNTL==0

Comments: If the destination trajectory is rectangular, SRC_X_DIR and SRC_Y_DIR track DST_X_DIR@DST_CNTL and DST_Y_DIR@DST_CNTL. For lines, SRC_LINE_X_DIR@SRC_CNTL is used and the source trajectory does not advance in the Y direction.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Trajectory Modifier 1, SRC_BYTE_ALIGN

When SRC_BYTE_ALIGN@SRC_CNTL is set, the source pointer skips to the next byte boundary when the destination trajectory advances in the Y direction. There is a similar bit for host data called HOST_BYTE_ALIGN@HOST_CNTL. These bits are only meaningful for 1bpp or 4bpp data.

Trajectory Modifier 2, DST_POLYGON_ENA

The DST_POLYGON_ENA affects both lines and blits.

When drawing a line, only a single pixel is drawn per scan line (this only affects X major lines). Horizontal lines are not drawn. Lines whose trajectory goes left of the left scissor are saturated to the left scissor.

When blitting, at the beginning of each destination line, an internal polygon fill flag is reset. If the polygon fill flag is reset, drawing is inhibited at the destination. For each pixel, an implicit 1bpp polygon boundary source (this is neither a monochrome nor a color source, but an implicit third source) is read. If the result is '1' (a polygon edge) the polygon fill flag is toggled. Both left and right edges of the polygon are inclusive.

Trajectory Modifier 3, DP_BYTE_PIX_ORDER

The DP_BYTE_PIX_ORDER@DP_PIX_WIDTH bit affects the pixel order of 1bpp and 4 bpp data within a byte. This affects the source area, destination area, and host data consumption. When set, pixel order proceeds from least significant bit or nibble to most significant bit or nibble within a byte. The bytewise order is unaffected. See *Host Data Consumption* on page 2-35 for a graphical view of the pixel ordering.

SIDE EFFECTS

- The source pointer is always reset to the original SRC_X, SRC_Y after completion of a draw operation.
- The destination pointer is set according to the DST_X_TILE and DST_Y_TILE bits after completion of a blit operation. If DST_X_TILE is set, then DST_X = original_DST_X + DST_WIDTH for left-to-right destination trajectories, or DST_X = original_DST_X - DST_WIDTH for right-to-left destination trajectories; otherwise, it is set to the original DST_X value before the draw occurred. Similarly for the DST_Y_TILE bit (with DST_Y and DST_HEIGHT).
- For lines, the final DST_X, DST_Y rests on the last pixel of the line. The LAST_PEL_ON bit specifies whether the last pixel on that line is drawn.

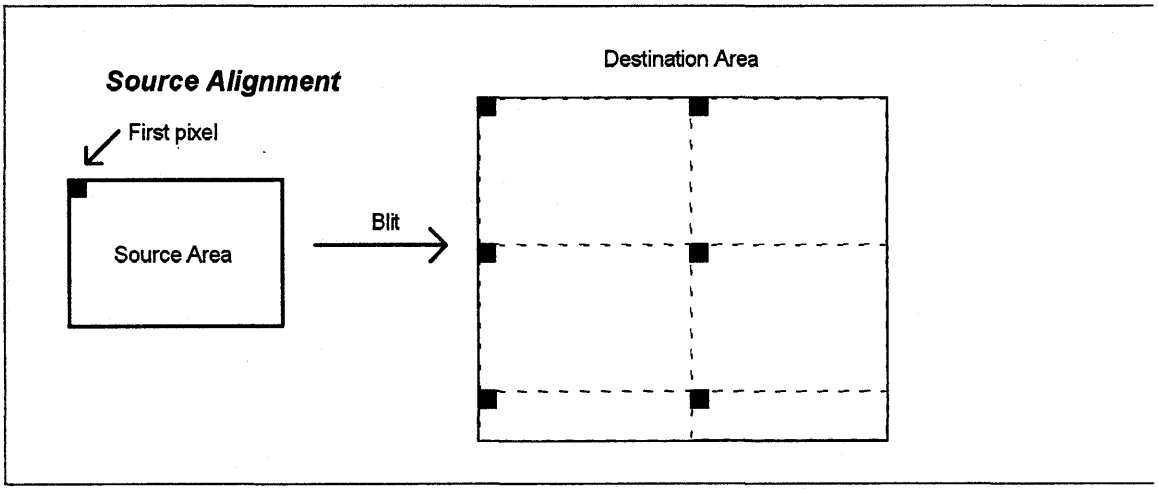
© 1993 ATI Technologies Inc.
Proprietary and Confidential

SOURCE AND DESTINATION ALIGNMENT

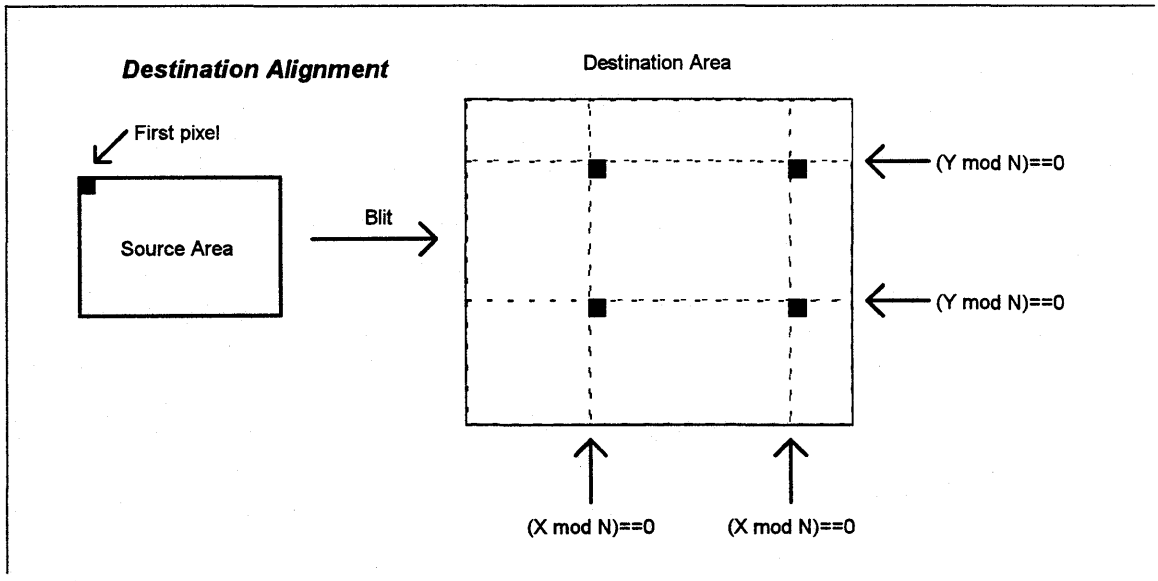
Sources may have one of two possible alignments:

- **Source alignment**
- **Destination alignment.**

Source alignment means that the top left corner of the source area is aligned to the top left corner of the destination area, as illustrated below:



Destination alignment to the Nth pixel means that the top left corner of the source area is aligned to (X mod N)==0 and (Y mod N)==0:



© 1993 ATI Technologies Inc.
Proprietary and Confidential

Various sources and their implicit alignments are listed below:

Source	Alignment
DP_FRGD_CLR	Destination aligned
DP_BKGD_CLR	Destination aligned
Fixed 8x8 mono pattern	Destination aligned X8 Y8
Fixed 4x2 color pattern	Destination aligned X4 Y2
Fixed 8x1 color pattern	Destination aligned X8
Mono host	Source aligned
Color host	Source aligned
Any blit source (strictly linear, unbounded Y, general pattern, general pattern with rotation)	Source aligned

The strict definition of **source alignment** is that a QWORD (or DWORD, depending on memory type and size) for a source-aligned source is rotated to align with the destination. No rotation occurs for destination-aligned sources.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SOURCE AND DESTINATION MIXING LOGIC

The available mix functions and compare functions are listed in the tables below. The ALU will mix the source and destination data to any of the functions listed. More complex functions may be accomplished with multiple passes.

The comparison functions compare a color register against either the destination data at the current pixel, or the color source data before processing by the ALU.

- If the result of the comparison is FALSE, the result of the ALU is written to the destination; otherwise, the destination data is written to the destination.

Mix Functions	
0	not D
1	0
2	1
3	D
4	not S
5	D xor S
6	(not D) xor S
7	S
8	(not D) or (not S)
9	D or (not S)
A	(not D) or S
B	D or S
C	D and S
D	(not D) and S
E	D and (not S)
F	(not D) and (not S)
17	(D+S) >> 1

Comparison functions	
0	False
1	True
2	Reserved
3	Reserved
4	Pixel != CLR_CMP_COLOR
5	Pixel == CLR_CMP_COLOR
6	Reserved
7	Reserved

Function 0x17 additionally requires the DP_CHAIN_MASK register to be set. Each '1' in the mask will prevent the carry bit from that bit position from adding to the next bit. The register should be set according to the following table:

4bpp	0x8888
8bpp	0x8080
15bpp	0x4210
16bpp	0x8410
24bpp	0x8080
32bpp	0x8080

© 1993 ATI Technologies Inc.
Proprietary and Confidential

DRAW ENGINE CONTEXTS

The *mach64* can save its complete draw engine state to memory. A saved state in memory is a **context**. More than one context can be saved in memory — in fact, the number of contexts that can be saved is limited only by the amount of memory on the graphics card. Contexts can later be loaded using a single draw engine command. The load can be of two types:

- Load the context.
- Load the context, and initiate a draw operation.

Chained contexts

Contexts may also be chained together. When a chained context is loaded, each individual context is loaded and initiated in turn. A chained context therefore acts like a single, atomic draw operation.

Contexts may be defined and restored from anywhere in screen memory, on any 64 DWORD boundary. The context structure is shown in the following table:

DWORD Offset	Register
0	CONTEXT_MASK
1	Reserved
2	DST_OFF_PITCH
3	DST_Y_X
4	DST_HEIGHT_WIDTH
5	DST_BRES_ERR
6	DST_BRES_INC
7	DST_BRES_DEC
8	SRC_OFF_PITCH
9	SRC_Y_X
A	SRC_HEIGHT1_WIDTH1
B	SRC_Y_X_START
C	SRC_HEIGHT2_WIDTH2
D	PAT_REG0
E	PAT_REG1
F	SC_LEFT_RIGHT
10	SC_TOP_BOTTOM
11	DP_BKGD_CLR
12	DP_FRGD_CLR
13	DP_WRITE_MASK
14	DP_CHAIN_MASK
15	DP_PIX_WIDTH
16	DP_MIX
17	DP_SRC
18	CLR_CMP_CLR
19	CLR_CMP_MASK
1A	CLR_CMP_CNTL
1B	GUI_TRAJ_CNTL
1C	CONTEXT_LOAD_CNTL
1D-3F	Reserved

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Notes:

DST_BRES_LNTH is actually aliased to the DST_WIDTH register.

The CONTEXT_MASK register does not affect the CONTEXT_MASK entry or the CONTEXT_LOAD_CNTL entry on loading.

The CONTEXT_LOAD_CNTL must be set to no-op to stop it from chaining.

Context save operators are not currently supported.

Contexts are allocated in reverse order in screen memory. Lower screen memory addresses correspond to higher context numbers. For instance, context number 0 resides at the top of screen memory minus 64 DWORDS, context 1 resides at top of memory minus 128 DWORDS, etc.

Context loads can selectively mask out any DWORD. Bit 0 of the context mask register corresponds to DWORD 0 of the context, etc. Setting any bit to zero in the context mask will inhibit loading of the corresponding DWORD, except the mask bits for CONTEXT_MASK and CONTEXT_LOAD_CNTL are ignored and these registers are always loaded. The CONTEXT_MASK entry which is loaded controls the remaining DWORDs to be loaded. The CONTEXT_LOAD_CNTL entry must be set to no-op to halt a context chain.

One possible context usage methodology is to set up various contexts for all possible "flavors" of draw operation at initialization time. It would be a simple matter to load a context, set up the trajectory registers, and initiate the draw operation. A load context simplifies the task of setting all the context registers manually. See *Saving and Restoring a Context* in *Chapter 3,Simple Draw Operations.*

Contexts may also be used to improve concurrency (see *Performance Issues* in *Chapter 5, Advanced Topics II*). If the draw engine is required to perform many small draw operations, these operation contexts may be written directly to screen memory and chained together. A single context load initiator is used to execute the entire chain. See *Context Chains* in *Chapter 5, Advanced Topics I.*

© 1993 ATI Technologies Inc.
Proprietary and Confidential

HARDWARE CURSOR

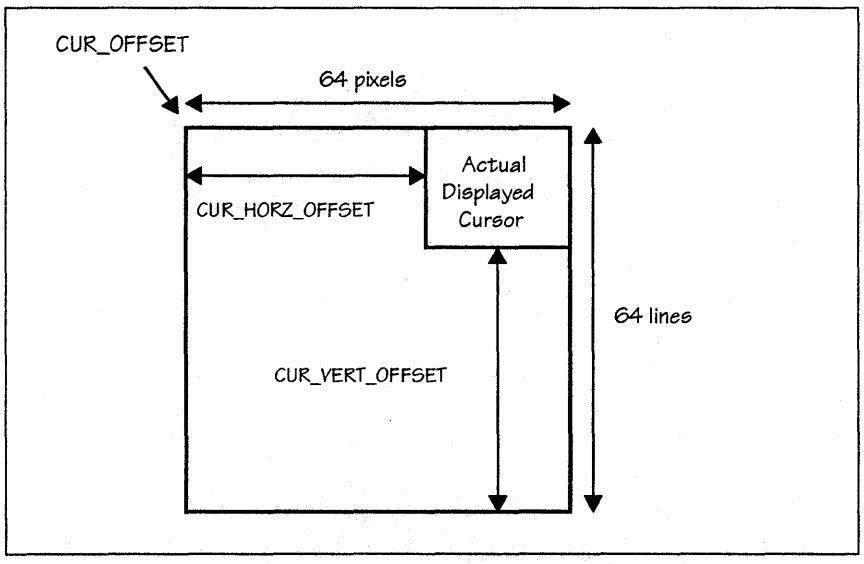
The *mach64* hardware cursor is similar in function to the *mach32* hardware cursor. Each cursor pixel is defined by a 2-bit field with the definition below:

Pixel Value	Meaning
00	Cursor color 0
01	Cursor color 1
10	Transparent
11	Complement

Cursor pitch is always 64 pixels: that is, each scan line of the hardware cursor definition is defined with 64*2 bits (16 bytes) of data, regardless of the actual cursor width. The pixel definition is specified in Intel order: the first pixel is defined in the low-order 2 bits of the low-order byte in memory. Each cursor scan line definition resides back-to-back in memory.

Pseudo color modes and Direct color modes

Cursor colors are defined by CUR_CLR0 and CUR_CLR1. Note that for *pseudo color modes*, the colors are specified in color indices, and for *direct color modes*, the colors are specified in 24-bit true color. The meaning of other registers is illustrated below:



The screen position of the top left corner of the displayed cursor is specified by CUR_HORZ_VERT_POSN. Care must be taken when the cursor hot spot is not the top left corner and the physical cursor position becomes negative. The *mach64* will not display the cursor at all if either the horizontal or vertical cursor position is negative.

- **If X becomes negative**, the cursor manager must adjust the CUR_HORZ_OFFSET to a larger number and saturate CUR_HORZ_POSN to zero.
- **If Y becomes negative**, CUR_VERT_OFFSET must be adjusted to a larger number, CUR_OFFSET must be adjusted to point to the appropriate line in the cursor definition, and CUR_VERT_POSN must be saturated to zero.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Sample code CURSOR.DOC

```
This is sample code to setup a hardware cursor bitmap. For more details,
see CURSOR.C and HWCURSOR.C.


/* 32x32 Cursor bitmap - 32 x 4 words/line */
unsigned int cursor32x32[128] =
{
                0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0xaa55, 0x55aa, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0x5555, 0x5555, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0xff00, 0xffff, 0xffff, 0x00ff,
                0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000,
                0x0000, 0x0000, 0x0000, 0x0000
};


----

/* Main C program */

int main(void)
{
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    ...

    // The hardware cursor definition requires 2 bits per pixel. The cursor
    // size regardless of the visible cursor size is 64 pixels x 64 lines.
    // The total memory required to define the cursor is 16 bytes / line
    // for 64 lines or 1024 bytes of data. The data must be in a contigious
    // format. The 2 bit cursor code values are as follows:
    //
    //   00 - pixel color = CURSOR_COLOR_0
    //   01 - pixel color = CURSOR_COLOR_1
    //   10 - pixel color = transparent (current display pixel)
    //   11 - pixel color = 1's complement of current display pixel


    // ---- Use 16 bpp to draw cursor data information ----
    //
    // By programming the starting address ('offset'), all destination draw
    // operations will start at this address. This is useful for setting up
    // data that is resolution and color depth independent.

    // The cursor data area will be first filled with 'transparent code'
    // data. Cursor data that is transparent will not be visible when the
    // hardware cursor is enabled. Secondly, the cursor pattern will be drawn
    // to the appropriate area within the cursor data area. Note that the
    // cursor data is in linearized format. It can be thought of as a
    // rectangle whose lines are joined contigiously together in memory.

    // setup a 32x32 hardware cursor - the cursor data will be visible
    set_hwcursor(modeinfo.yres - 10, 32, 32,
                 get_color_code(LIGHTBLUE),      // cursor color 0
                 get_color_code(YELLOW),         // cursor color 1
                 cursor32x32);

    // set hardware cursor to center of screen
    set_hwcursor_pos((modeinfo.xres / 2) - (32 / 2),
                     (modeinfo.yres / 2) - (32 / 2));

    // enable hardware cursor
    enable_hwcursor();


    ...

}


/* ---------------------------------------------------------------------
  SET_HWCURSOR - define hardware cursor bitmap

  This function sets up the hardware cursor data region according to the
  given bitmap data. The data region is located at (0, y).
```

```
  The hardware cursor be vary in size from 1x1 to 64x64. The expected bitmap
  format is LSB to MSB. The LSB will be drawn first in a left to right
  direction. Since the cursor position is NOT set in this routine, the
  position should be set (set_hwcursor_pos) prior to enabling the cursor.
---------------------------------------------------------------------------- */
void set_hwcursor(int y, int width, int height,
                  unsigned long color0, unsigned long color1,
                  unsigned int *bitmap)
{
    unsigned long cur_offset, cur_size, cur_pitch;
    unsigned long temp1, temp2, temp3, temp4, temp5, temp6;
    unsigned long red1, green1, blue1, red2, green2, blue2;
    unsigned long rshift, gshift, bshift;
    unsigned long bitdata;
    int i, index, dataindex, start, widthwords;
    PALETTE entry;

    // Check that cursor size is within limits
    if ((width < 1) || (width > 64)) return;
    if ((height < 1) || (height > 64)) return;

    // set cursor dimensions
    cursordata.y = y;
    cursordata.width = width;
    cursordata.height = height;
    cursordata.color0 = color0;
    cursordata.color1 = color1;

    // set hwcursor bitmap to transparent
    for (index = 0; index < (HWCURHEIGHT * HWCURWIDTH); index++)
                {
        cursordata.bitmap[index] = 0xaaaa;
                }

    // load user hwcursor data into bitmap
    dataindex = 0;
    widthwords = width / 8;
    if (width > widthwords * 8)
    {
        widthwords++;
    }
    start = HWCURWIDTH - widthwords;
   for (index = start; index < (HWCURWIDTH * height); index = index + HWCURWIDTH)
                {
        i = 0;
        do
        {
            cursordata.bitmap[index + i] = *(bitmap + dataindex);
            if (width < 8)
            {
                cursordata.bitmap[index + i] = cursordata.bitmap[index + i] <<
((8 - width) * 2);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        }
        dataindex++;
        i++;
    } while (i < widthwords);
}

// calculate offset in bytes
cur_offset = (unsigned long)(modeinfo.pitch);
if (modeinfo.bpp == 4)
{
    cur_offset = (unsigned long)(cur_offset / 2);
}
else
{
    cur_offset = (unsigned long)(cur_offset * (modeinfo.bpp / 8));
}
cur_offset = (unsigned long)(cur_offset * y);

// calculate cursor pitch (assuming 16 bpp)
cur_pitch = HWCURWIDTH / 8;
cur_pitch = cur_pitch << 22;

// convert byte offset to qword offset and limit to 20 bits (used DWORD
//   format used in DST_OFF_PITCH
cur_offset = (cur_offset / 8) & 0x000fffff;

// Use 16 bpp to setup hardware cursor bitmap

// save vital registers
wait_for_idle();
temp1 = regr(DP_PIX_WIDTH);
temp2 = regr(DP_CHAIN_MASK);
temp3 = regr(DST_OFF_PITCH);
temp4 = regr(DP_SRC);
temp5 = regr(DP_MIX);
temp6 = regr(DST_CNTL);

// load bitmap data to hardware cursor bitmap data area
regw(DP_PIX_WIDTH, (temp1 & 0xffc00000) | BYTE_ORDER_LSB_TO_MSB |
                   HOST_16BPP | SRC_16BPP | DST_16BPP);
regw(DP_CHAIN_MASK, 0x0410);  // chain mask for 16 bpp
regw(DST_OFF_PITCH, cur_pitch | cur_offset);
regw(DP_SRC, FRGD_SRC_HOST);
regw(DP_MIX, FRGD_MIX_S | BKGD_MIX_D);
regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);
regw(DST_X, 0);
regw(DST_Y, 0);
regw(DST_HEIGHT, HWCURHEIGHT);
regw(DST_WIDTH, HWCURWIDTH);
for (index = 0; index < (HWCURHEIGHT * HWCURWIDTH * 2); index = index + 2)
{
    wait_for_fifo(1);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        bitdata = (unsigned long)(cursordata.bitmap[index+1]);
        bitdata = (bitdata << 16) | (cursordata.bitmap[index]);
        regw(HOST_DATA0, bitdata);
    }
    wait_for_idle();

    // set cursor size offsets
    cur_size = (unsigned long)(64 - height);
    cur_size = (unsigned long)((cur_size << 16) | (64 - width));
    regw(CUR_HORZ_VERT_OFF, cur_size);

    // set cursor colors
    if (modeinfo.bpp > 8)
    {
        // for 15, 16, 24, 32 color modes
        switch(modeinfo.bpp)
        {
            case 16:
                if (modeinfo.depth == 555)
                {
                    rshift = 3;
                    gshift = 3;
                    bshift = 3;
                }
                else  // 565 weight
                {
                    rshift = 3;
                    gshift = 2;
                    bshift = 3;
                }
                break;

            case 24:
            case 32:
                rshift = 0;
                gshift = 0;
                bshift = 0;
                break;
        }

        // cursor color 0
        red1 = get_primary_color(RED, color0) << rshift;
        green1 = get_primary_color(GREEN, color0) << gshift;
        blue1 = get_primary_color(BLUE, color0) << bshift;

        // cursor color 1
        red2 = get_primary_color(RED, color1) << rshift;
        green2 = get_primary_color(GREEN, color1) << gshift;
        blue2 = get_primary_color(BLUE, color1) << bshift;

        if (querydata.dac_type != DAC_ATI68860)
        {
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            // standard setup for other dacs
            regw(CUR_CLR0, (red1 << 24) | (green1 << 16) | (blue1 << 8));
            regw(CUR_CLR1, (red2 << 24) | (green2 << 16) | (blue2 << 8));
        }
    }
    else
    {
        // for 4, 8 bpp color modes
        if (querydata.dac_type == DAC_ATI68860)
        {
            entry = get_palette((int)(color0 & 0xff));
            red1 = (unsigned long)(entry.red);
            green1 = (unsigned long)(entry.green);
            blue1 = (unsigned long)(entry.blue);
            entry = get_palette((int)(color1 & 0xff));
            red2 = (unsigned long)(entry.red);
            green2 = (unsigned long)(entry.green);
            blue2 = (unsigned long)(entry.blue);
        }
        else
        {
            regw(CUR_CLR0, color0 & 0xff);
            regw(CUR_CLR1, color1 & 0xff);
        }
    }

    if (querydata.dac_type == DAC_ATI68860)
    {
        // special setup for ATI68860/880 dac for cursor colors
        outp(ioDAC_CNTL, 1);
        outp(ioDAC_REGS, 0);

        outp(ioDAC_REGS+1, (int)red1);
        outp(ioDAC_REGS+1, (int)green1);
        outp(ioDAC_REGS+1, (int)blue1);

        outp(ioDAC_REGS+1, (int)red2);
        outp(ioDAC_REGS+1, (int)green2);
        outp(ioDAC_REGS+1, (int)blue2);

        outp(ioDAC_CNTL, 0);
    }

    // set offset to cursor data region
    regw(CUR_OFFSET, cur_offset);

    // restore vital registers
    regw(DP_PIX_WIDTH, temp1);
    regw(DP_CHAIN_MASK, temp2);
    regw(DST_OFF_PITCH, temp3);
    regw(DP_SRC, temp4);
    regw(DP_MIX, temp5);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    regw(DST_CNTL, temp6);
}

/* ------------------------------------------------------------------------
  ENABLE_HWCURSOR - turn on the hardware cursor
------------------------------------------------------------------------ */
void enable_hwcursor(void)
{
    // enable hardware cursor
    outpw(ioGEN_TEST_CNTL, inpw(ioGEN_TEST_CNTL) | 0x80);
}

/* ------------------------------------------------------------------------
  SET_HWCURSOR_POS - set the hardware cursor position
------------------------------------------------------------------------ */
void set_hwcursor_pos(int x, int y)
{
    unsigned long cur_pos;

    // check for coordinate violations
    if (x < 0) x = 0;
    if (y < 0) y = 0;

    // set cursor position
    cur_pos = (unsigned long)y;
    cur_pos = (unsigned long)((cur_pos << 16) | x);
    regw(CUR_HORZ_VERT_POSN, cur_pos);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

DRAW OPERATIONS

There are two draw operations available on the *mach64*: **lines** and **rectangle fills** (**blits** are a subset of rectangle fills).

- **Blits** are initiated by writing to the DST_WIDTH, DST_X_WIDTH, or DST_HEIGHT_WIDTH registers.
- **Lines** are initiated by writing to the DST_BRES_LNTH register. Lines or blits may also be initiated by a context load command.

Some possible programming operating styles are listed below:

Style 1, Drawing Without Using Contexts

1. Set up the draw engine context manually by writing to all the appropriate registers. A summary of these registers can be found in *Appendix E, Register Summary.*
2. Set up the draw trajectory by writing to the trajectory control registers.
3. Initiate the draw operation by writing to the appropriate draw initiator register.
4. Repeat steps 1 to 3 for each draw operation.

Sample code STYLE1.DOC

```
This is sample code to show programming style 1.

  1. Setup draw engine context manually
  2. Setup draw trajectory of draw operation
  3. Initiate draw operation by writing to the appropriate register
  4. Repeat steps 1 to 3 for each operation

See STYLE1.C and INIT.C for more details.


    // Step 1: Setup draw engine context manually -- and clear screen
    init_engine();
    clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

    // Step 2: Setup draw trajectory of draw operation - i.e. rectangle fill

    // draw a filled rectangle
    wait_for_fifo(5);
    regw(DP_FRGD_CLR, get_color_code(LIGHTGREEN));
    regw(DST_X, 0);
    regw(DST_Y, 0);
    regw(DST_HEIGHT, modeinfo.yres / 4);

    // Step 3: Initiate the draw operation - DST_WIDTH is a draw initiator
    regw(DST_WIDTH, modeinfo.xres / 4);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Style 2, Drawing Using Restored Contexts

1. At initialization time, reserve some screen memory, and save all possible contexts to that area. Contexts may be saved by writing directly to screen memory.
2. Load desired default draw context.
3. Set up trajectory registers.
4. Initiate the draw operation.
5. Repeat steps 2 to 4 for each draw operation.

Sample code STYLE2.DOC

```
This is sample code to show programming style 2.

  1. At initialization, store engine context(s) at appropriate addresses.
  2. Load desired default draw context.
  3. Setup draw trajectory registers.
  4. Initiate the draw operation.
  5. Repeat steps 2 to 4 for each operation

See STYLE2.C for more details.

----

unsigned long context[64];      // context array


    // Step 1: Store engine context at appropriate address to load later

    // determine top of memory address
    memcntl = inpw(ioMEM_CNTL);
    switch(memcntl & 7)
    {
        case 0: contextaddr = 0x80000; break;   // 512K
        case 1: contextaddr = 0x100000; break;  // 1M
        case 2: contextaddr = 0x200000; break;  // 2M
        case 3: contextaddr = 0x400000; break;  // 4M
        case 4: contextaddr = 0x600000; break;  // 6M
        case 5: contextaddr = 0x800000; break;  // 8M
    }

    /*
       Fill context for context pointer 4. Each context pointer represents
       256 bytes of video memory. The context load address decreases as the
       context load pointer increases. Context pointer 0 points to the top of
       memory - 256. Some restrictions apply for context pointers 0-3 if the
       linear aperture size equals the memory size (i.e. 4M aperture size,
       4M of video memory). In this case, the memory mapped registers occupy
       1K of memory below the top of aperture. The only method to reach this
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
   area in this case is to use the VGA paged aperture.
*/
context[0] = 0xffffffff;
context[1] = 0x00000000;
context[2] = (pitch / 8) << 22;          // destination pitch
context[3] = 0x00000000;
context[4] = 0x00000000;
context[5] = 0x00000000;
context[6] = 0x00000000;
context[7] = 0x00000000;
context[8] = (pitch / 8) << 22;          // source pitch
context[9] = 0x00000000;
context[10] = 0x00000000;
context[11] = 0x00000000;
context[12] = 0x00000000;
context[13] = 0x00000000;
context[14] = 0x00000000;
context[15] = xres << 16;                // scissors
context[16] = yres << 16;
context[17] = 0x00000000;
context[18] = 0xffffffff;
context[19] = 0xffffffff;

// set DP_PIX_WIDTH and CHAIN_MASK according to color depth
switch(modeinfo.bpp)
{
    case 4:
        context[20] = 0x00008888;
        context[21] = HOST_4BPP | SRC_4BPP | DST_4BPP;
        break;
    case 16:
        if (modeinfo.depth == 555)
        {
            // 555 color weighting
            context[20] = 0x00004210;
            context[21] = HOST_15BPP | SRC_15BPP | DST_15BPP;
        }
        else
        {
            // 565 color weighting
            context[20] = 0x00000410;
            context[21] = HOST_16BPP | SRC_16BPP | DST_16BPP;
        }
        break;
    case 32:
        context[20] = 0x00008080;
        context[21] = HOST_32BPP | SRC_32BPP | DST_32BPP;
        break;
    default:
    case 8:
    case 24:
        context[20] = 0x00008080;
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            context[21] = HOST_8BPP | SRC_8BPP | DST_8BPP;
            break;
    }

    context[22] = FRGD_MIX_S | BKGD_MIX_S;
    context[23] = FRGD_SRC_FRGD_CLR;
    context[24] = 0x00000000;
    context[25] = 0xffffffff;
    context[26] = 0x00000000;
    context[27] = DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT;
    context[28] = 0x00000000;
    for (i = 29; i < 64; i++)
    {
        context[i] = 0;
    }

    // Upload contexts

    /*
       The main focus here is to upload the context information for later
       retrieval by the engine context load feature. The method used will
       depend on the hardware setup - see STYLE2.C for more details.
    */

    // Context load address calculation:
    //
    // Address = total video memory - (context pointer + 1) * 0x100)
    //

    // load up first context -> load pointer = 4
    upload_context(contextaddr - ((4 + 1) * 0x100));

    // Step 2: Load default context (load context pointer 4)

    // load all GUI registers
    wait_for_fifo(2);
    regw(CONTEXT_MASK, 0xffffffff);

    // load using context pointer 4
    regw(CONTEXT_LOAD_CNTL, CONTEXT_LOAD | 4);

    // Step 3: Setup draw trajectory of draw operation - i.e. rectangle fill

    // set foreground color
    wait_for_fifo(5);
    regw(DP_FRGD_CLR, get_color_code(LIGHTCYAN));

    // draw a filled rectangle
    regw(DST_X, modeinfo.xres / 32);
    regw(DST_Y, modeinfo.yres / 32);
    regw(DST_HEIGHT, modeinfo.yres / 4);

    // Step 4: Initiate the draw operation - DST_WIDTH is a draw initiator
    regw(DST_WIDTH, modeinfo.xres / 2);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Style 3, Drawing Using Context Chains

1. Configure all draw operations by writing context chains directly to screen memory.
2. Do a load context with a draw initiate.
3. Repeat steps 1 to 2 for each sequence of draw operations.

Sample code STYLE3.DOC

```
This is sample code to show programming style 3.

  1. Store pre-set engine draw context at appropriate address.
  2. Load pre-set engine draw context with a draw initiate command.
  3. Repeat steps 1 to 2 for each operation

See STYLE2.3 for more details.

----

unsigned long context[64];      // context array


    // Step 1: Store engine context at appropriate address to load later

    // determine top of memory address
    memcntl = inpw(ioMEM_CNTL);
    switch(memcntl & 7)
    {
        case 0: contextaddr = 0x80000; break;   // 512K
        case 1: contextaddr = 0x100000; break;  // 1M
        case 2: contextaddr = 0x200000; break;  // 2M
        case 3: contextaddr = 0x400000; break;  // 4M
        case 4: contextaddr = 0x600000; break;  // 6M
        case 5: contextaddr = 0x800000; break;  // 8M
    }

    /*
       Fill context for context pointer 4. Each context pointer represents
       256 bytes of video memory. The context load address decreases as the
       context load pointer increases. Context pointer 0 points to the top of
       memory - 256. Some restrictions apply for context pointers 0-3 if the
       linear aperture size equals the memory size (i.e. 4M aperture size,
       4M of video memory). In this case, the memory mapped registers occupy
       1K of memory below the top of aperture. The only method to reach this
       area in this case is to use the VGA paged aperture.
    */
    context[0] = 0xffffffff;
    context[1] = 0x00000000;
    context[2] = (pitch / 8) << 22;                    // destination pitch
    temp = (unsigned long)(modeinfo.xres / 32);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
context[3] = (temp << 16) | (modeinfo.yres / 32);  // (x, y)
temp = (unsigned long)(modeinfo.xres / 8);
context[4] = (temp << 16) | (modeinfo.yres / 4);   // (width, height)
context[5] = 0x00000000;
context[6] = 0x00000000;
context[7] = 0x00000000;
context[8] = (pitch / 8) << 22;                    // source pitch
context[9] = 0x00000000;
context[10] = 0x00000000;
context[11] = 0x00000000;
context[12] = 0x00000000;
context[13] = 0x00000000;
context[14] = 0x00000000;
context[15] = xres << 16;                          // scissors
context[16] = yres << 16;
context[17] = 0x00000000;
context[18] = get_color_code(LIGHTBLUE);           // foreground color
context[19] = 0xffffffff;

// set DP_PIX_WIDTH and CHAIN_MASK according to color depth
switch(modeinfo.bpp)
{
    case 4:
        context[20] = 0x00008888;
        context[21] = HOST_4BPP | SRC_4BPP | DST_4BPP;
        break;
    case 16:
        if (modeinfo.depth == 555)
        {
            // 555 color weighting
            context[20] = 0x00004210;
            context[21] = HOST_15BPP | SRC_15BPP | DST_15BPP;
        }
        else
        {
            // 565 color weighting
            context[20] = 0x00000410;
            context[21] = HOST_16BPP | SRC_16BPP | DST_16BPP;
        }
        break;
    case 32:
        context[20] = 0x00008080;
        context[21] = HOST_32BPP | SRC_32BPP | DST_32BPP;
        break;
    default:
    case 8:
    case 24:
        context[20] = 0x00008080;
        context[21] = HOST_8BPP | SRC_8BPP | DST_8BPP;
        break;
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
context[22] = FRGD_MIX_S | BKGD_MIX_S;
context[23] = FRGD_SRC_FRGD_CLR;
context[24] = 0x00000000;
context[25] = 0xffffffff;
context[26] = 0x00000000;
context[27] = DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT;
context[28] = 0x00000000;
for (i = 29; i < 64; i++)
{
    context[i] = 0;
}

// Upload contexts

/*
   The main focus here is to upload the context information for later
   retrieval by the engine context load feature. The method used will
   depend on the hardware setup - see STYLE2.C for more details.
*/

// Context load address calculation:
//
// Address = total video memory - (context pointer + 1) * 0x100)
//

// load up first context -> load pointer = 4
upload_context(contextaddr - ((4 + 1) * 0x100));

// Step 2: Load context with blit draw initiate (load context pointer 4)

// load all GUI registers
wait_for_fifo(2);
regw(CONTEXT_MASK, 0xffffffff);

// load & initiate using context pointer 4
regw(CONTEXT_LOAD_CNTL, CONTEXT_LOAD_AND_DO_FILL | 4);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

REMARKS ON PIXEL DEPTH

Not all pixel depths are created equal:

- 1 bpp mode is supported by the drawing engine, but not by the CRT controller. Therefore, 1 bpp mode can only be used in off-screen memory.
- Pitch is normally specified in multiples of 8. An additional restriction is that it must also fall on a 64-bit boundary. That implies that pitch for 1bpp mode must be a multiple of 64, and pitch for 4bpp mode must be a multiple of 16.
- The DP_BYTE_PIX_ORDER@DP_PIX_WIDTH bit only affects pixel ordering within a byte. Therefore, only 1bpp and 4bpp modes are affected.
- All pixel depths above 8bpp are direct color modes. 4bpp and 8bpp modes are pseudocolor modes.
- Packed 24 bpp mode is actually 24bpp CRTC mode and 8bpp draw mode with special rotations done on DP_FRGD_CLR, DP_BKGD_CLR, DP_WRITE_MASK, and fixed 8x8 mono patterns. See *Drawing in Packed 24 Bit Per Pixel Modes* in *Chapter 4, Advanced Topics I.*
- DP_CHAIN_MASK must be manually set for the destination pixel depth (this register only affects the mix function 0x17, the averaging function). The following table lists the settings:

Pixel Depth	DP_CHAIN_MASK
1	N/A
4	0x8888
8	0x8080
8 (RGB 332)	0x9292
15 (RGB 555)	0x4210
16 (RGB 565)	0x8410
packed 24	0x8080
32	0x8080

- 15bpp and 16bpp modes are identical draw modes, but different DAC modes must be set (use BIOS services for mode switching so the application doesn't have to handle it). 15bpp mode is always RGB 555, and 16bpp mode is always RGB 565.
- Although pixel depths for source area, destination area, and host may be set independently, the only pixel depth conversion available is 1bpp to any pixel depth monochrome expansion. Behavior is undefined for any other mixing and matching of pixel depths.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

VGA INTERACTION

Remember that physical memory is shared between the on-chip VGA and the accelerator. A logical boundary may be enabled with the MEM_CNTL register to inhibit the two logical devices from accessing the other's memory.

- **When the memory boundary is disabled,** each device has full access to on-board memory.
- **When the memory boundary is enabled,** any memory accesses through the VGA aperture are inhibited. All draw engine functions that access the memory below the boundary are inhibited. The boundary may be set to zero. Remember to set all draw engine offsets above the memory boundary.
- Memory accesses through the big linear aperture are not affected by the memory boundary register.
- If the application destroys VGA memory, the application must re-initialize the VGA mode before exiting.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Chapter 3
Simple Draw Operations

SAVING AND RESTORING A CONTEXT

Context pointers begin at the high address in screen memory and proceed downward. This facilitates any stack-based implementations of context saving and restoring. To calculate the byte address of a context load location:

byte_address = ((memory_size / 256 - 1) - context_number) * 256

The exact mapping of a context block is shown in *Draw Engine Contexts* in *Chapter 2, Programming Model.* Sample code is provided in *Draw operations* in *Chapter 2, Programming Model.*

RECTANGLE FILL

A rectangle fill is a **draw operation** that has a rectangular destination trajectory. Any source may be used.

Sample code

```
This is sample code to draw a filled rectangle. See RECT.C and DRAW.C for
more details.


    // Draw a filled rectangle at (x, y) of size (width x height) using
    // the foreground color as the source.

    wait_for_fifo(7);
    regw(DP_FRGD_CLR, get_color_code(LIGHTBLUE));  // color depth independent
    regw(DP_SRC, FRGD_BLIT_SRC);
    regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

    regw(DST_X, x);
    regw(DST_Y, y);
    regw(DST_HEIGHT, height);
    regw(DST_WIDTH, width);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

BITBLT

A bitblt is a **rectangle fill** that specifically uses a color blit source. There are four types of blit source trajectory, as described in *Trajectories* in *Chapter 2, Programming Model*. Note that the source trajectory direction always tracks the destination trajectory direction. Blit sources are always source-aligned.

Simple One-To-One Bitblt

Sample code SBLIT.DOC

```
This is sample code to show a simple one-to-one blit. See SBLIT.C and DRAW.C
for more details.


    // Draw some filled rectangles which will be used as the blit source at
    // (srcx, srcy) of size (srcwidth, srcheight). These are resolution
    // independent.

    set_fg_color(get_color_code(WHITE));
    draw_rectangle(srcx, srcy, srcwidth, srcheight);

    set_fg_color(get_color_code(LIGHTRED));
    draw_rectangle(srcx + (modeinfo.xres / 32),
                   srcy + (modeinfo.yres / 32),
                   srcwidth - (2 * (modeinfo.xres / 32)),
                   srcheight - (2 * (modeinfo.yres / 32)));

    set_fg_color(get_color_code(LIGHTBLUE));
    draw_rectangle(srcx + (modeinfo.xres / 16),
                   srcy + (modeinfo.yres / 16),
                   srcwidth - (2 * (modeinfo.xres / 16)),
                   srcheight - (2 * (modeinfo.yres / 16)));


    // Perform a simple one-to-one color blit

    wait_for_fifo(11);

    regw(DP_SRC, FRGD_SRC_BLIT);
    regw(SRC_CNTL, 0);
    regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

    regw(SRC_X, srcx);
    regw(SRC_Y, srcy);
    regw(SRC_HEIGHT1, srcheight);
    regw(SRC_WIDTH1, srcwidth);

    regw(DST_X, dstx);
    regw(DST_Y, dsty);
    regw(DST_HEIGHT, srcheight);
    regw(DST_WIDTH, srcwidth);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

General Pattern

Sample code GPBLIT.DOC

This is sample code to show a general pattern blit. See GPBLIT.C for more details.

```
// Setup dimensions of source and destination

x1 = modeinfo.xres / 64;          // source
y1 = modeinfo.yres / 48;
width1 = modeinfo.xres / 16;
height1 = modeinfo.yres / 12;

x2 = modeinfo.xres / 8;           // destination
y2 = modeinfo.yres / 16;
width2 = modeinfo.xres / 4;
height2 = modeinfo.yres / 3;

// draw some filled rectangles which will be used as the blit source
set_fg_color(get_color_code(WHITE));
draw_rectangle(x1, y1, width1, height1);

set_fg_color(get_color_code(LIGHTBLUE));
draw_rectangle(x1+4, y1+4, width1-8, height1-8);

// Draw a general pattern blit:
//
//  The general pattern blit allows the source to wrap in the X & Y
//  directions while continuing to draw to the destination
//


// set source to blit
wait_for_fifo(11);
regw(DP_SRC, FRGD_SRC_BLIT);

// use general pattern for source
regw(SRC_CNTL, SRC_PATTERN_ENABLE);
regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

// perform general pattern blit:
//
//   note that destination rectangle is larger than source rectangle
//

regw(SRC_X, x1);
regw(SRC_Y, y1);
regw(SRC_HEIGHT1, height1);
regw(SRC_WIDTH1, width1);

regw(DST_X, x2);
regw(DST_Y, y2);
regw(DST_HEIGHT, height2);
regw(DST_WIDTH, width2);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

General Pattern With Rotation

Sample code GPRBLIT.DOC

See source file GPRBLIT.C for more details.

```
    // Draw a patterned source made up from several filled rectangles
    draw_rectangle(10, 10, 40, 40, WHITE);
    draw_rectangle(14, 14, 32, 32, LIGHTBLUE);
    draw_rectangle(18, 18, 24, 24, LIGHTGREEN);
    draw_rectangle(22, 22, 16, 16, DARKRED);

    // draw a general pattern blit:
    //  the general pattern blit allows the source to wrap in the X & Y
    //  directions while continuing to draw to the destination
    waitforfifo(15);            // wait for 15 fifo entries
                                // set source to blit
    regw(DP_SRC, FRGD_SRC_BLIT);
                                // use general pattern with rotation for src
    regw(SRC_CNTL, SRC_ROTATION_ENABLE | SRC_PATTERN_ENABLE);
    regw(SRC_X, 18);
    regw(SRC_Y, 18);
    regw(SRC_HEIGHT1, 24);
    regw(SRC_WIDTH1, 24);
    regw(SRC_X_START, 10);
    regw(SRC_Y_START, 10);
    regw(SRC_HEIGHT2, 40);
    regw(SRC_WIDTH2, 40);
                                // wrap source 4 times in X & Y directions
                                //  with source pattern rotation
    regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);
    regw(DST_X, 100);
    regw(DST_Y, 50);
    regw(DST_HEIGHT, 160);
    regw(DST_WIDTH, 160);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Monochrome Expansion Bitblt

Sample code MEBLIT.DOC

This is sample code to show a monochrome to color expansion blit. See MEBLIT.C for more details.

```
    // Setup source and destination coordinates and sizes
    dstx = 0;
    dsty = 4;
    dstwidth = 16;
    dstheight = 20;

    srcx = 0;
    srcy = 0;
    srcwidth = dstwidth * dstheight;
    srcheight = 1;

    /* ---- Setup a monochrome source pattern ---- */

    // This is done by drawing host data linearly in memory. The pattern will
    // be placed at (0, 0) so that it is visible on the screen. The linear
    // source data will be monochrome expanded into a 16x20 rectangle (320
    // bits). A "1" bit in the source data will be expanded to the foreground
    // color and a "0" will be expanded to the background color. The
    // resultant pattern will be a blue box with a white diamond and border.

    // monochrome host, src, dst
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_1BPP | SRC_1BPP | DST_1BPP);

    // src = host data
    regw(DP_SRC, FRGD_SRC_HOST);

    // overpaint mix for src, dst
    set_fg_mix(S_MIX);
    set_bg_mix(S_MIX);

    // setup source rectangle to be filled with monochrome host data
    wait_for_fifo(4);
    regw(DST_X, 0);
    regw(DST_Y, 0);
    regw(DST_HEIGHT, 1);
    regw(DST_WIDTH, 320);

    // copy host data to memory - pattern order is LSB to MSB
    wait_for_fifo(10);
    regw(HOST_DATA0, 0x8001FFFF);
    regw(HOST_DATA0, 0x81818181);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
regw(HOST_DATA0, 0x83C183C1);
regw(HOST_DATA0, 0x87E187E1);
regw(HOST_DATA0, 0x9FF98FF1);
regw(HOST_DATA0, 0x8FF19FF9);
regw(HOST_DATA0, 0x87E187E1);
regw(HOST_DATA0, 0x83C183C1);
regw(HOST_DATA0, 0x81818181);
regw(HOST_DATA0, 0xFFFF8001);

// insure host transfer is done
wait_for_idle();

// use 8 bpp for this example
regw(DP_PIX_WIDTH, BYTE_ORDER_LSB_TO_MSB |
                   HOST_1BPP | SRC_1BPP | DST_8BPP);

// blit host, src = frgd
regw(DP_SRC, MONO_SRC_BLIT | FRGD_SRC_FRGD_CLR);

// foreground color = WHITE
set_fg_color(get_color_code(WHITE));

// background color = LIGHTBLUE
set_bg_color(get_color_code(LIGHTBLUE));


/* ---- Do monochrome expansion blit ---- */

// src data is in linear format
wait_for_fifo(10);
regw(SRC_CNTL, SRC_LINEAR_ENABLE);
regw(SRC_X, srcx);
regw(SRC_Y, srcy);
regw(SRC_HEIGHT1, srcheight);
regw(SRC_WIDTH1, srcwidth);

regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);
regw(DST_X, dstx);
regw(DST_Y, dsty);
regw(DST_HEIGHT, dstheight);
regw(DST_WIDTH, dstwidth);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Line Patterns

When the destination trajectory is a line, the source trajectory behaves in almost the same fashion as for a rectangular destination trajectory. The only differences are:

- The source trajectory never advances in the Y direction (the source height is implicitly equal to one).
- The source trajectory X direction is independent of the destination X direction, and is settable by SRC_LINE_X_DIR@SRC_CNTL.

Sample code GPLINE.DOC

```
This is sample code to show drawing a line using a pattern (general pattern
line). See GPLINE.C for more details.


    // Setup line source, destination, and lengths

    // source line (a rectangle with a height of 1) at (srcx, srcy)
    srcx = 0;
    srcy = 0;
    src_line_length = modeinfo.xres / 20;

    // first line starting coordinate at (dstx1, dsty1)
    dstx1 = modeinfo.xres / 40;
    dsty1 = modeinfo.yres / 30;

    // second line starting coordinate at (dstx2, dsty2)
    dstx2 = modeinfo.xres / 20;
    dsty2 = modeinfo.yres / 30;

    // destination lines length
    dst_line_length = modeinfo.xres / 6;


    /* ---- Setup source line (rectangle with height of 1) ---- */

    // Setup a color source pattern from host data. The number of host data
    // writes will depend on the current pixel depth. Also, the host data
    // must be packed into 32 bit pieces (i.e. for 8bpp modes, each host
    // write draws 4 pixels).

    // Setup a rectangle fill with host data
    wait_for_fifo(6);
    regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);
    regw(DP_SRC, FRGD_SRC_HOST);    // src = host data
    regw(DST_X, srcx);
    regw(DST_Y, srcy);
    regw(DST_HEIGHT, 1);
    regw(DST_WIDTH, src_line_length);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// Fill source rectangle (height of 1) with packed host data - this
// function draws a line of source data of alterating colors - see
// GPLINE.C for more information.

host_fill(src_line_length);

// insure that host transfer is done
wait_for_idle();

// Draw a diagonal line using data from the source. When the src data
// runs out, the source pointer will wrap and repeat according to
// the source x direction line bit in SRC_CNTL. Both directions are
// demonstrated.

/* ---- Draw line using LEFT TO RIGHT source x direction ---- */

wait_for_fifo(13);
regw(DP_SRC, FRGD_SRC_BLIT);              // src = blit
regw(SRC_CNTL, SRC_LINE_X_LEFT_TO_RIGHT | SRC_PATTERN_ENABLE);
regw(SRC_X, srcx);
regw(SRC_Y, srcy);
regw(SRC_HEIGHT1, 1);
regw(SRC_WIDTH1, src_line_length);

// draw a diagonal line using blit src
regw(DST_CNTL, DST_LAST_PEL | DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);
regw(DST_BRES_ERR, 0);
regw(DST_BRES_INC, 1);
regw(DST_BRES_DEC, 0x3FFFF);
regw(DST_X, dstx1);
regw(DST_Y, dsty1);
regw(DST_BRES_LNTH, dst_line_length);     // src data will repeat


/* ---- Draw line using RIGHT TO LEFT source x direction ---- */

wait_for_fifo(6);
regw(SRC_CNTL, SRC_LINE_X_RIGHT_TO_LEFT | SRC_PATTERN_ENABLE);
regw(SRC_X, src_line_length - 1);
regw(SRC_Y, srcy);

regw(DST_X, dstx2);
regw(DST_Y, dsty2);
regw(DST_BRES_LNTH, dst_line_length);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

FIXED PATTERNS

Three types of fixed pattern are available:

- **4x2 color pattern**
- **8x1 color pattern**
- **8x8 monochrome pattern.**

The fixed color patterns are only supported in 8bpp mode. Fixed patterns are always destination-aligned. See *Pattern Consumption* in *Chapter 2, Programming Model* for a depiction of pattern consumption.

Sample code FIXPAT.DOC

```
This is sample code to show the three fixed patterns available on the
Mach 64. Note that the 4x2 and 8x1 pattern modes are only available in
8 bpp accelerator modes. See FIXPAT.C for more details.


    /* ---- 8x8 mono pattern ---- */
    wait_for_fifo(11);

    // host = 8x8 pattern, src = foreground clr
    regw(DP_SRC, MONO_SRC_PATTERN | FRGD_SRC_FRGD_CLR);
    regw(DP_MIX, FRGD_MIX_S | BKGD_MIX_S);

    // enable 8x8 mono pattern
    regw(PAT_CNTL, PAT_MONO_8x8_ENABLE);
    regw(PAT_REG0, 0x818181FF);
    regw(PAT_REG1, 0xFF818181);

    // frgd color used when pattern bit is "1"
    // bkgd color used when pattern bit is "0"

    regw(DP_FRGD_CLR, get_color_code(WHITE));
    regw(DP_BKGD_CLR, get_color_code(LIGHTBLUE));

    regw(DST_X, 0);
    regw(DST_Y, 0);
    regw(DST_HEIGHT, 16);
    regw(DST_WIDTH, 16);

    /* ---- 4x2 color pattern ---- */
    wait_for_fifo(8);

    // src = color pattern
    regw(DP_SRC, FRGD_SRC_PATTERN);

    // enable 4x2 color pattern
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
regw(PAT_CNTL, PAT_CLR_4x2_ENABLE);

// 0x09 - LIGHTBLUE, 0x0F - WHITE
regw(PAT_REG0, 0x0F09090F);
regw(PAT_REG1, 0x0F09090F);

regw(DST_X, 32);
regw(DST_Y, 0);
regw(DST_HEIGHT, 16);
regw(DST_WIDTH, 32);

/* ---- 8x1 color pattern ---- */
wait_for_fifo(7);

// enable 8x1 color pattern
regw(PAT_CNTL, PAT_CLR_8x1_ENABLE);

// 0x09 - LIGHTBLUE, 0x0F - WHITE
regw(PAT_REG0, 0x0909090F);
regw(PAT_REG1, 0x0F090909);

regw(DST_X, 80);
regw(DST_Y, 0);
regw(DST_HEIGHT, 16);
regw(DST_WIDTH, 32);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

LINE DRAW

Line draws are performed using an 18-bit Bresenham line draw engine.

To draw a line:

1. Set up the draw context with either a context load or many register writes.
2. Determine the direction octant that the line trajectory will be drawn and set the DST_X_DIR, DST_Y_DIR and DST_Y_MAJOR bits accordingly. Also set the LAST_PEL_ON bit as desired (this bit only determines whether the last pixel in the line is drawn; it has no effect on the actual DST_X, DST_Y trajectory).
3. From the start and endpoints of the line, calculate all the Bresenham parameters and write them out:

```
DST_BRES_ERR = 2 * min(|dx|,|dy|) - max(|dx|,|dy|)
DST_BRES_INC = 2 * min(|dx|,|dy|)
DST_BRES_DEC = 2 * [min(|dx|,|dy|) - max(|dx|,|dy|)]
```

4. Write out the desired number of pixels drawn to DST_BRES_LNTH.

Sample code

```
This is sample code to draw a line. The engine does not support lines in
24 bpp modes. See LINE.C and DRAW.C for more details.


/* ------------------------------------------------------------------------
  DRAW_LINE - draw a line from (x1, y1) to (x2, y2)

  The drawing of the last pixel in the line is determined by the current
  setting of the DST_CNTL register (LAST_PEL bit).
------------------------------------------------------------------------ */
void draw_line(int x1, int y1, int x2, int y2)
{
    int dx, dy;
    int small, large;
    int x_dir, y_dir, y_major;
    unsigned long err, inc, dec, temp;


    /* determine x & y deltas and x & y direction bits */
    if (x1 < x2)
    {
        dx = x2 - x1;
        x_dir = 1;
    }
    else
    {
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        dx = x1 - x2;
        x_dir = 0;
    }

    if (y1 < y2)
    {
        dy = y2 - y1;
        y_dir = 2;
    }
    else
    {
        dy = y1 - y2;
        y_dir = 0;
    }

    /* determine x & y min and max values; also determine y major bit */
    if (dx < dy)
    {
        small = dx;
        large = dy;
        y_major = 4;
    }
    else
    {
        small = dy;
        large = dx;
        y_major = 0;
    }

    /* calculate bressenham parameters and draw line */
    err = (unsigned long)((2 * small) - large);
    inc = (unsigned long)(2 * small);
    dec = 0x3ffff - ((unsigned long)(2 * (large - small)));

    wait_for_idle();        // wait for idle before reading GUI registers

    // save used registers
    temp = regr(DST_CNTL);

    // draw bressenham line
    regw(DST_X, (unsigned long)x1);
    regw(DST_Y, (unsigned long)y1);

    // allow setting of last pel bit and polygon outline bit for line drawing

    regw(DST_CNTL, (temp & 0x60) | (unsigned long)(y_major | y_dir | x_dir));
    regw(DST_BRES_ERR, err);
    regw(DST_BRES_INC, inc);
    regw(DST_BRES_DEC, dec);
    regw(DST_BRES_LNTH, (unsigned long)(large + 1));

    // restore
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    regw(DST_CNTL, temp);
}


/* Main C program */

void main(void)
{
    ...

    /* ---- Draw a line in each octant ---- */

    // The line draw direction is determined in DST_CNTL. These bits are
    // determined in the draw_line() routine from the given start and end
    // point coordinates. Note that the engine does not support lines in 24
    // bpp. However, it is possible to use the engine to draw each 24 bpp
    // pixel of a line and use a line drawing algorithm such as bressenham
    // to determine where the pixels are drawn. The side effect is a great
    // loss in performance especially if patterned lines are considered.

    // source = foreground color
    wait_for_fifo(1);
    regw(DP_SRC, FRGD_SRC_FRGD_CLR);

    // foreground color = WHITE
    set_fg_color(get_color_code(WHITE));

    // draw a line
    draw_line(x1, y1, x2, y2);

    ...
}
```

Line drawing is not supported in packed 24bpp modes.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SCISSORING AND MASKING

Drawing may be inhibited outside a rectangular region by setting the scissor registers. Scissors are inclusive on all edges. Drawing behavior is undefined for any objects drawn outside the device coordinate space, whether they are scissored or not. The device coordinate space is -4096 to +4095 in the X direction, and -16384 to +16383 in the Y direction.

Sample code SCISSOR.DOC

```
This is sample code to setup the engine scissors. See SCISSOR.C for more
details.


    // Setup rectangle size
    dstx = 0;
    dsty = 0;
    dstwidth = 640;
    dstheight = 480;

    // Set the scissors to the size of the screen mode (inclusive)
    wait_for_fifo(4);
    regw(SC_LEFT, 0);
    regw(SC_TOP, 0);
    regw(SC_RIGHT, dstwidth - 1);
    regw(SC_BOTTOM, dstheight - 1);

    // Draw a WHITE filled rectangle
    set_fg_color(get_color_code(WHITE));
    draw_rectangle(dstx, dsty, dstwidth, dstheight);

    // Set the scissors to a slightly smaller size than the drawn rectangle
    wait_for_fifo(4);
    regw(SC_LEFT, 50);
    regw(SC_TOP, 50);
    regw(SC_RIGHT, dstwidth - 50);
    regw(SC_BOTTOM, dstheight - 50);

    // Draw a LIGHTRED filled rectangle of the same size as the previous one.
    // The outer border of the rectangle will be clipped by the scissors.

    set_fg_color(get_color_code(LIGHTRED));
    draw_rectangle(dstx, dsty, dstwidth, dstheight);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Bits within a particular pixel may be selectively inhibited by setting the DP_WRITE_MASK register

Sample code WMASK.DOC

This is sample code to show the effects of the DP_WRITE_MASK register. See WMASK.C for more details.

```
// Set write mask to pass all bits to destination
wait_for_fifo(1);
regw(DP_WRITE_MASK, 0xffffffff);

// Draw a WHITE filled rectangle - resolution independent
set_fg_color(get_color_code(WHITE));
draw_rectangle(0, 0, modeinfo.xres / 2, modeinfo.yres / 2);

// setup mask for WRITE_MASK register
switch(modeinfo.bpp)
{
    case 4: mask = 0xfffffff0; break;
    case 8: mask = 0xffffff00; break;
    case 16: mask = 0xffff0000; break;
    case 24: mask = 0xff000000; break;
    case 32: mask = 0x00000000; break;
}

// Set write mask to mask off bits giving a result color of
// LIGHTMAGENTA without changing the DP_FRGD_CLR register value

wait_for_fifo(1);
regw(DP_WRITE_MASK, mask | get_color_code(LIGHTMAGENTA));

// Draw a filled rectangle with the new write mask setting
set_fg_color(get_color_code(WHITE));
draw_rectangle(modeinfo.xres / 2, 0,
               modeinfo.xres / 2, modeinfo.yres / 2);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SOURCE AND DESTINATION MIXING

A source and destination pixel may be mixed in two ways:

- A logical operation or an averaging function may be performed on the source and destination to produce a composite pixel. The process may be referred to as an ALU function, a mix function, or a ROP (raster operation).
- The color source pixel (before ALU processing) or the destination pixel can be compared to a color compare register. If the result is FALSE, the result of the ALU is written; otherwise, the destination pixel is written back to the destination (no pixel is drawn). In this manner, the source pixel can be selectively inhibited from writing to the destination.

ALU functions and compare functions may be used at the same time, but the ALU will only operate on pixels on which the compare function returns FALSE. All compare functions and ALU functions are listed in *Source and Destination Mixing Logic* in *Chapter 2, Programming Model.*

Sample code COMPARE.DOC

```
This is sample code to show the effect of mixing and the comparing of the
source with the destination. See COMPARE.C for more details.


    // Fill the screen with two filled rectangles - BLUE on top, YELLOW below

    // Set foreground mix to OVERPAINT
    set_fg_mix(S_MIX);

    set_fg_color(get_color_code(LIGHTBLUE));
    draw_rectangle(0, 0, modeinfo.xres / 2, modeinfo.yres / 4);

    set_fg_color(get_color_code(YELLOW));
    draw_rectangle(0, modeinfo.yres / 4,
                   modeinfo.xres / 2, modeinfo.yres / 4);


    // Draw a rectangle on top of the previously drawn one. The YELLOW
    // rectangle on the screen is used as the source comparison. Since the
    // compare color is YELLOW and the compare function is EQUAL, the
    // resulting rectangle draw will not affect the YELLOW rectangle. The
    // LIGHTBLUE rectangle, however, will be overwritten. The show mixing,
    // the resulting rectangle (written on top of the LIGHTBLUE rectangle)
    // will be LIGHTMAGENTA (LIGHTBLUE destination OR LIGHTRED source).

    // set mixes
    set_fg_mix(D_OR_S_MIX);
    set_bg_mix(D_MIX);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// compare color = YELLOW
wait_for_fifo(3);
regw(CLR_CMP_CLR, get_color_code(YELLOW));

// allow all bits to pass in comparison
regw(CLR_CMP_MASK, 0xffffffff);

// compare source = destination
regw(CLR_CMP_CNTL, COMPARE_EQUAL);

// draw LIGHTRED rectangle over both LIGHTBLUE & YELLOW regions
set_fg_color(get_color_code(LIGHTRED));
draw_rectangle(0, 0, modeinfo.xres / 2, modeinfo.yres / 2);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Chapter 4
Advanced Topics I

POLYGONS

The *mach64* uses an alternate-fill algorithm for polygon filling. Polygon fills are simply rectangle fills with the DST_POLYGON_ENA@DST_CNTL bit set. At the beginning of each destination scan line, an internal polygon fill flag is reset. Whenever this flag is in a reset state, drawing is inhibited. The polygon boundary source (this source is implicit, set it up using the blit source registers) is consumed, providing polygon boundary data. Whenever a polygon edge is detected, the internal polygon fill flag is toggled. Only rectangular destinations proceeding in a left-to-right and top-to-bottom direction are supported for polygon filling.

Note that any monochrome or color sources may be selected in the pixel data path except for blit sources (because the blit source registers are used to configure the polygon source trajectory) when polygon filling. Polygon boundary source is only meaningful when configured to 1bpp pixel depth (set this with DP_SRC_PIX_WIDTH@DP_PIX_WIDTH).

Polygon boundaries are created by drawing lines in 1bpp mode with the DST_POLYGON_ENA@DST_CNTL bit set. This bit causes a maximum of one pixel per scan line to be drawn (horizontal lines are not drawn at all), and lines exceeding the left scissor boundary are saturated to the left scissor.

To draw a polygon:

1. Set the destination pixel depth to 1bpp.
2. Set the scissors to the outline drawing region.
3. Clear the off-screen area where the polygon outlines are to be drawn.
4. Set the mix to XOR. (This takes care of the degenerate case where two polygon boundary lines culminate in a vertical peak).
5. Set the DST_POLYGON_ENA bit.
6. Draw all polygon outline lines from top to bottom with LAST_PEL_OFF.
7. Set the scissors to the final destination area.
8. Set all necessary destination context registers (pixel depth, mix, etc.).

© 1993 ATI Technologies Inc.
Proprietary and Confidential

9. Set up the blit source registers to point to the polygon outline area.
10. Set up the blit destination registers to point to the final destination area (on-screen memory).
11. Blit.

Sample code POLYGON.DOC

```
This is sample code to show how to setup and fill a polygon shape. Polygons
are not supported in 24 bpp modes.


/* Main C program */

int main(void)
{
    unsigned long offset, color_depth;
    int width, height;
    int box_x, box_y;
    int x1, y1, x2, y2, xtemp, ytemp;
    int i;
    POINT points[6];

    // check if Mach64 adapter is installed
    if (detect_mach64() != YES_MACH64)
    {
        printf("Mach64 based adapter was not found.\n");
        return (1);
    }

    // fill global query structure by calling Mach 64 ROM
    if (query_hardware() != NO_ERROR)
    {
        printf("Failed ROM call to query Mach64 hardware.\n");
        return (1);
    }

    // set an accelerator mode
    if (open_mode(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8) != NO_ERROR)
    {
        printf("Error in setting display mode.\n");
        return (1);
    }

    // Check for 24 bpp mode - Polygons are not supported in 24 bpp modes
    if (modeinfo.bpp == 24)
    {
        // disable accelerator mode and switch back to VGA text mode
        close_mode();

        printf("Polygons are not supported in 24 bpp modes.\n");
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    return (1);
}

// setup engine context and clear screen
init_engine();
clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

// Polygon shape is a 5 sided pentagon - resolution independent

points[0].x = modeinfo.xres / 2;
points[0].y = modeinfo.yres / 12;
points[1].x = (modeinfo.xres / 2) + (modeinfo.xres / 4);
points[1].y = modeinfo.yres / 2;
points[2].x = (modeinfo.xres / 2) + (modeinfo.xres / 8);
points[2].y = modeinfo.yres - (modeinfo.yres / 12);
points[3].x = (modeinfo.xres / 2) - (modeinfo.xres / 8);
points[3].y = modeinfo.yres - (modeinfo.yres / 12);
points[4].x = (modeinfo.xres / 2) - (modeinfo.xres / 4);
points[4].y = modeinfo.yres / 2;
points[5].x = points[0].x;
points[5].y = points[0].y;

// determine width, height, x & y offsets
width = points[1].x - points[4].x + 1;
height = points[2].y - points[0].y + 1;
box_x = points[4].x;
box_y = points[0].y;

// Draw outline of shape to be filled to show the inclusivity of the
// polygon fill. The bottom line is a horizontal line. Since polygon
// outlines are drawn upto but not including the last pixel, this line
// will remain. For the same reason, the top pixel of the shape will
// remain.

// Draw outline with regular lines in WHITE using the same top to bottom
// drawing direction.
set_fg_color(get_color_code(WHITE));
for (i = 0; i < 5; i++)
{
    x1 = points[i].x;
    y1 = points[i].y;
    x2 = points[i+1].x;
    y2 = points[i+1].y;

    // swap points if direction is not top to bottom
    if (y1 > y2)
    {
        ytemp = y1;
        y1 = y2;
        y2 = ytemp;

        xtemp = x1;
        x1 = x2;
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            x2 = xtemp;
        }

        draw_line(x1, y1, x2, y2);
    }

    // wait for a carriage return to continue
    getch();


    /* ---- Draw filled polygon ---- */

    // Calculate dword offset address of the start of off-screen memory

    offset = (unsigned long)(modeinfo.yres);
    offset = (unsigned long)(offset * modeinfo.pitch);
    if (modeinfo.bpp == 4)
    {
        offset = (unsigned long)(offset / 2);
    }
    else
    {
        offset = (unsigned long)(offset * (modeinfo.bpp / 8));
    }

    // convert byte offset to dword offset
    offset = offset / 8;

    // 1. Set destination to 1 bpp for memory clearing and outline drawing
    wait_for_fifo(16);
    regw(DP_PIX_WIDTH, HOST_1BPP | SRC_1BPP | DST_1BPP);


    // 2. Set scissors to outline drawing region
    regw(SC_LEFT, 0);
    regw(SC_TOP, 0);
    regw(SC_RIGHT, width-1);
    regw(SC_BOTTOM, height-1);


    // 3. Clear off-screen memory where polygon outlines are to be drawn

    // insure engine idleness for GUI register reading
    wait_for_idle();

    // set destination operations to off-screen memory
    regw(DST_OFF_PITCH, regr(DST_OFF_PITCH) | offset);

    // clear memory
    regw(DP_SRC, FRGD_SRC_FRGD_CLR);
    regw(DP_MIX, FRGD_MIX_ZERO | BKGD_MIX_ZERO);
    regw(DST_X, 0);
    regw(DST_Y, 0);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
regw(DST_HEIGHT, height);
regw(DST_WIDTH, width);


// 4. Set mix to XOR
regw(DP_MIX, FRGD_MIX_D_XOR_S | BKGD_MIX_ZERO);
regw(DP_FRGD_CLR, 1);


// 5. Set the DST_POLYGON_ENABLE bit, clear LAST_PEL bit
regw(DST_CNTL, DST_POLYGON_ENABLE |
               DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);


// 6. Draw all polygon outlines from top to bottom with LAST_PEL_OFF
for (i = 0; i < 5; i++)
{
    x1 = points[i].x - box_x;
    y1 = points[i].y - box_y;
    x2 = points[i+1].x - box_x;
    y2 = points[i+1].y - box_y;

    // swap points if direction is not top to bottom
    if (y1 > y2)
    {
        ytemp = y1;
        y1 = y2;
        y2 = ytemp;

        xtemp = x1;
        x1 = x2;
        x2 = xtemp;
    }

    draw_line(x1, y1, x2, y2);
}

// 7. Set scissors to the final destination area
wait_for_fifo(4);
regw(SC_LEFT, 0);
regw(SC_TOP, 0);
regw(SC_RIGHT, modeinfo.xres-1);
regw(SC_BOTTOM, modeinfo.yres-1);


// 8. Set all necessary destination context registers

// insure engine idleness for GUI register reading
wait_for_idle();

// set source operations to off-screen memory
regw(SRC_OFF_PITCH, regr(SRC_OFF_PITCH) | offset);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    // set destination operations to on-screen memory
    regw(DST_OFF_PITCH, regr(DST_OFF_PITCH) & 0xffc00000);

    // set destination color depth to current mode
    switch(modeinfo.bpp)
    {
        case 4: color_depth = DST_4BPP; break;
        case 8: color_depth = DST_8BPP; break;
        case 16:
            if (modeinfo.depth == 555)    // 555 color weighting
            {
                color_depth = DST_15BPP;
            }
            else                          // 565 color weighting
            {
                color_depth = DST_16BPP;
            }
            break;
        case 32: color_depth = DST_32BPP; break;
    }
    regw(DP_PIX_WIDTH, HOST_1BPP | SRC_1BPP | color_depth);

    // set desired mix and color values
    set_fg_mix(S_MIX);
    set_bg_mix(D_MIX);
    set_fg_color(get_color_code(LIGHTRED));


    // 9. Setup blit source registers to point to polygon outline area
    wait_for_fifo(8);
    regw(SRC_X, 0);
    regw(SRC_Y, 0);
    regw(SRC_HEIGHT1, height);
    regw(SRC_WIDTH1, width);

    // 10. Setup blit destination registers to point to final destination area
    regw(DST_X, box_x);
    regw(DST_Y, box_y);
    regw(DST_HEIGHT, height);

    // 11. Blit
    regw(DST_WIDTH, width);

    // wait for a carriage return
    getch();

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

DRAWING IN PACKED 24 BIT PER PIXEL MODE

There is no 24-bit packed draw engine mode, but there is a 24-bit packed display mode. Drawing in this mode is accomplished by setting the engine in 8 bit per pixel mode and manipulating the DST_24_ROT and DST_24_ROT_ENA bits. The following rules must be followed for drawing in this mode:

- Set source and destination pitches to three times the display pitch.
- All X coordinates and widths must be specified at three times the normal value. Remember that left-to-right operations begin on an R value, and right-to-left operations begin on a B value. That means that for left-to-right operations, the initial DST_X is expressed as (X * 3) and for right-to-left DST_X is (X * 3 + 2).
- Before any draw operation is initiated, the DST_24_ROT_ENA@DST_CNTL must be enabled, and DST_24_ROT@DST_CNTL must be set to ((DST_X / 4) mod 6), where DST_X is the starting DST_X value as described above.

DST_X (8bpp)																												
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	....
X (24bpp)																												
0			1			2			3			4			5			6			7			8			....	
DWORD																												
0				1				2				3				4				5				6			....	
DST_24_ROT Value																												
0		0	0		1	1		2	2		2	3		3	3		4	4		5	5		5	0		0	....	
COLOR COMPONENTS																												
R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	....

In the above table:

- **X** is the desired X coordinate in packed 24bpp mode.
- **DST_X** is the value that you actually write to the draw engine (remember to start on an R component on left-to-right operations, and on a B component for right-to-left operations).
- The **DWORD** and **color components** rows show how memory is actually laid out in relation to pixel data.
- The **DST_24_ROT** row shows the value to place in the DST_24_ROT@DST_CNTL field before initiating a draw operation. Use the leftmost DST_24_ROT number in the column for left-to-right operations, and the rightmost number for right-to-left operations.
- The **DST_24_ROT** value is simply the (DWORD-value-of-the-starting-byte mod 6).
- The **rotation enable bit** only affects DP_FRGD_CLR, DP_BKGD_CLR, DP_WRITE_MASK, and fixed 8x8 mono patterns. Colors and masks are rotated appropriately, keying on the DST_24_ROT value.
- The line draw engine does not function in 24bpp packed mode.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SCROLLING AND PANNING

Scrolling and panning of the display area to the limits of the draw area can be simply done by changing the value of CRTC_OFFSET@CRTC_OFF_PITCH.

Note that offset has a granularity of 64 bits, which means that horizontal panning will be more "jerky" at lower pixel depths than at higher pixel depths.

Sample code PAN.DOC

```
This is sample code to show panning of a 1024x1024 image in a 640x480
viewable screen area. See PAN.C for more details.

// Keyboard scan and ascii codes for demo

#define ESC         0x1b  // ascii code
#define LEFT_ARROW  0x4b  // scan code
#define RIGHT_ARROW 0x4d  // scan code
#define UP_ARROW    0x48  // scan code
#define DOWN_ARROW  0x50  // scan code
#define HOME        0x47  // scan code
#define END         0x4f  // scan code

/* ------------------------------------------------------------------------
  GET_KEY - get keyboard scan code using system ROM call

  Upper 8 bits = scan code
  Lower 8 bits = ascii code
------------------------------------------------------------------------ */
int get_key(void)
{
    union REGS regs;

    regs.x.ax = 0x1000;
    int86(0x16, &regs, &regs);

    return (regs.x.ax);
}


/* Main C program */

int main(void)
{
    unsigned long offset;
    int color, ch;
    int xindex, yindex, xmax, ymax;
    int max_x, max_y, size_x, size_y;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// check if Mach64 adapter is installed
if (detect_mach64() != YES_MACH64)
{
    printf("Mach64 based adapter was not found.\n");
    return (1);
}

// fill global query structure by calling Mach 64 ROM
if (query_hardware() != NO_ERROR)
{
    printf("Failed ROM call to query Mach64 hardware.\n");
    return (1);
}

// set an accelerator mode of 640x480 with a pitch of 1024
if (open_mode(MODE_640x480, PITCH_1024, COLOR_DEPTH_8) != NO_ERROR)
{
    printf("Error in setting display mode.\n");
    return (1);
}

/*
   To scroll and pan the display, the viewable display area must be
   smaller than the CRTC display area. In this case, the viewable area
   is 640x480 while the CRTC display area is 1024x1024.
*/

// 'desktop' size = 1024 x 1024
max_x = 1024;
max_y = 1024;

// 'viewable screen size = 640 x 480
size_x = 640;
size_y = 480;

// setup engine context and clear screen
init_engine();

// adjust engine scissors to desktop area (1024x1024)
wait_for_fifo(4);
regw(SC_LEFT, 0);
regw(SC_TOP, 0);
regw(SC_RIGHT, max_x-1);
regw(SC_BOTTOM, max_y-1);

// clear area
clear_screen(0, 0, max_x, max_y);

// Fill area with some full size rectangles (CRTC width)
for (color = DARKBLUE-1; color < WHITE; color++)
{
    set_fg_color(get_color_code(color + 1));
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        draw_rectangle(color * 30,
                       color * 30,
                       max_x - (color * 60) - 1,
                       max_y - (color * 60) - 1);
    }

    // Scroll and pan around image:
    //
    //   Viewable display is 640x480
    //   Image size is 1024x1024
    //
    //   ESC         - exit
    //   LEFT_ARROW  - move left
    //   RIGHT_ARROW - move right
    //   UP_ARROW    - move up
    //   DOWN_ARROW  - move down
    //   HOME        - move to top-left corner
    //   END         - move to bottom-right corner
    //

    xindex = 0;
    yindex = 0;

    xmax = (max_x - size_x) / 2;      // 2 pixels per step (in 8 bpp)
    ymax = (max_y - size_y) / 2;      // 2 lines per step
    if (modeinfo.bpp == 4)
    {
        xmax = xmax / 2;
    }
    else
    {
        xmax = xmax * (modeinfo.bpp / 8);
    }

    offset = 0;
    ch = 0;
    while ((ch & 0xff) != ESC)
    {
        // wait for key input
        while (kbhit() == 0) ;
        ch = get_key();

        // position CRTC offset according to key input
        if ((ch & 0xff) != ESC)
        {
            switch (ch >> 8)
            {
                case LEFT_ARROW:   // move left 2 pixels
                    if (xindex > 0)
                    {
                        offset = offset - 1;
                        xindex = xindex - 4;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    }
    break;

case RIGHT_ARROW:  // move right 2 pixels
    if (xindex < xmax)
    {
        offset = offset + 1;
        xindex = xindex + 4;
    }
    break;

case UP_ARROW:     // move up 2 lines
    if (yindex > 0)
    {
        if (modeinfo.bpp == 4)
        {
            offset = offset - (256 / 2);
        }
        else
        {
            offset = offset - (256 * (modeinfo.bpp / 8));
        }
        yindex = yindex - 1;
    }
    break;

case DOWN_ARROW:   // move down 2 lines
    if (yindex < ymax)
    {
        if (modeinfo.bpp == 4)
        {
            offset = offset + (256 / 2);
        }
        else
        {
            offset = offset + (256 * (modeinfo.bpp / 8));
        }
        yindex = yindex + 1;
    }
    break;

case HOME:         // move to top-left corner (0, 0)
    offset = 0;
    xindex = 0;
    yindex = 0;
    break;

case END:          // move to bottom-right corner

  // calculate dword address of coordinate (1024-640,1024-480)

    offset = (unsigned long)(max_y - size_y);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                    offset = (unsigned long)(offset * max_x);  // pitch
                    if (modeinfo.bpp == 4)
                    {
                        offset = (unsigned long)(offset / 2);
                     offset = (unsigned long)(offset + ((max_x - size_x) / 2));
                    }
                    else
                    {
                        offset = (unsigned long)(offset * (modeinfo.bpp / 8));
                       offset = (unsigned long)(offset + ((modeinfo.bpp / 8) *
(max_x - size_x)));
                    }
                    offset = offset / 8;
                    xindex = xmax;
                    yindex = ymax;
                    break;
            }

            // vary CRTC offset while maintaining mode pitch
          iow(ioCRTC_OFF_PITCH, (ior(ioCRTC_OFF_PITCH) & 0xffc00000)| offset);
        }
    }

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}
```

INTERRUPTS

The *mach64* is able to generate hardware interrupts under a variety of conditions:

- Interrupt on command FIFO overflow (BUS_CNTL)
- Interrupt on host data error (BUS_CNTL)
- Interrupt on CRTC vertical blank (CRTC_INT_CNTL)
- Interrupt on CRTC vertical line count == CRTC_VLINE (CRTC_INT_CNTL)

It is not recommended that interrupts be used in retail software applications, because ISA-based systems tend to be fully loaded with hardware-interruptable devices, and ISA interrupts are not shareable.

See *Single Buffering (Synchronized)* on page 3-22 for an example of how interrupts can be used.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

TRANSPARENT BLITS

A **transparent blit** is simply a blit where a designated color (background color) from the source is inhibited from being drawn to the destination. This kind of blit is useful for copying odd-shaped objects onto a bitmapped background (games, for example). A simple blit with source compare enabled will do a transparent blit.

Sample code TBLIT.DOC

```
This is sample code to show a transparent blit. See TBLIT.C and DRAW.C for
more details.


    // Draw a background with different colored rectangles - resolution
    // independent
    for (i = BLACK; i <= WHITE; i++)
    {
        set_fg_color(get_color_code(i));
        draw_rectangle(i * (modeinfo.xres / 40),
                       i * (modeinfo.yres / 40),
                       modeinfo.xres - (i * 2 * (modeinfo.xres / 40)),
                       modeinfo.yres - (i * 2 * (modeinfo.yres / 40)));
    }


    // Draw the source to be blited
    set_fg_color(get_color_code(WHITE));
    draw_rectangle(0, 0, modeinfo.xres / 8, modeinfo.yres / 6);

    set_fg_color(get_color_code(LIGHTRED));
    draw_rectangle(modeinfo.xres / 64,
                   modeinfo.yres / 48,
                   (modeinfo.xres / 8) - (2 * (modeinfo.xres / 64)),
                   (modeinfo.yres / 6) - (2 * (modeinfo.yres / 48)));

    set_fg_color(get_color_code(LIGHTBLUE));
    draw_rectangle(modeinfo.xres / 32,
                   modeinfo.yres / 24,
                   (modeinfo.xres / 8) - (2 * (modeinfo.xres / 32)),
                   (modeinfo.yres / 6) - (2 * (modeinfo.yres / 24)));

    // Set color compare registers to source compare:
    //
    // In this example, the LIGHTBLUE center of the source rectangle
    // is selected as being the source color to ignore when bliting.
    // The resultant blit will contain all the source data except pixels
    // having the same color as the compare color (LIGHTBLUE). Note that
    // the compare source is SOURCE. Also, the compare color and the color
    // pixel being compared are both ANDed with the color compare mask.
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    wait_for_fifo(3);
    regw(CLR_CMP_CNTL, COMPARE_SOURCE | COMPARE_EQUAL);
    regw(CLR_CMP_CLR, get_color_code(LIGHTBLUE));
    regw(CLR_CMP_MASK, 0xffffffff);

    // draw several transparent blits at different locations

    // set src type for blit
    wait_for_fifo(1);
    regw(DP_SRC, FRGD_SRC_BLIT);

    for (i = 0; i < 6; i++)
    {
        blit(0, 0, (modeinfo.xres / 8) + (i * (modeinfo.xres / 8)),
             i * (modeinfo.xres / 8),
             modeinfo.xres / 8,
             modeinfo.xres / 8);
    }


    // By changing the compare function from COMPARE_EQUAL to
    // COMPARE_NOT_EQUAL, the resultant blit will contain ONLY the source
    // data having the same color as the compare color (LIGHTBLUE).

    wait_for_fifo(1);
    regw(CLR_CMP_CNTL, COMPARE_SOURCE | COMPARE_NOT_EQUAL);

    // draw several transparent blits at different locations
    for (i = 0; i < 6; i++)
    {
        blit(0, 0, (2 * (modeinfo.xres / 8)) + (i * (modeinfo.xres / 8)),
             i * (modeinfo.xres / 8),
             modeinfo.xres / 8,
             modeinfo.xres / 8);
    }

    // wait for a carriage return
    getch();

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

CRT SYNCHRONIZATION

For smooth animation, it is necessary to inhibit drawing to areas of the screen that are currently being scanned by the CRT controller. Failure to take necessary precautions will cause flickering or tearing effects on the animated object. Outlined below are several possible strategies that can be used for smooth animation.

Double Buffering (Memory)

Two areas of screen memory are allocated, each big enough for an entire display screen. While one memory area is being displayed, the other is updated, thus avoiding any collision between the CRTC and the draw engine. The system timer can be used to generate interrupts at constant time intervals.

In the interrupt service routine for the system timer:

1. Wait-for-idle to ensure that the draw engine is not in the middle of drawing.
2. Set CRT_OFFSET to toggle to the memory area to display. Optionally, the application may wait for a vertical blank or a vertical line range before writing to CRT_OFFSET to prevent tearing between the two images. However, this would significantly degrade system performance, and the torn display disappears after 1/43rd to 1/76th of a second (depending on refresh rate) so it is unlikely to have any visible impact on animation smoothness.
3. Set DST_OFFSET to enable writing to the non-displayed area.
4. Signal the application program that the display offset has changed.

In the mainline application:

1. Determine which memory area is being displayed. Remember that the displayed and non-displayed areas are not identical (they are one frame apart) and the application must update the non-displayed area by two frames.
2. Disable interrupts for critical draw operations.
3. Update the non-displayed area with critical draw operations. A critical draw operation may be something like an undraw of an object and a redraw at a new location. The application does not want the CRT_OFFSET to change until the object is completely redrawn at the new location.
4. Enable interrupts.

The buffer switching may also be done in the main line application, and using the system timer to switch buffers is optional. The advantage to using the system timer is a constant frame rate.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Sample code SAMPLE.DOC

This is sample code to show image drawing using double buffering. This example uses 8 targa files located in the IMAGE directory for its source images. See below for description.

```
/*=========================================================================
  DBUF.C

  Example code to show double buffered drawing using the system timer.
  Several image "frames" are loaded into off-screen memory so that they can
  blited to the screen in sequence. Image tearing is avoided by using a
  double page buffering. The page being updated is never the page being
  displayed. In this example, the page buffer is controlled by waiting for a
  system timer tick which yields an approximate frame rate of 18. A page swap
  is done every timer tick. Double buffering does not require waiting for a
  specific vertical line or blanking before drawing. If the frame is more
  than approximately half the vertical sync frequency, image tearing will
  occur. In this case, it will be necessary to wait for a specific vertical
  line before drawing.

  Copyright (c) 1994 ATI Technologies Inc. All rights reserved
 =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include "..\util\atim64.h"
#include "..\util\sample.h"
#include "..\util\vtga.h"
#include "isr8.h"

#define BACKGROUND_COLOR    0x11


/* Main C program */

int main(void)
{
    char filename[20];
    TARGA_HEADER header;
    int width, height;
    int srcx, srcy;
    int savex1, savey1;
    int savex2, savey2;
    int savex, savey;
    int i, j;
    int y_draw;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    int frame;
    int frames;
    int framesperwidth;
    int topline;
    int y_update;
    int update_flag;
    int old_flag;
    int step;
    unsigned long offset;
    POINT points[8];

    // check if Mach64 adapter is installed
    if (detect_mach64() != YES_MACH64)
    {
        printf("Mach64 based adapter was not found.\n");
        return (1);
    }

    // fill global query structure by calling Mach 64 ROM
    if (query_hardware() != NO_ERROR)
    {
        printf("Failed ROM call to query Mach64 hardware.\n");
        return (1);
    }

    // check if Mach 64 VGA controller is enabled
    if (querydata.vga_type != VGA_ENABLE)
    {
        printf("This sample code example requires an enabled Mach 64 VGA
controller.\n");
        return (1);
    }

    // set an accelerator mode
    if (open_mode(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8) != NO_ERROR)
    {
        printf("Error in setting display mode.\n");
        return (1);
    }

    // setup engine context and clear screen
    init_engine();
    clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

    // get targa header information
    if (get_targa_header("..\\image\\frame1.tga", &header) != SUCCESS)
    {
        close_mode();
        printf("Error reading targa file header information.\n");
        return (1);
    }
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// setup image size, source area, and save area
width = header.width;
height = header.height;
srcx = 0;
srcy = (modeinfo.yres * 2) + height;
savex1 = 0;
savey1 = modeinfo.yres * 2;
savex2 = width;
savey2 = savey1;
y_draw = modeinfo.yres - height;
step = 2;

// inform ISR where second page begins (QWORD address)

// calculate byte address
offset = (unsigned long)(modeinfo.yres);
offset = (unsigned long)(offset * modeinfo.pitch);
if (modeinfo.bpp == 4)
{
    offset = (unsigned long)(offset / 2);
}
else
{
    offset = (unsigned long)(offset * (modeinfo.bpp / 8));
}

// convert byte address to qword address
offset = offset / 8;

// add current pitch from CRTC_OFF_PITCH to variable
offset = offset | (ior(ioCRTC_OFF_PITCH) & 0xffc00000);
setcrtcoffset(offset);

// determine how large to expand the scissors
frames = 8;
framesperwidth = modeinfo.xres / width;
topline = frames / framesperwidth;
if ((topline * framesperwidth) != frames)
{
    topline++;
}
topline = ((topline + 1) * height) + (modeinfo.yres * 2);

// expand scissors
wait_for_fifo(4);
regw(SC_LEFT, 0);
regw(SC_TOP, 0);
regw(SC_RIGHT, modeinfo.xres - 1);
regw(SC_BOTTOM, topline - 1);

// setup background the same as the image background
set_fg_color(BACKGROUND_COLOR);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
draw_rectangle(0, 0, modeinfo.xres, modeinfo.yres);

// copy background (first buffer) to second buffer
blit(0, 0, 0, modeinfo.yres, modeinfo.xres, modeinfo.yres);

// load source images
frame = 0;
i = 0;
j = 0;
while (frame < frames)
{
    // record each frame coordinate
    points[frame].x = srcx + (width * i);
    points[frame].y = srcy + (height * j);

    // load next frame image into video memory
    sprintf(filename, "..\\image\\frame%d.tga", frame+1);
    if (load_targa(filename, points[frame].x, points[frame].y) != SUCCESS)
    {
        close_mode();
        printf("Error loading targa file to memory.\n");
        return (1);
    }

    // adjust location of frame load coordinate as necessary
    frame++;
    i++;
    if (i > ((modeinfo.xres / header.width) - 1))
    {
        i = 0;
        j++;
    }
}

// set palette from targa color table (8 bpp)
if (header.pixel_depth == 8)
{
    if (set_targa_palette("..\\image\\frame1.tga") != SUCCESS)
    {
        close_mode();
        printf("Error reading targa file color table information.\n");
        return (1);
    }
}

// wait for a key to start
getch();

// setup for blits
wait_for_fifo(3);
regw(DP_SRC, FRGD_SRC_BLIT);
regw(SRC_CNTL, 0);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

// copy starting postions to buffer save areas
blit(0, y_draw, savex1, savey1, width, height);
blit(0, modeinfo.yres + y_draw, savex2, savey2, width, height);

// setup loop variables
i = 0;
frame = 0;
update_flag = getswapflag();
old_flag = update_flag;

// enable timer ISR for page swaps each timer tick
inittimerisr();

// main draw loop
while (i < (modeinfo.xres-width))
{
    // syncronize frame updates with ISR
    while (update_flag == old_flag)
    {
        update_flag = getswapflag();
    }
    old_flag = update_flag;

    // don't allow ISR to swap page while updating frame
    interrupts_off();

    // set update and save variables according to active frame
    if (update_flag == 1)
    {
        // display frame 1, update frame 2
        y_update = modeinfo.yres;
        savex = savex2;
        savey = savey2;
    }
    else
    {
        // display frame 2, update frame 1
        y_update = 0;
        savex = savex1;
        savey = savey1;
    }

    // restore current frame from last frame (each page lags by 2 frames)
    if (i > step)
    {
      blit(savex, savey, i - (2 * step), y_update + y_draw, width, height);
    }

    // save current frame
    blit(i, y_update + y_draw, savex, savey, width, height);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        // update current frame
     blit(points[frame].x, points[frame].y, i, y_update + y_draw, width, height);

        // allow ISR to swap page
        interrupts_on();

        // increment image position
        i = i + step;

        // determine next frame image
        frame++;
        if (frame >= frames)
        {
            frame = 0;
        }
    }

    // disable timer ISR
    canceltimerisr();

    // wait for a carriage return
    getch();

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}
```

Double Buffering (Palette)

The palette-driven double buffer is just a specialized case of the double buffer scheme described above. In 8bpp mode, two memory areas can be allocated, overlaid on top of each other, each 4bpp deep. The palette must be defined such that the lower four bits and the upper four bits specify the same 16 colors. The same algorithm is used as above, except that DAC_MASK is used to switch the displayed area, and DP_WRITE_MASK is used to write to the non-displayed area.

Single Buffering (Synchronized)

Simple animations (small update areas) may be accomplished with a single buffer with no flickering or tearing by refraining from drawing until the CRTC vertical line count is within a certain range. The vertical line count can be polled by reading CRTC_CRNT_VLINE@CRTC_VLINE_CRNT_VLINE or it can be interrupt-driven by setting CRTC_VLINE_INT_EN@CRTC_INT_CNTL and CRTC_VLINE@CRTC_VLINE_CRNT_VLINE.

Interrupts from the *mach64* chip are not recommended because ISA systems cannot share interrupts, and commonly run out of IRQ levels.

Sample code SBUFP.DOC

```
This is sample code to show image drawing using vertical line polling. This
example uses 8 targa files located in the IMAGE directory for its source
images. See below for description.


/*=========================================================================
  SBUFP.C

  Example code to show single buffered drawing by polling. Several image
  "frames" are loaded into off-screen memory so that they can blited to
  the screen in sequence. To prevent image tearing, a blit is not performed
  until the display reaches a specific vertical line (usually after the lower
  part of the blit). In this example, this is done by polling VLINE register.
  The frame rate is determined by the vertical frequency of the display
  unless the draw code can not keep up with the frame rate.

  Copyright (c) 1994 ATI Technologies Inc. All rights reserved
 =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include "..\util\atim64.h"
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
#include "..\util\sample.h"
#include "..\util\vint.h"
#include "..\util\vtga.h"

#define BACKGROUND_COLOR    0x11


/* Main C program */

int main(void)
{
    char filename[20];
    TARGA_HEADER header;
    int width, height;
    int srcx, srcy;
    int savex, savey;
    int i, j;
    int y_draw;
    int frame;
    int frames;
    int framesperwidth;
    int topline;
    int step;
    POINT points[8];

    // check if Mach64 adapter is installed
    if (detect_mach64() != YES_MACH64)
    {
        printf("Mach64 based adapter was not found.\n");
        return (1);
    }

    // fill global query structure by calling Mach 64 ROM
    if (query_hardware() != NO_ERROR)
    {
        printf("Failed ROM call to query Mach64 hardware.\n");
        return (1);
    }

    // check if Mach 64 VGA controller is enabled
    if (querydata.vga_type != VGA_ENABLE)
    {
        printf("This sample code example requires an enabled Mach 64 VGA
controller.\n");
        return (1);
    }

    // set an accelerator mode
    if (open_mode(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8) != NO_ERROR)
    {
        printf("Error in setting display mode.\n");
        return (1);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
}

// setup engine context and clear screen
init_engine();
clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

// get targa header information
if (get_targa_header("..\\image\\frame1.tga", &header) != SUCCESS)
{
    close_mode();
    printf("Error reading targa file header information.\n");
    return (1);
}

// setup image size, source area, save area, and position increment
width = header.width;
height = header.height;
srcx = 0;
srcy = modeinfo.yres + height;
savex = 0;
savey = modeinfo.yres;
y_draw = modeinfo.yres - height;
step = 2;

// determine how large to expand the scissors
frames = 8;
framesperwidth = modeinfo.xres / width;
topline = frames / framesperwidth;
if ((topline * framesperwidth) != frames)
{
    topline++;
}
topline = ((topline + 1) * height) + modeinfo.yres;

// expand scissors to include source and save areas
wait_for_fifo(4);
regw(SC_LEFT, 0);
regw(SC_TOP, 0);
regw(SC_RIGHT, modeinfo.xres - 1);
regw(SC_BOTTOM, topline - 1);

// draw background the same as the image background
set_fg_color(BACKGROUND_COLOR);
draw_rectangle(0, 0, modeinfo.xres, modeinfo.yres);

// load source images into off-screen memory for frame blitting and
// record their position
frame = 0;
i = 0;
j = 0;
while (frame < frames)
{
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        // record each frame coordinate
        points[frame].x = srcx + (width * i);
        points[frame].y = srcy + (height * j);

        // load next frame image into video memory
        sprintf(filename, "..\\image\\frame%d.tga", frame+1);
        if (load_targa(filename, points[frame].x, points[frame].y) != SUCCESS)
        {
            close_mode();
            printf("Error loading targa file to memory.\n");
            return (1);
        }

        // adjust location of frame load coordinate as necessary
        frame++;
        i++;
        if (i > ((modeinfo.xres / header.width) - 1))
        {
            i = 0;
            j++;
        }
    }

    // set palette from targa color table (8 bpp)
    if (header.pixel_depth == 8)
    {
        if (set_targa_palette("..\\image\\frame1.tga") != SUCCESS)
        {
            close_mode();
            printf("Error reading targa file color table information.\n");
            return (1);
        }
    }

    // wait for a key to start
    getch();

    // setup engine for blits
    wait_for_fifo(3);
    regw(DP_SRC, FRGD_SRC_BLIT);
    regw(SRC_CNTL, 0);
    regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

    // set vline to wait (to prevent image tearing) and enable
    if (y_draw < (modeinfo.yres - height))
    {
        set_vline(y_draw + height);
    }
    else
    {
        set_vline(modeinfo.yres);
    }
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    enable_vlineint();

    // main draw loop
    i = 0;
    frame = 0;
    while (i < (modeinfo.xres - width))
    {
        // display old frame, update new frame
        wait_for_vline();  // wait for vertical line

        // restore previous frame (if first frame has occurred already)
        if (i > 0)
        {
            blit(savex, savey, i-step, y_draw, width, height);
        }

        // save current frame
        blit(i, y_draw, savex, savey, width, height);

        // update current frame
        blit(points[frame].x, points[frame].y, i, y_draw, width, height);

        // cycle through frames
        i = i + step;
        frame++;
        if (frame >= frames)
        {
            frame = 0;
        }
    }

    // disable vline interrupts
    disable_vlineint();

    // wait for a carriage return
    getch();

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

SBUFI.DOC

This is sample code to show image drawing using vertical line interrupts. This example uses 8 targa files located in the IMAGE directory for its source images. See below for description.

```
/*=========================================================================
  SBUFI.C

  Example code to show single buffered drawing using interrupts. Several
  image "frames" are loaded into off-screen memory so that they can blited to
  the screen in sequence. To prevent image tearing, a blit is not performed
  until the display reaches a specific vertical line (usually after the lower
  part of the blit). In this example, this is done by setting the VLINE
  and INT_CNTL registers to trigger an interrupt service routine (ISR). For
  the ISR to function, the Mach64's IRQ2 jumper must be installed. The ISR
  may fail to function correctly if the IRQ channel is used by other hardware
  or is prevented to function.

  Note that the blit code is in the ISR instead of the main loop (as is the
  case for the polled example - SBUFP).

  The frame rate is determined by the vertical frequency of the display
  unless the draw code can not keep up with the frame rate.

  Copyright (c) 1994 ATI Technologies Inc. All rights reserved
 =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include "..\util\atim64.h"
#include "..\util\sample.h"
#include "..\util\vint.h"
#include "..\util\vtga.h"
#include "visr.h"

#define IRQ                 2
#define IRQ_TIMEOUT         18
#define BACKGROUND_COLOR    0x11


/* Main C program */

int main(void)
{
    char filename[20];
    TARGA_HEADER header;
    int width, height;
```

```
    int srcx, srcy;
    int savex, savey;
    int i, j;
    int y_draw;
    int update;
    int frame;
    int frames;
    int framesperwidth;
    int topline;
    int step;
    int oldcount;
    unsigned int starttick, endtick;
    POINT points[8];

    // check if Mach64 adapter is installed
    if (detect_mach64() != YES_MACH64)
    {
        printf("Mach64 based adapter was not found.\n");
        return (1);
    }

    // fill global query structure by calling Mach 64 ROM
    if (query_hardware() != NO_ERROR)
    {
        printf("Failed ROM call to query Mach64 hardware.\n");
        return (1);
    }

    // check if Mach 64 VGA controller is enabled
    if (querydata.vga_type != VGA_ENABLE)
    {
        printf("This sample code example requires an enabled Mach 64 VGA
controller.\n");
        return (1);
    }

    // set an accelerator mode
    if (open_mode(MODE_640x480, PITCH_XRES, COLOR_DEPTH_8) != NO_ERROR)
    {
        printf("Error in setting display mode.\n");
        return (1);
    }

    // setup engine context and clear screen
    init_engine();
    clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

    // get targa header information
    if (get_targa_header("..\\image\\frame1.tga", &header) != SUCCESS)
    {
        close_mode();
        printf("Error reading targa file header information.\n");
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    return (1);
}

// setup image size, source area, save area, and position increment
width = header.width & 0xfffe;
height = header.height & 0xfffe;
srcx = 0;
srcy = modeinfo.yres + height;
savex = 0;
savey = modeinfo.yres;
y_draw = modeinfo.yres - height;
step = 2;

// setup blit dimensions for ISR
setblitinfo(width, height, srcx, srcy, savex, savey, y_draw, step);
setxstart(0);
setxpos(0);
setimage(0, srcx, srcy);

// determine how large to expand the scissors
frames = 8;
framesperwidth = modeinfo.xres / width;
topline = frames / framesperwidth;
if ((topline * framesperwidth) != frames)
{
    topline++;
}
topline = ((topline + 1) * height) + modeinfo.yres;

// expand scissors to include source and save areas
wait_for_fifo(4);
regw(SC_LEFT, 0);
regw(SC_TOP, 0);
regw(SC_RIGHT, modeinfo.xres - 1);
regw(SC_BOTTOM, topline - 1);

// set background color the same as the image background
set_fg_color(BACKGROUND_COLOR);
draw_rectangle(0, 0, modeinfo.xres, modeinfo.yres);

// load source images into off-screen memory for frame blitting and
// record their position
frame = 0;
i = 0;
j = 0;
while (frame < frames)
{
    // record each frame coordinate
    points[frame].x = srcx + (width * i);
    points[frame].y = srcy + (height * j);

    // load next frame image into video memory
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        sprintf(filename, "..\\image\\frame%d.tga", frame+1);
        if (load_targa(filename, points[frame].x, points[frame].y) != SUCCESS)
        {
            close_mode();
            printf("Error loading targa file to memory.\n");
            return (1);
        }

        // adjust location of frame load coordinate as necessary
        frame++;
        i++;
        if (i > ((modeinfo.xres / header.width) - 1))
        {
            i = 0;
            j++;
        }
    }

    // set palette from targa color table (8 bpp)
    if (header.pixel_depth == 8)
    {
        if (set_targa_palette("..\\image\\frame1.tga") != SUCCESS)
        {
            close_mode();
            printf("Error reading targa file color table information.\n");
            return (1);
        }
    }

    // wait for a key to start
    getch();

    // setup engine for blits
    wait_for_fifo(3);
    regw(DP_SRC, FRGD_SRC_BLIT);
    regw(SRC_CNTL, 0);
    regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

    // set vline to wait for (to prevent image tearing) and enable
    if (y_draw < (modeinfo.yres - height))
    {
        set_vline(y_draw + height);
    }
    else
    {
        set_vline(modeinfo.yres);
    }
    enable_vlineint();

    // enable ISR and start draw loop
    initvlineisr(IRQ);
    update = getcount();
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    oldcount = update;
    i = 0;
    frame = 0;
    while (i < (modeinfo.xres - width))
    {
        // syncronize with ISR (vline trigger) - time out if no response
        starttick = *((unsigned int far *)(DOS_TICK_ADDRESS));
        endtick = starttick;
        while (update == oldcount)
        {
            update = getcount();
            endtick = *((unsigned int far *)(DOS_TICK_ADDRESS));
            if (abs(endtick - starttick) > IRQ_TIMEOUT)
            {
                disable_vlineint();
                cancelvlineisr();
                close_mode();
                printf("IRQ channel %d is not responding.\n", IRQ);
                return (1);
            }
        }
        oldcount = update;

        // cycle through frames
        setimage(frame, points[frame].x, points[frame].y);
        frame++;
        if (frame >= frames)
        {
            frame = 0;
        }

        // increment position
        i = i + step;
        setxpos(i);
    }

    // disable vline interrupts
    disable_vlineint();

    cancelvlineisr();

    // wait for a carriage return
    getch();

    // disable accelerator mode and switch back to VGA text mode
    close_mode();

    return (0);
}

...
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
OldIntVector    dw  ?
                dw  ?
IRQnum          db  2
IRQindex        db  0ah
IRQmask1        db  ?
IRQmask2        db  ?
Count           db  0
Iwidth          dw  0
Height          dw  0
SrcX            dw  0
SrcY            dw  0
SaveX           dw  0
SaveY           dw  0
Ydraw           dw  0
Step            dw  0
Xstart          dw  0
Xpos            dw  0
Image           dw  0

; ---------------------------------------------------------------------------
; DO_BLIT
;
; INPUT:  WORD x1,
;         WORD y1,
;         WORD x2,
;         WORD y2,
;         WORD width,
;         WORD height
;
; OUTPUT: none
;
; Assumes VGA aperture is enabled
; ---------------------------------------------------------------------------
            public  do_blit

do_blit     proc    far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; Save used registers
            push    bx
            push    es

            ; Wait for fifo in FIFO_STAT (using VGA aperture)
            mov     ax, 0b000h
            mov     es, ax

waitfifo:
            mov     ax, es:[0ff10h]
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
or      ax, ax
jnz     waitfifo

; Load up blit registers
xor     bx, bx

; SRC_X = x1
mov     ax, WORD PTR [bp+PARM]
mov     es:[0fd84h], ax
mov     es:[0fd86h], bx

; SRC_Y = y1
mov     ax, WORD PTR [bp+PARM+2]
mov     es:[0fd88h], ax
mov     es:[0fd8ah], bx

; SRC_HEIGHT1 = height
mov     ax, WORD PTR [bp+PARM+10]
mov     es:[0fd94h], ax
mov     es:[0fd96h], bx

; SRC_WIDTH1 = width
mov     ax, WORD PTR [bp+PARM+8]
mov     es:[0fd90h], ax
mov     es:[0fd92h], bx

; DST_X = x2
mov     ax, WORD PTR [bp+PARM+4]
mov     es:[0fd04h], ax
mov     es:[0fd06h], bx

; DST_Y = y2
mov     ax, WORD PTR [bp+PARM+6]
mov     es:[0fd08h], ax
mov     es:[0fd0ah], bx

; DST_HEIGHT = height
mov     ax, WORD PTR [bp+PARM+10]
mov     es:[0fd14h], ax
mov     es:[0fd16h], bx

; DST_WIDTH = width
mov     ax, WORD PTR [bp+PARM+8]
mov     es:[0fd10h], ax
mov     es:[0fd12h], bx

; restore saved registers
pop     es
pop     bx

; remove frame pointer
mov     sp, bp
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            pop     bp

            ret

do_blit     endp

; ---------------------------------------------------------------------------
; VlineISR - Handler for VLINE interrupt
;
; The Mach64 can have one of four IRQs set: 2, 3, 5, 10. It is assumed that
; the CRTC_VLINE_INT_EN bit is set in the CRTC_INT_CNTL register.
; ---------------------------------------------------------------------------
            public  VlineISR

VlineISR    proc    far

            cli

            push    ax
            push    dx

            ; Check if this interrupt trigger belongs to the VLINE interrupt
            mov     dx, ioCRTC_INT_CNTL
            in      ax, dx
            and     ax, 18h
            jnz     acknowledge
            jmp     skip

acknowledge:
            ; Increment count
            inc     Count

            ; Ack 8259 interrupt controllers according to the IRQ number
            cmp     IRQnum, 0ah
            je      ack_2nd_8259

            ; ack master 8259 - required for IRQ 2, 3, 5
            mov     dx, 20h
            mov     al, 20h
            out     dx, al

            cmp     IRQnum, 2
            jne     ack_vline

ack_2nd_8259:
            ; ack cascaded 8259 - required for IRQ 2, 10
            mov     dx, 0a0h
            mov     al, 20h
            out     dx, al

ack_vline:
            ; ack vline interrupt bit on Mach64
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            mov     dx, ioCRTC_INT_CNTL
            in      ax, dx
            or      ax, 10h
            out     dx, ax

            ; ---- restore (if needed), save, and update frame ----

            ; don't restore unless x position is more the zero
            cmp     Xpos, 0
            je      no_restore

            ; restore from last frame
            mov     ax, Height
            push    ax
            mov     ax, Iwidth
            push    ax
            mov     ax, Ydraw
            push    ax
            mov     ax, Xpos
            sub     ax, Step
            push    ax
            mov     ax, SaveY
            push    ax
            mov     ax, SaveX
            push    ax
            call    FAR PTR do_blit
            pop     ax
            pop     ax
            pop     ax
            pop     ax
            pop     ax
            pop     ax

no_restore:
            ; save frame
            mov     ax, Height
            push    ax
            mov     ax, Iwidth
            push    ax
            mov     ax, SaveY
            push    ax
            mov     ax, SaveX
            push    ax
            mov     ax, Ydraw
            push    ax
            mov     ax, Xpos
            push    ax
            call    FAR PTR do_blit
            pop     ax
            pop     ax
            pop     ax
            pop     ax
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            pop     ax
            pop     ax

            ; update current frame
            mov     ax, Height
            push    ax
            mov     ax, Iwidth
            push    ax
            mov     ax, Ydraw
            push    ax
            mov     ax, Xpos
            push    ax
            mov     ax, SrcY
            push    ax
            mov     ax, SrcX
            push    ax
            call    FAR PTR do_blit
            pop     ax
            pop     ax
            pop     ax
            pop     ax
            pop     ax
            pop     ax

skip:
            pop     dx
            pop     ax

            pushf
            call    DWORD PTR OldIntVector

            sti

            iret

VlineISR    endp

; ---------------------------------------------------------------------------
; GETCOUNT
;
; Retrieve the count value controlled by the ISR.
;
; Inputs:  none
;
; Outputs: WORD count
; ---------------------------------------------------------------------------
            public  getcount

getcount    proc    far

            ; get count
            mov     al, Count
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            xor     ah, ah

            ret

getcount    endp

; ---------------------------------------------------------------------------
; SETBLITINFO
;
; Set blit dimension information for ISR.
;
; Inputs:  WORD image width,
;          WORD image height,
;          WORD source x,
;          WORD source y,
;          WORD save x,
;          WORD save y,
;          WORD y draw,
;          WORD step
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  setblitinfo

setblitinfo    proc    far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; Save used registers
            push    ax

            ; get image width
            mov     ax, WORD PTR [bp+PARM]
            mov     Iwidth, ax

            ; get image height
            mov     ax, WORD PTR [bp+PARM+2]
            mov     Height, ax

            ; get source x
            mov     ax, WORD PTR [bp+PARM+4]
            mov     SrcX, ax

            ; get source y
            mov     ax, WORD PTR [bp+PARM+6]
            mov     SrcY, ax

            ; get save x
            mov     ax, WORD PTR [bp+PARM+8]
            mov     SaveX, ax
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            ; get save y
            mov     ax, WORD PTR [bp+PARM+10]
            mov     SaveY, ax

            ; get y draw
            mov     ax, WORD PTR [bp+PARM+12]
            mov     Ydraw, ax

            ; get step
            mov     ax, WORD PTR [bp+PARM+14]
            mov     Step, ax

            ; restore saved registers
            pop     ax

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

setblitinfo    endp

; ---------------------------------------------------------------------------
; SETXSTART
;
; Set image starting position for the ISR.
;
; Inputs:  WORD x image position
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  setxstart

setxstart   proc    far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; Save used registers
            push    ax

            ; get image starting position (x)
            mov     ax, WORD PTR [bp+PARM]
            mov     Xstart, ax

            ; restore saved registers
            pop     ax

            ; remove frame pointer
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            mov     sp, bp
            pop     bp

            ret

setxstart   endp

; ---------------------------------------------------------------------------
; SETXPOS
;
; Set current image position for ISR.
;
; Inputs:  WORD x image position
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  setxpos

setxpos     proc    far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; Save used registers
            push    ax

            ; get current image position (x)
            mov     ax, WORD PTR [bp+PARM]
            mov     Xpos, ax

            ; restore saved registers
            pop     ax

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

setxpos     endp

; ---------------------------------------------------------------------------
; SETIMAGE
;
; Set source image for ISR.
;
; Inputs:  WORD source image,
;          WORD source x,
;          WORD source y
;
; Outputs: none
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; ------------------------------------------------------------------------
            public  setimage

setimage    proc    far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; Save used registers
            push    ax

            ; get source image
            mov     ax, WORD PTR [bp+PARM]
            mov     Image, ax

            ; get image coordinates
            mov     ax, WORD PTR [bp+PARM+2]
            mov     SrcX, ax
            mov     ax, WORD PTR [bp+PARM+4]
            mov     SrcY, ax

            ; restore saved registers
            pop     ax

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

setimage    endp

; ------------------------------------------------------------------------
; GETIRQINDEX
;
; Return the IRQ index given an IRQ number
;
; Inputs : WORD IRQ num
;            AX: valid values  : 2  , 3  , 5  , 0Ah
;
; Outputs: WORD IRQ index
;            AX: valid returns : 0Ah, 0Bh, 0Dh, 72h
; ------------------------------------------------------------------------
            public  getirqindex

getirqindex proc    far

            ; determine IRQ index based on IRQ number
            xor     ah, ah
            cmp     al, 2
            jne     chkirq3
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            mov     al, 0ah
            jmp     done

chkirq3:
            cmp     al, 3
            jne     chkirq5
            mov     al, 0bh
            jmp     done

chkirq5:
            cmp     al, 5
            jne     chkirq10
            mov     al, 0dh
            jmp     done

chkirq10:
            cmp     al, 0ah
            jne     badirqnum
            mov     al, 72h
            jmp     done

badirqnum:
            mov     al, 0

done:
            ret

getirqindex endp

; ---------------------------------------------------------------------------
; INITVLINEISR
;
; Chain in VLINE interrupt service routine.
;
; Inputs : WORD IRQnum
;            AX: valid values: 2, 3, 5, 0Ah
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  initvlineisr

initvlineisr proc    far

            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save used registers
            push    bx
            push    dx
            push    ds
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            push    es

            ; get IRQ number
            mov     ax, WORD PTR [bp+PARM]
            mov     IRQnum, al
            Mcall   getirqindex
            mov     IRQindex, al

            ; save IRQ mask for master 8259 (IRQs 0 - 7)
            mov     dx, 21h
            in      al, dx
            mov     IRQmask1, al

            ; save IRQ mask for cascaded 8259 (IRQs 8 - 15)
            mov     dx, 0a1h
            in      al, dx
            mov     IRQmask2, al

            ; load IRQ number and enable interrupts in appropriate 8259
            xor     ch, ch
            mov     cl, IRQnum

            ; if IRQ 2, 3, 5 - program master 8259
            ; if IRQ 10 - program cascaded 8259
            ; bit = 0 - enable interrupt
            ; bit = 1 - disable interrupt
            cmp     cl, 0ah
            je      irq10

            ; enable interrupt on master 8259 for IRQ 2, 3, 5
            mov     dx, 21h
            mov     ah, 0ffh
            mov     al, 1
shiftleft:
            shl     al, 1
            loop    shiftleft
            sub     ah, al
            mov     al, IRQmask1
            and     al, ah
            cli                         ; disable interrupts during mask update
            out     dx, al
            sti
            jmp     continue

irq10:
            ; enable interrupt on cascaded 8259 for IRQ 10 - IRQ 2 also needs
            ; to be enabled

            ; enable cascaded interrupt 2 on master 8259
            mov     ah, 0fbh
            mov     dx, 021h            ; IRQ 2 cascades to IRQ 10
            mov     al, IRQmask1
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            and     al, ah
            cli                     ; disable interrupts during mask update
            out     dx, al
            sti

            ; enable interrupt 10 on cascaded 8259
            mov     dx, 0a1h
            mov     al, IRQmask2
            and     al, ah
            cli                     ; disable interrupts during mask update
            out     dx, al
            sti

continue:
            ; initialize variables for ISR
            mov     ax, 0
            mov     Count, al

            ; get old IRQ vector address (in ES:BX)
            mov     al, IRQindex
            mov     ah, 35h
            int     21h

            ; save vector address
            mov     ax, es
            mov     OldIntVector, bx
            mov     OldIntVector+2, ax

            ; get address of interrupt service routine (in DS:DX)
            mov     dx, offset cs:VlineISR
            mov     ax, cs
            mov     ds, ax

            ; set new vector address
            mov     al, IRQindex
            mov     ah, 25h
            int     21h

            ; restore registers
            pop     es
            pop     ds
            pop     dx
            pop     bx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

initvlineisr endp
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; ---------------------------------------------------------------------------
; CANCELVLINEISR
;
; Disable timer interrupt routine.
;
; Inputs : none
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  cancelvlineisr

cancelvlineisr proc    far

            ; save used registers
            push    dx
            push    ds

            ; restore old IRQ vector address (in DS:DX)
            mov     dx, OldIntVector
            mov     ax, OldIntVector+2
            mov     ds, ax

            mov     al, IRQindex
            mov     ah, 25h
            int     21h

            ; restore IRQ masks - disable interrupts while updating masks
            cli
            mov     dx, 21h
            mov     al, IRQmask1
            out     dx, al
            mov     dx, 0a1h
            mov     al, IRQmask2
            out     dx, al
            sti

            ; restore registers
            pop     ds
            pop     dx

            ret

cancelvlineisr endp

; ---------------------------------------------------------------------------
; INTERRUPTS_OFF
;
; Disable system interrupts.
;
; Inputs : none
;
; Outputs: none
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; ----------------------------------------------------------------------------
            public  interrupts_off

interrupts_off proc far

            ; disable system interrupts
            cli

            ret

interrupts_off endp

; ----------------------------------------------------------------------------
; INTERRUPTS_ON
;
; Enable system interrupts.
;
; Inputs : none
;
; Outputs: none
; ----------------------------------------------------------------------------
            public  interrupts_on

interrupts_on proc far

            ; enable system interrupts
            sti

            ret

interrupts_on endp
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Single Buffering (Delta Framing)

Delta framing is a method of achieving flicker-free animation without CRT synchronization. Only the changes from one frame to the next are drawn on the screen. The animation will be flicker-free because no undrawing is ever done. Tearing will occur, but the effects will be minimal, given the draw rate.

Sample code

SAMPLE.DOC

```
[SAMPLE CODE FOR DELTA FRAMING TO FOLLOW]
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

OFF-SCREEN MEMORY MANAGEMENT

Off-screen memory management is a requirement for any real application that directly uses the accelerator. Hardware cursor definitions, context save areas, font caches, and bitmap caches are all kept in off-screen memory. Independent source and destination pitches and offsets, and a linear source trajectory facilitate implementation of an off-screen memory manager.

Memory can be allocated in linear chunks, aligned to 64-bit boundaries. One possible implementation of an off-screen memory manager is show below:

Sample code SAMPLE.DOC

```
[CODE SAMPLE FOR OFF-SCREEN MEMORY MANAGER TO FOLLOW]
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

CONTEXT CHAINS

Context chains are used to group many draw operations into a single atomic operation. This is useful for improving concurrency on engine-bound operations (see *Performance Issues* in *Chapter 5, Advanced Topics II*).

The CONTEXT_MASK entry in the context load structure can be used to inhibit loading of any register so that chains may take advantage of draw side effects.

Note that on loading, the CONTEXT_MASK register has no effect on the CONTEXT_MASK entry or the CONTEXT_LOAD_CNTL entry in the context save structure. These two registers are always loaded. To halt a context chain, the CONTEXT_LOAD_CNTL entry must be set to do nothing.

Sample code CHAIN.DOC

```
This is sample code to demonstrate chained context operations. See CHAIN.C
for more details.

----

unsigned long context[64];      // context array


/* Main C program */
int main(void)
{
    ...

    // get modal information for resolution independent operation
    xres = (unsigned long)(modeinfo.xres);
    yres = (unsigned long)(modeinfo.yres);
    pitch = (unsigned long)(modeinfo.pitch);

    // setup engine context and clear screen
    init_engine();
    clear_screen(0, 0, modeinfo.xres, modeinfo.yres);

    // determine top of memory address
    memcntl = inpw(ioMEM_CNTL);
    switch(memcntl & 7)
    {
        case 0: contextaddr = 0x80000; break;   // 512K
        case 1: contextaddr = 0x100000; break;  // 1M
        case 2: contextaddr = 0x200000; break;  // 2M
        case 3: contextaddr = 0x400000; break;  // 4M
        case 4: contextaddr = 0x600000; break;  // 6M
        case 5: contextaddr = 0x800000; break;  // 8M
    }
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
/*
   Fill context. Each context pointer represents 256 bytes of video
   memory. The context load address decreases as the context load pointer
   increases. Context pointer 0 points to the top of memory - 256. Some
   restrictions apply for context pointers 0-3 if the linear aperture
   size equals the memory size (i.e. 4M aperture size, 4M of video
   memory). In this case, the memory mapped registers occupy 1K of memory
   below the top of aperture. The only method to reach this area in this
   case is to use the VGA paged aperture.
*/
context[0] = 0xffffffff;
context[1] = 0x00000000;
context[2] = (pitch / 8) << 22;          // destination pitch
context[3] = 0x00200010;
context[4] = 0x00400080;
temp = (unsigned long)(modeinfo.xres / 20);
context[3] = (temp << 16) | (modeinfo.yres / 30);  // (x, y)
temp = (unsigned long)(modeinfo.xres / 10);
context[4] = (temp << 16) | (modeinfo.yres / 4);   // (width, height)
context[5] = 0x00000000;
context[6] = 0x00000001;
context[7] = 0x0003ffff;
context[8] = (pitch / 8) << 22;          // source pitch
context[9] = 0x00000000;
context[10] = 0x00000000;
context[11] = 0x00000000;
context[12] = 0x00000000;
context[13] = 0x00000000;
context[14] = 0x00000000;
context[15] = xres << 16;                // scissors
context[16] = yres << 16;
context[17] = 0x00000000;
context[18] = get_color_code(YELLOW);    // foreground color
context[19] = 0xffffffff;

// set DP_PIX_WIDTH and CHAIN_MASK according to color depth
switch(modeinfo.bpp)
{
    case 4:
        context[20] = 0x00008888;
        context[21] = HOST_4BPP | SRC_4BPP | DST_4BPP;
        break;
    case 16:
        if (modeinfo.depth == 555)
        {
            // 555 color weighting
            context[20] = 0x00004210;
            context[21] = HOST_15BPP | SRC_15BPP | DST_15BPP;
        }
        else
        {
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                // 565 color weighting
                context[20] = 0x00000410;
                context[21] = HOST_16BPP | SRC_16BPP | DST_16BPP;
            }
            break;
        case 32:
            context[20] = 0x00008080;
            context[21] = HOST_32BPP | SRC_32BPP | DST_32BPP;
            break;
        default:
        case 8:
        case 24:
            context[20] = 0x00008080;
            context[21] = HOST_8BPP | SRC_8BPP | DST_8BPP;
            break;
    }

    context[22] = FRGD_MIX_S | BKGD_MIX_S;
    context[23] = FRGD_SRC_FRGD_CLR;
    context[24] = 0x00000000;
    context[25] = 0xffffffff;
    context[26] = 0x00000000;
    context[27] = DST_LAST_PEL | DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT;
    context[28] = 0x00000000;
    for (i = 29; i < 64; i++)
    {
        context[i] = 0;
    }

    // Upload contexts

    // Setup 2 contexts - one chained to the other. This is done by setting
    // up the CONTEXT_LOAD_CNTL context register (offset 28) for the current
    // context load. When a context load is initiated, the CONTEXT_LOAD_CNTL
    // register will be loaded and executed according to its contents. This
    // feature allows context chaining. In this example, the first context
    // will draw a YELLOW filled rectangle and the second context will draw
    // a LIGHTGREEN diagonal line. Since the second context is to be chained, the
    // the load pointer bits of the CONTEXT_LOAD_CNTL register entry of the
    // FIRST context (load pointer = 4) will be set to point to the second
    // context (load pointer = 5).

    // Context load address calculation:
    //
    // Address = total video memory - (context pointer + 1) * 0x100)
    //

    /*
       The main focus here is to upload the context information for later
       retrieval by the engine context load feature. The method used will
       depend on the hardware setup - see CHAIN.C for more details.
    */
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    // load up first context -> load pointer = 4
    context[28] = CONTEXT_LOAD_AND_DO_LINE | 5;   // do line for next context
    upload_context(contextaddr - ((4 + 1) * 0x100));

    // load up second context -> load pointer = 5
    temp = (unsigned long)(modeinfo.xres / 4);
    context[3] = (temp << 16) | (modeinfo.yres / 4);   // (x, y) for line
    context[18] = get_color_code(LIGHTGREEN);     // set line color
    context[28] = 0;                              // halt chain
    upload_context(contextaddr - ((5 + 1) * 0x100));

    // initiate context chain
    wait_for_fifo(2);                             // wait for 2 fifo entries
    regw(CONTEXT_MASK, 0xffffffff);
    regw(CONTEXT_LOAD_CNTL, CONTEXT_LOAD_AND_DO_FILL | 4);

    ...
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Chapter 5
Advanced Topics II

BOOT-TIME INITIALIZATION

This section describes the registers required to initialize a *mach64* after power-up. All boot-time initialization is performed in the adapter ROM.

- The scratch registers, **SCRATCH_REG0** and **SCRATCH_REG1**, may be used at the adapter ROM's discretion, with the exception of the lower 7 bits of SCRATCH_REG1. These bits are used to communicate ROM segment location to applications, and must be initialized at boot-time. Typically, installed mode information and other flags are stored in the other bits.
- **BUS_CNTL** is used to configure the *mach64* bus interface unit and to control FIFO error and host error interrupts. At boot-time, all interrupts should be disabled, and the bus interface unit must be programmed appropriately for the type of host expansion bus. In determining the appropriate initialization values, safest values should be used first, and incrementally reduced until minimum safe values are discovered.
- **MEM_CNTL** is used to configure the memory interface unit. Memory size must be determined by the adapter ROM, and written appropriately. Initial memory boundary information should be stored in the non-volatile storage area. All other configuration bits are first determined empirically using the methods described for BUS_CNTL, and later hard-coded for particular memory configurations.
- **GEN_TEST_CNTL** is used for accessing an external EEPROM, enabling overscan to external DACs, enabling the hardware cursor, resetting the draw engine, enabling VRAM block write memory cycles, and chip diagnostic functions. At boot-time, overscan and block write must be initialized. The hardware cursor must be disabled, and the draw engine must be reset and enabled.
- **CONFIG_CNTL** is used for initializing the linear aperture and small apertures, setting the card ID, and disabling the VGA. The apertures should be disabled on power-up, and should only be initialized when an application calls the ROM for aperture services. The card ID should be set to zero in single-card systems. The VGA disable bit should never be touched.

- **CONFIG_CHIP_ID**, **CONFIG_STAT0**, and **CONFIG_STAT1** are used to determine board configuration for initialization, for ROM query functions, or for hardware debugging.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Of all the registers listed above, only CONFIG_CHIP_ID, GEN_TEST_CNTL, and SCRATCH_REG1 should be touched by applications. CONFIG_CHIP_ID is used to identify a specific class and revision of accelerator, and GEN_TEST_CNTL is used to enable the hardware cursor and reset the draw engine. No other bits should be touched in GEN_TEST_CNTL. SCRATCH_REG1 is used to determine the ROM segment location for calling ROM service routines.

ACCESSING THE EEPROM

The EEPROM interface pins are directly accessible from the lower four bits of GEN_TEST_CNTL. Note that these pins are multiplexed with display output pins, so garbage will appear on the display when reading or writing from the EEPROM. The CRTC should be disabled before any EEPROM operations. EEPROM operations are enabled with the GEN_EE_ENABLE bit.

- Note that EEPROMs are generally very slow devices, and the GEN_EE_CLOCK can only be strobed at the maximum rate specified by the device data sheets. Please consult the manufacturer's EEPROM specifications for more information on reading or writing from that device.
- Note that not all configurations of *mach64* will use an EEPROM for a non-volatile storage device.

Sample code SAMPLE.DOC

```
[CODE SAMPLE FOR READING AND WRITING FROM THE EEPROM TO FOLLOW]
```

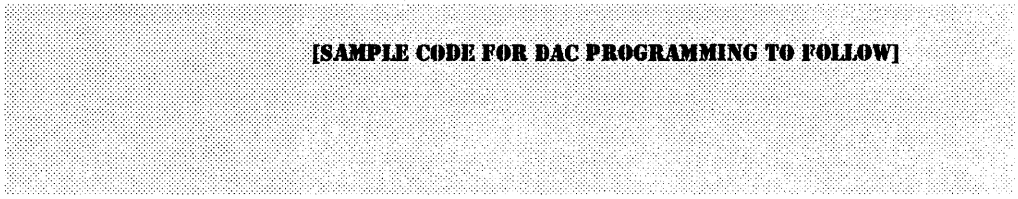

© 1993 ATI Technologies Inc.
Proprietary and Confidential

DAC PROGRAMMING

The *mach64* supports DACs from many manufacturers. All will be compatible with VGA resolutions (640x480, 800x600, 1024x768, at 8 bits per pixel, up to 80MHz pixel clock). Modes beyond these require special programming considerations.

DACs from different manufacturers will access the extended DAC registers in different ways. Some require a special sequence of register reads and writes to remap the register address space to its extended registers. Others have extra address lines accessible through DAC_EXT_SEL@DAC_CNTL to select extended DAC registers. Please consult the manufacturer's DAC specification for more information. Below is sample code for programming some common DAC types:

Sample code SAMPLE.DOC



© 1993 ATI Technologies Inc.
Proprietary and Confidential

DIAGNOSTIC FEATURES

Numerous self-diagnostic features have been designed into the *mach64*. Test modes are set using the GEN_TEST_CNTL register and the scratch registers perform different functions depending on the set mode. The test modes available are listed below:

Test Mode 0, All Test Features Disabled

The scratch registers are used as scratch registers.

Test Mode 1, Memory Read/Write Test

SCRATCH_REG0 is used as an indirect address register, and SCRATCH_REG1 is a data register. It is conceptually convenient to rename these register as TEST_REG0 and TEST_REG1. Strobing the GEN_TEST_MEM_STROBE@GEN_TEST_CNTL bit will read or write 32 bits of data into TEST_REG1 at the DWORD address specified in TEST_REG0. If GEN_TEST_MEM_WR@GEN_TEST_CNTL is set, data is written to memory from TEST_REG1; otherwise, data at the specified address is loaded into TEST_REG1.

Test Mode 2, Source and Destination Length Test

The addresses of SCRATCH_REG0 and SCRATCH_REG1 are remapped to point to internal counters. These are referred to as TEST_REG2 and TEST_REG3 and are read- only. By putting the chip into single step mode (GEN_TEST_DST_SS_EN@GEN_TEST_CNTL or GEN_TEST_SRC_SS_EN@GEN_TEST_CNTL), and by watching these registers while strobing one of the single step bits (GEN_TEST_DST_SS_STROBE@GEN_TEST_CNTL or GEN_TEST_SRC_SS_STROBE@GEN_TEST_CNTL), the integrity of these counters can be assured.

Test Mode 3, Source FIFO Read Length Counter Test

Similar to test mode 2, except that SCRATCH_REG0 is mapped to the source FIFO read length counter and is referred to TEST_REG4. The size of the source FIFO is 4x64.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Test Mode 4, CRTC Test

The internal CRTC counters may be tested in this mode by single stepping the CRTC with GEN_TEST_CC_EN@GEN_TEST_CNTL and GEN_TEST_CC_STROBE@GEN_TEST_CNTL. TEST_REG5 and TEST_REG6 are used to determine the internal state of the CRTC counters, flags, and comparators.

Test Mode 5, Display CRC Test

A CRC check of the display output can be done in this mode. It provides a convenient method of checking the DAC interface, overscan, and hardware cursor circuitry. Initiate the CRC by writing to the GEN_TEST_CRC_STR@GEN_TEST_CNTL bit, and waiting for the GEN_TEST_CRC_DONE@GEN_TEST_CNTL bit to go high. TEST_REG7 contains the accumulated CRC.

Other Test Features

Other test features include:

- The GEN_TEST_FIFO_EN@GEN_TEST_CNTL bit, which halts consumption of the command FIFO so that all FIFO entries may be guaranteed to be filled and tested.
- The GEN_TEST_GUI_REGS_EN@GEN_TEST_CNTL bit, which prevents draw operations and context loads from initiating. This allows all registers to be written and read without causing the draw engine to do unwanted draw operations.

All other registers, data paths, sequencers, and draw functions may be tested using standard draw operations, and comparing the results against known data.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Chapter 6
Performance Issues

PERFORMANCE ISSUES

Performance is a complex issue, which requires a clear definition of the terminology and an explanation of the factors affecting graphics performance.

Redundancy

Redundancy is the duplication of information. Most draw operations are redundant in that the same pixel or pattern of pixels is repeatedly written into memory. Since host expansion buses (ISA, EISA, MCA, VLB, PCI) tend to be slow, draw operations performed by the host CPU tend to be slow as well. Graphics accelerators improve performance by reducing the amount of redundant information traveling across the host expansion bus by simply specifying the type of pixel information to be written and the draw trajectory.

Any operation whose draw information cannot easily be reduced (such as a host-to-screen bitmap transfer) should do direct memory writes into the linear frame buffer instead of being drawn by the draw engine, because draw setup overhead will slow the operation.

Draw Speed

Draw speed is a raw measure of how fast the draw engine can put pixels to memory, measured in pixels per second. Many benchmark programs do not measure draw speed correctly because they do not factor in concurrency.

Concurrency

Concurrency is the inherent ability of graphics accelerators to perform a draw operation at the same time that the host CPU is doing something else. An accelerator is a fixed-function processor that performs dedicated tasks and relieves the CPU to do other tasks. Concurrency and reduction of redundancy are the

© 1993 ATI Technologies Inc.
Proprietary and Confidential

primary reasons why graphics accelerators are faster than dumb frame buffer devices, such as the VGA.

Efficiency

Efficiency is a measure of concurrency. Maximum efficiency in a software process is achieved when the host is never idle and the draw engine is never idle (this never happens). Efficiency will be affected by draw speed, CPU speed, FIFO depth, and order of draw operations (for example, a draw engine operation followed by a linear frame buffer access requires a wait-for-engine-idle in between, which causes the CPU to idle, thus decreasing efficiency).

Performance should be measured on both slow CPUs and fast CPUs, because efficiency differs radically from system to system.

Expansion Buses

There are currently five different **expansion bus** standards for X86 platforms:

- ISA
- EISA
- MCA
- VLB
- PCI.

Each differs in maximum and typical throughput. Bus type will only affect the performance of host-to-screen and screen-to-host transfers. Most other draw operations have very low redundancy, and bus transfer times are very small.

VRAM vs. DRAM

The CRT controller needs to fetch screen data at pixel clock rates. VRAM is dual-ported, and allows the CRTC to read memory while the draw engine is accessing the same memory. Some VRAMs also have specialized circuitry to do fast monochrome expansions. Accesses to DRAM must be arbitrated between the CRTC and draw engine (the CRTC has priority), thus reducing the total number of memory cycles available to the draw engine, and slowing draw speed. In DRAM configurations, graphics mode has a direct correlation to draw speed, because the higher the pixel clock rate, the less available memory bandwidth.

In summary, VRAMs are faster than DRAMs, but the difference will only be perceptible and measurable if the target application is engine-bound (i.e., CPU must wait for draw engine = poor efficiency).

Note that graphics benchmark programs are atypical because they have inherently low efficiency.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Block write

Block write is a high-speed color fill feature of VRAMs and some specialized types of DRAMs. Four consecutive addresses may be filled with a solid color in the time it takes to do a single memory access.

The *mach64* uses block write if GEN_BLOCK_WR_EN@GEN_TEST_CNTL is enabled, the foreground mix is set to paint (function 7), the background mix is set to transparent (function 3), the color compare function is set to FALSE, WRT_MASK is set to all '1's, destination pixel size is 8, 15, 16, or 32bpp, and DST_24_ROT_EN@DST_CNTL is disabled. Any monochrome source may be used. It is the adapter ROM's responsibility to enable GEN_BLOCK_WR_EN@GEN_TEST_CNTL at boot time if a compatible type of memory is detected.

Memory Bandwidth

Memory bandwidth is a measure of the number of memory accesses per second, which is easily quantifiable. On the *mach64*, a memory access to a current page costs two cycles; a page faulted memory access costs seven cycles.

Page

A **page** is defined as 512 addresses, where the data width may be 32 bits or 64 bits wide, depending on memory configuration. The frequency of page faulting depends on the burst rates of the various devices contending for the memory bus. The source FIFO is 4x64, and the CRTC video FIFO is 16x64.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Example: A screen-to-screen blit with dimensions 160x120, and a destination mix of XOR, at 30 frames per second in 8bpp mode, and a pitch of 1024 pixels; assume a memory clock of 50MHz, and data width of 64 bits.

Number of QWORDs in 160x120 area: (160x120 pixels / (8 pixels / QWORD)) = 2400 QWORDs

Number of memory accesses per QWORD: source-read + dest-read + dest-write = 3 accesses/QWORD

Note that the source read occurs because it is a screen-to screen-operation, and the destination read occurs because it is a read-modify-write destination mix.

Number of memory accesses: 2400 * 3 = 7200 accesses.

Memory page size: (512 x 64 bits) / (8 bits/pixel) / (1024 pixels/line) = 4 lines

Page faulting from operation size is so infrequent that we will ignore this factor.

Page faults from muxing source-reads and destination-read-modify-writes should occur every four memory accesses. Therefore, average access time is: (3 * 2 + 1 * 7) / 4 = 3.25 cycles/access

Number of memory cycles needed for a single blit: 7200 * 3.25 = 23400 cycles.

Draw speed = (160x120 pixels) / (23400 cycles / 50000000 cycles/sec) = 41 Mpixels/second.

Percent memory bandwidth used: (23400 cycle/frame * 30 frames/sec) / 50000000 cycles/sec * 100% = 1.4%.

Notes:
The average access time calculation will vary depending on a number of factors. Not aligning a source or destination edge to a QWORD boundary will increase the average access time by a small amount. A destination-write operation will page fault much less than the read-read-modify-write operation in the example above.

On DRAM configurations, the CRTC will cause the draw engine to page fault more frequently as well (the frequency can be calculated by computing the percent bandwidth the CRTC will use and using this ratio to approximate the page fault rate given the draw speed).

Also note that these calculations are only applicable to large draw operations. Draw engine setup overhead becomes much more significant for small operations.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Performance

Performance is a holistic measure, and depends greatly upon host configuration, accelerator configuration, and application efficiency. Performance cannot be quantified in a single measure.

- **System performance** can be improved with faster host and accelerator configurations.
- **Application performance** on a fixed hardware configuration can be improved by reducing redundancy and improving efficiency on a particular target system.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Appendix A
BIOS Services

INTRODUCTION

All accelerator services are provided in the accelerator ROM segment at offset 64h. The ROM segment is determined by reading the SCRATCH_REG1 register and calculating:

SEGMENT = (SCRATCH_REG1 & 0x7F) * 0x80 + 0xC000

where **SCRATCH_REG1** is **046ECh**

Function codes are placed in AL. Functions are called by first doing a PUSHF, and then far calling to SEGMENT:64h.

All functions return with error code in AH

```
AH    = 0    ; No error
AH    = 1    ; Function complete with error
AH    = 2    ; Function not supported
```

Function 0, load coprocessor CRTC parameters

```
CL[3-0    = color depth
          = 1          ; 4bpp
          = 2          ; 8bpp
          = 3          ; 15bpp (555)
          = 4          ; 16bpp (565)
          = 5          ; 24bpp
          = 6          ; 32bpp
CL[5]     = 1          ; Enable gamma correction if 15bpp and above
                       ; Set the RAMDAC to 8 bit if in 8bpp mode, for support of
                         256 color greyscale
CL[7-6]   = pitch size
          = 0          ; 1024
          = 1          ; Don't change
          = 2          ; Pitch is the same as horizontal display
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Function 0, load coprocessor CRTC parameters (cont'd)

```
CH      = Resolution
        = 12h       ; 640x480
        = 6ah       ; 800x600
        = 55h       ; 1024x768
        = 80h       ; Load table from offset of external storage (EEPROM) in BX
        = 81h       ; Load table according to data in DX:BX
        = 82h       ; OEM-specific mode
        = 83h       ; 1280x1024
DX:BX   =  Pointer to parameter table if CH = 81h
BX      = Offset into EEPROM table if CH = 80h
```

Function 1, set display mode

```
CL[0]   = 0     ; VGA and set the RAMDAC to 6-bit
        = 1     ; Coprocessor
CL[7]   = 1     ; Enable 8-bit DAC or Gamma Correction
                ; This bit is OR'ed with CL[5] in function AL = 0
```

Function 2, load coprocessor CRTC parameters and set display mode

```
Same arguments as AL = 0
```

Function 3, read EEPROM data

```
BX      = Index
returns
        BX = Data
```

Function 4, write EEPROM data

```
BX      = Index
DX      = Data
```

Function 5, memory aperture services

```
CL      = 0     ; Disable memory aperture
CL[0]   = 1     ; Enable memory aperture
CL[2]   = 1     ; Enable VGA memory aperture
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Function 6, short query function

```
AL[5-0]   = Aperture configuration
          = 0          ; Disable
          = 1          ; 4M
          = 2          ; 8M
AL[6]     = 0          ; Aperture address is user-configurable
          = 1          ; Aperture address in predefined or hard-coded in BIOS
AL[7]     = 0          ; Aperture address is in 128M range
          = 1          ; Aperture address is in 4G range
BX        = Aperture address
CH        = Color depth support (see offset 13 in the Query Structure table - page A-7)
CL        = Memory size
DX        = ASIC identification, [7-0] = revision, [15-8] = type
```

Function 7, return hardware capability List

In return DX:BX = offset into a table specifying the maximum dot clock information; the table is terminated by a zero in the first column (see below).

```
AL        = Format
          type
          = 0
```

H_DISP	DAC_MASK	Memory Requirements	Maximum Dot Clock	Pixel Width
0				

H_DISP = Horizontal resolution, in number of characters.

DAC_MASK = (1 shl dactype)

Memory Requirements = Minimum memory required to support the specified resolution and color depth (DRAM requirement shl4) or (VRAM requirement)

Maximum Dot Clock = Maximum dot clock with the specified resolution and color depth, in MHz

Pixel Width = Color depth

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Function 8, return query device data structure in bytes

```
on entry
CL[0]    = 0          ; Buffer size for header information only
         = 1          ; Buffer size for header information and mode tables
return
CX       = Number of bytes
```

Function 9, query device

```
DX:BX    = Pointer to buffer
CL[0]    = 0          ; Return header information only
         = 1          ; Return header information and mode table
```

Function 0Ah, return clock chip frequency table

```
AL       = Clock chip type
DX:BX    = Offset pointing to the 16 words containing the pre-programmed dot clock
           frequency; units are in KHz/10 (four significant digits)
DX:CX    = Offset pointing to the table containing clock chip information in the
           following format:
           db    Clock chip type
           db    Frequency table identification
           dw    Minfreq, maxfreq (in KHz/10)
           db    User-programmable entry if <>0ffh
           db    Reserved
           dw    Hardware-dependent information
```

Function 0Bh, program a specified clock entry

```
CH       = Entry in the frequency table
BX       = Units are in KHz/10
in return
         AL = Clock chip type
         BX = Programming word depending on type
```

Function 0Ch, DPMS service, set DPMS mode

```
CL[1-0]  = 0          ; Active
         = 1          ; Standby
         = 2          ; Suspend
         = 3          ; Off
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

Function 0Dh, return current DPMS state in CL

Function 0Eh, set graphics controller's power management state

Function 0Fh, return current graphics controller's current power management state

Function 10h, set RAMDAC to different states

CL	= 80	; Reserved
CL[0]	= 0	; Set RAMDAC to normal mode
CL[1]	= 1	; Set RAMDAC to sleep mode

Function 11h, return external storage device information (INSTALL should use this information to dynamically configure the data structure)

CL	= External data structure information	
CL[7]	= 1	; No external data storage can be used; everything is predefined
	= 0	; External data storage available
CL[6-4]	= 000	; External data is readable and writable
	= 001	; External data storage is readable but not writable
	= 011	; External data storage is not readable or writable
	= 100	; External data storage is readable and writable; writing must be handled by the application program, based on *device type* in CL[3-0]
CL[3-0]	= 0	; device type
CH	= Number of 16-bit entries in the storage device.	
DH	= Number of 16-bit entries in the storage device, these entries are read-only	
BL	= Offset into the CRTC parameter table	
BH	= Size of the CRTC parameter table; if the number is smaller than the one in the CRTC table, discard the bottom ones	

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Function 12, Short Query

on return
AX = *Reserved*
BX = *Reserved*
CX = *Reserved*
DX = IO base address

Query Structure	
Byte Offset	**Description**
0-1	Size of structure in bytes
2	Revision of structure
3	Number of mode tables
4-5	Offset in bytes to mode tables
6	Size of each mode table in bytes
8-9	ASIC identification
7	VGA Type: 0 = disabled 1 = enabled
0Ah	VGA Boundary: 0 = full access 1 = 256K 2 = 512K 3 = 768K 4 = 1M 10h = no access through VGA
0Bh	Memory Size: 0 = 512K 1 = 1M 2 = 2M 3 = 4M 4 = 6M 5 = 8M 6 = 12M 7 = 8M
0Ch	Bits 3-0, DAC Type: 0 = *Reserved* 1 = *Reserved* 2 = TI 34075/ATI68875 3 = Brooktree BT476/8 4 = Brooktree BT481, AT&T20C490/491, SC15025/15026, IMS-G174, MU9C4910, MU9C1880 5 = ATI68860 6 = STG1700 7 = SC15021 Bits 7-4 = *Reserved*

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Query Structure	
Byte Offset	**Description**
0Dh	Memory Type: 0 = DRAM 256Kx16 1 = VRAM 256Kx4 2 = VRAM 256Kx16 3 = DRAM 256Kx4 5 = VRAM 256x4 special 6 = VRAM 256x16 special
0Eh	Bus Type: 0 = ISA 1 = EISA 2 = *Reserved* 3 = *Reserved* 4 = *Reserved* 5 = *Reserved* 6 = VLB 7 = PCI
0Fh	Bit 7 - Enable composite sync Bit 6 - Enable sync on green
10h:11h	Aperture address in megabytes (0-4095)
12h	Aperture configuration (see extended BIOS function AL=6)
13h	Color Depth Support Bit Definition 7 = 1 ; *Reserved* 6 = 1 ; *Reserved* 5 = 1 ; *Reserved* 4 = 1 ; support 32bpp (unpack 24 bpp) 3 = 1 ; support BGR in 24bpp 2 = 1 ; support RGB in 24bpp 1 = 1 ; support 16 bpp, 555 0 = 1 ; support 16 bpp, 565
14h	RAMDAC support feature Bit Definition 7 = 1 ; support sync on green 6 = 1 ; support gamma correction 5 = 1 ; support 256 grey scale 4 = 1 ; support sleep mode
15h	Reserved
16h:17h	Offset into current mode table if non-zero (not implemented)
18h:19h	I/O base address
18h:1Fh	Reserved

Mode tables immediately follow the device status table. Use the forward pointer to reference mode tables, as the device status table may expand in the future. It is possible to have no modes installed. Typically, between two and seven mode tables will be returned.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Mode Table Structure	
Byte Offset	**Description**
Installed Mode Table 1	
0:1	Horizontal display resolution, in pixels
2:3	Vertical display resolution, in scanlines
4	Maximum pixel depth (see *function 0, CL[3-0]*, on page A-1 for interpretation)
5	*Reserved*
6:7	Offset into EEPROM = 0 ;Table is generated from VGA parameters <>0 ;offset into EEPROM table
8:9	*Reserved*
0Ah:0Bh	*Reserved*
0Ch:0Dh	Bits 15:11= *Reserved* Bit 10 = Enable MUX mode Bit 9 = Enable interlace Bit 8 = Enable double scan Bits 7:0 = *Reserved*
0Eh	CRTC_H_TOTAL
0Fh	CRTC_H_DISP
10h	CRTC_H_SYNC_STRT
11h	CRTC_H_SYNC_WID
12h:13h	CRTC_V_TOTAL
14h:15h	CRTC_V_DISP
16h:17h	CRTC_V_SYNC_STRT
18h	CRTC_V_SYNC_WID
19h	CLOCK_CNTL
1Ah:1Bh	Dot Clock for coprocessor mode, for programmable clock chip
1Ch:1Dh	Bits 15 -12 = CRTC_H_TOTAL_DLYBit 3 - 2 = OVR_WID_LEFT Bits 11 - 8 = CRTC_H_SYNC_DLY Bits 7 - 4 = OVR_WID_RIGHT Bits 3 - 0 = OVR_WID_LEFT
1Eh:1Fh	OVR_WID_T0P, OVR_WID_BOTTOM
20h:21h	OVR_CLR_B, OVR_CLR_8
22h:23h	OVR_CLR_G, OVR_CLR_R
Installed Mode Table 2	
24h:47h	Entries definition same as mode table 1.
. . .	
Installed Mode Table n	
N*24h-(N*24+23h)	Entries definition same as mode table 1.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Appendix B
EEPROM Map

CX EEPROM DATA STRUCTURE		
Offset	**Bits**	**Description**
0h	15:0	EEPROM Write Counter
1h	15:8	*Reserved*
	7:0	EEPROM checksum, modular 8 of 8-bit data; the sum of all the entries in the EEPROM must be 0
2h	15:0	*Reserved* — no application program should change the factory default, zero
3h	15:4	*Reserved*
	3:0	EEPROM table revision
4h	15:0	Custom monitor indices
5h	15:8	1280x1024 refresh rate information
	7	= 1; Use stored 640x480 coprocessor parameters for coprocessor mode
	6	= 1; Enable 640x480 72Hz
	5:2	*Reserved*
	1	Enable sync on green
	0	Enable composite sync
6h	15:8	*Reserved*
	7	= 1; Use stored 800x600 coprocessor parameters for coprocessor mode
	6	*Reserved*
	5	= 1; Select 800x600 in 72Hz
	4	= 1; Select 800x600 in 70Hz
	3	= 1; Select 800x600 in 60Hz
	2	= 1; Select 800x600 in 56Hz
	1	= 1; Select 800x600 in 89Hz interlaced
	0	= 1; Select 800x600 in 85Hz interlaced

© 1993 ATI Technologies Inc.
Proprietary and Confidential

CX EEPROM DATA STRUCTURE		
Offset	**Bits**	**Description**
7h	15:8	*Reserved*
	7	= 1; Use stored 1024x768 coprocessor parameters for coprocessor mode
	6:4	*Reserved*
	3	= 1; Select 1024x768 in 72Hz
	2	= 1; Select 1024x768 in 70Hz
	1	= 1; Select 1024x768 in 60Hz
	0	= 1; Select 1024x768 in 87Hz interlaced
8h	15:8	Power-Up Video Mode: 3 = VGA color - secondary 5 = VGA monochrome - secondary 9 = VGA color - primary B = VGA monochrome - primary
	7:6	Monochrome Mode Color Select: 0 = White 1 = Green 2 = Amber
	5	Dual monitor enable
	4	Font Selection at power-up: 0 = 8x14 or 9x14 1 = 8x16 or 9x16
	3	VGA Bus I/O: 0 = 8-bit 1 = 16-bit
	2	Zero Wait State Ram 0 = Disabled 1 = Enabled
	1	Zero Wait State ROM 0 = Disabled 1 = Enabled
	0	16 Bit ROM 0 = Disabled 1 = Enabled

© 1993 ATI Technologies Inc.
Proprietary and Confidential

CX EEPROM DATA STRUCTURE		
Offset	**Bits**	**Description**
9h	15:14	Host data transfer width: 0 = Auto select 1 = 16-bit 2 = 8-bit 3 = 8-bit host, 16-bit others
	13:8	Monitor code
	7:6	*Reserved*
	5:4	VGA boundary: 0 = No boundary 1 = 512KB 2 = 1MB
	3	Monitor Alias enable
	2:0	Monitor alias
Ah	15:4	Aperture Location (in MByte)
	3:0	Aperture Size (BIOS will not use this value if Aperture Location is a non-zero value — aperture size will be based on amount of video memory installed)
Bh	15:8	Mouse address: 00h = Mouse disabled 08h = Secondary address selected 18h = Primary address selected
	7:0	Interrupt level: 20h = IRQ 5 28h = IRQ 4 30h = IRQ 3 38h = IRQ 2
0Ch:16h		*Reserved*
17h:25h		CRT parameter table 1
26h:34h		CRT parameter table 2
35h:43h		CRT parameter table 3
44h:52h		CRT parameter table 4
53h:61h		CRT parameter table 5
62h:70h		CRT parameter table 6
71h:7Fh		CRT parameter table 7

© 1993 ATI Technologies Inc.
Proprietary and Confidential

CX CRT PARAMETER TABLE			
Offset	**Bits**	**Description**	
		VGA Parameters	**Accelerator Parameters**
0h	15:8	Video mode select 1	*Reserved*
	7:0	Video mode select 2	*Reserved*
1h	15:8	Video mode select 3	Video mode select
	7:0	CRT refresh rate bit mask	0x80 - for coprocessor mode
2h	15:14	*Reserved*	
	13	Mux mode enable	
	12:0	*Reserved*	
	9	Interlace enable	
	8	Double scan enable	
	7	Vertical sync polarity (VGA only)	
	6	Horizontal sync polarity (VGA only)	
	5	*Reserved*	
	4	CRT usage (VGA only) 0 = Use sync polarities only 1 = Use all CRT parameters	-
	3:0	*Reserved*	
3h	15:8	MAX_SCAN_LINE (CRT09)	CRTC_H_DISP
	7:0	H_TOTAL (CRT00)	CRTC_H_TOTAL
4h	15:8	H_RETRACE_END (CRT05)	CRTC_H_SYNC_WID
	7:0	RETRACE_STRT (CRT04)	CRTC_H_SYNC_STRT
5h	15:8	V_RETRACE_END (CRT11)	CRTC_V_TOTAL (15:8)
	7:0	V_RETRACE_STRT (CRT10)	CRTC_V_TOTAL (7:0)
6h	15:8	H_BLANK_END (CRT03)	CRTC_V_DISP (15:8)
	7:0	H_BLANK_STRT (CRT02)	CRTC_V_DISP (7:0)
7h	15:8	V_BLANK_END (CRT16)	CRTC_V_SYNC_STRT (15:8)
	7:0	V_BLANK_STRT (CRT15)	CRTC_V_SYNC_STRT (7:0)
8h	15:8	CRTC_OVERFLOW (CRT07) If == 0ffh or == programmable entry in clock chip, use Dot Clock in entry 9 and programmable entry in dot clock.	CLOCK_CNTL
	7:0	V_TOTAL (CRT06)	CRTC_V_SYNC_WIDTH
9h	15:8	V_DISP_END (CRT12)	Dot clock (15:8)
	7:0	CRT_MODE (CRT17)	Dot Clock (7:0)
Ah	15:0	bits 15:12 = *Reserved*	
		bits 11:8 = CRTC_H_SYNC_DLY	
		bits 7:4 = OVR_WID_RIGHT	
		bits 3:0 = OVR_WID_LEFT	

© 1993 ATI Technologies Inc.
Proprietary and Confidential

CX CRT PARAMETER TABLE			
Offset	**Bits**	**Description**	
		VGA Parameters	**Accelerator Parameters**
Bh	15:0	OVR_WID_TOP, OVR_WID_BOTTOM	
Ch	15:0	OVR_CLR_8, OVR_CLR_B	
Dh	15:0	OVR_CLR_G, OVR_CLR_R	
Eh	15:0	*Reserved*	

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SCRATCH REGISTERS	
SCRATCH_REG0 + 1 (42EDh)	800x600 refresh rate information
SCRATCH_REG0 + 3 (42EFh)	1024x768 refresh rate information
SCRATCH_REG1 (46ECh)	ROM location
SCRATCH_REG1 + 2 (46EEh)	
bit 7:6	CRTC pitch size
bit 5	mux mode
bit 4	Enable gamma correction or 256 color greyscale
bit 3	TI output clock select information
bit 0	current gamma correction or 356 color state
SCRATCH_REG1 + 3 (46EFh)	Programmable dot clock information
1CE/BB	
bit 7:6	640x480 refresh rate information
bit 5:4	monochrome mode; color information
bit 0	Set to VGA display if int0 is called

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Appendix C
CRT Parameters

640x480 60Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x14		
DOT_CLOCK		25.18MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x63	V_TOTAL	0x20C
Screen Display	H_DISP	0x4F	V_DISP	0x1DF
Sync Start	H_SYNC_STRT	0x52	V_SYNC_STRT	0x1EA
Sync Width	H_SYNC_WID	0x2C	V_SYNC_WID	0x22
Resolution	640		480	
Scan Frequency	31.469KHz		59.94Hz	
Polarity	(-)		(-)	
Sync Width	3.813us	12 chars	0.064ms	2 lines
Front Porch	0.953us	3 chars	0.350ms	11 lines
Back Porch	1.589us	5 chars	1.017ms	32 lines
Active Time	25.422us	80 chars	15.253ms	480 lines
Blank Time	6.356us	20 chars	1.430ms	45 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

640x480 60Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x14		
DOT_CLOCK		25.18MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x63	V_TOTAL	0x20C
Screen Display	H_DISP	0x4F	V_DISP	0x1DF
Sync Start	H_SYNC_STRT	0x52	V_SYNC_STRT	0x1EA
Sync Width	H_SYNC_WID	0x2C	V_SYNC_WID	0x22
Resolution	640		480	
Scan Frequency	31.469KHz		59.94Hz	
Polarity	(-)		(-)	
Sync Width	3.813us	12 chars	0.064ms	2 lines
Front Porch	0.953us	3 chars	0.350 ms	11 lines
Back Porch	1.589us	5 chars	1.017ms	32 lines
Active Time	25.422us	80 chars	15.253ms	480 lines
Blank Time	6.356us	20 chars	1.430ms	45 lines

640x480 72Hz NON-INTERLACED/32

GEN_CNTL		0x00		
CLOCK_CNTL		0x09		
DOT_CLOCK		32.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x6A	V_TOTAL	0x207
Screen Display	H_DISP	0x4F	V_DISP	0x1DF
Sync Start	H_SYNC_STRT	0x52	V_SYNC_STRT	0x1E8
Sync Width	H_SYNC_WID	0x25	V_SYNC_WID	0x23
Resolution	640		480	
Scan Frequency	37.383KHz		71.89Hz	
Polarity	(-)		(-)	
Sync Width	1.250us	5chars	0.080ms	3lines
Front Porch	0.750us	3chars	0.241ms	9lines
Back Porch	4.750us	19chars	0.749ms	28lines
Active Time	20.000us	80chars	12.840ms	480lines
Blank Time	6.750us	27chars	1.070ms	40lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

640x480 72Hz NON-INTERLACED/40

GEN_CNTL		0x00		
CLOCK_CNTL		0x1B		
DOT_CLOCK		40.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x6F	V_TOTAL	0x266
Screen Display	H_DISP	0x4F	V_DISP	0x1DF
Sync Start	H_SYNC_STRT	0x57	V_SYNC_STRT	0x211
Sync Width	H_SYNC_WID	0x30	V_SYNC_WID	0x2C
Resolution	640		480	
Scan Frequency	44.643KHz		72.59Hz	
Polarity	(-)		(-)	
Sync Width	3.200us	16 chars	0.269ms	12 lines
Front Porch	1.600us	8 chars	1.120ms	50 lines
Back Porch	1.600us	8 chars	1.635ms	73 lines
Active Time	16.000us	80 chars	10.752ms	480 lines
Blank Time	6.400us	32 chars	3.024ms	135 lines

800x600 89Hz INTERLACED

GEN_CNTL		0x02		
CLOCK_CNTL		0x1F		
DOT_CLOCK		32.50MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x80	V_TOTAL	0x2BC
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x65	V_SYNC_STRT	0x262
Sync Width	H_SYNC_WID	0x4	V_SYNC_WID	0x2C
Resolution	800		600	
Scan Frequency	31.492KHz		89.85Hz	
Polarity	(+)		(-)	
Sync Width	0.985us	4 chars	0.191ms	12 lines
Front Porch	0.492us	2 chars	0.175ms	11 lines
Back Porch	5.662us	23 chars	1.238ms	78 lines
Active Time	24.615us	100 chars	9.526ms	600 lines
Blank Time	7.138us	29 chars	1.604ms	101 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

800x600 95Hz INTERLACED

GEN_CNTL		0X02		
CLOCK_CNTL		0X03		
DOT_CLOCK		36.00MHz		
	Horizontal		Vertical	
Screen Total	H_TOTAL	0x84	V_TOTAL	0x2BC
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x6D	V_SYNC_STRT	0x262
Sync Width	H_SYNC_WID	0x10	V_SYNC_WID	0xC
Resolution	800		600	
Scan Frequency	33.835KHz		96.53Hz	
Polarity	(-)		(-)	
Sync Width	3.556us	16 chars	0.177ms	12 lines
Front Porch	2.222us	10 chars	0.163ms	11 lines
Back Porch	1.556us	7 chars	1.153ms	78 lines
Active Time	22.222us	100 chars	8.867ms	600 lines
Blank Time	7.333us	33 chars	1.493ms	101 lines

800x600 56Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x03		
DOT_CLOCK		36.00MHz		
	Horizontal		Vertical	
Screen Total	H_TOTAL	0x7F	V_TOTAL	0x270
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x66	V_SYNC_STRT	0x258
Sync Width	H_SYNC_WID	0x9	V_SYNC_WID	0x2
Resolution	800		600	
Scan Frequency	35.156KHz		56.25Hz	
Polarity	(+)		(+)	
Sync Width	2.000us	9 chars	0.057ms	2 lines
Front Porch	0.667us	3 chars	0.028ms	1 lines
Back Porch	3.556us	16 chars	0.626ms	22 lines
Active Time	22.222us	100 chars	17.067ms	600 lines
Blank Time	6.222us	28 chars	0.711ms	25 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

800x600 60Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0C		
DOT_CLOCK		39.91MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x83	V_TOTAL	0x273
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x68	V_SYNC_STRT	0x258
Sync Width	H_SYNC_WID	0x10	V_SYNC_WID	0x4
Resolution	800		600	
Scan Frequency	37.794KHz		60.18Hz	
Polarity	(+)		(+)	
Sync Width	3.207us	16 chars	0.106ms	4 lines
Front Porch	1.002us	5 chars	0.026ms	1 lines
Back Porch	2.205us	11 chars	0.609ms	23 lines
Active Time	20.045us	100 chars	15.876ms	600 lines
Blank Time	6.414us	32 chars	0.741ms	28 lines

800x600 70Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x07		
DOT_CLOCK		44.90MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x7D	V_TOTAL	0x27B
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x64	V_SYNC_STRT	0x260
Sync Width	H_SYNC_WID	0x12	V_SYNC_WID	0x2C
Resolution	800		600	
Scan Frequency	44.544KHz		70.04Hz	
Polarity	(+)		(-)	
Sync Width	3.207us	18 chars	0.269ms	12 lines
Front Porch	0.178us	1 chars	0.202ms	9 lines
Back Porch	1.247us	7 chars	0.337ms	15 lines
Active Time	17.817us	100 chars	13.470ms	600 lines
Blank Time	4.633us	26 chars	0.808ms	36 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

800x600 72Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x04		
DOT_CLOCK		50.35MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x82	V_TOTAL	0x29B
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x6A	V_SYNC_STRT	0x27C
Sync Width	H_SYNC_WID	0xF	V_SYNC_WID	0x6
Resolution	800		600	
Scan Frequency	48.044KHz		71.92Hz	
Polarity	(+)		(+)	
Sync Width	2.383us	15 chars	0.125ms	6 lines
Front Porch	1.112us	7 chars	0.770ms	37 lines
Back Porch	1.430us	9 chars	0.520ms	25 lines
Active Time	15.889us	100 chars	12.489ms	600 lines
Blank Time	4.926us	31 chars	1.415ms	68 lines

800x600 76Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x05		
DOT_CLOCK		56.64MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x86	V_TOTAL	0x2B1
Screen Display	H_DISP	0x63	V_DISP	0x257
Sync Start	H_SYNC_STRT	0x6D	V_SYNC_STRT	0x27E
Sync Width	H_SYNC_WID	0x28	V_SYNC_WID	0x24
Resolution	800		600	
Scan Frequency	52.444KHz		76.01Hz	
Polarity	(-)		(-)	
Sync Width	1.130us	8 chars	0.076ms	4 lines
Front Porch	1.412us	10 chars	0.744ms	39 lines
Back Porch	2.401us	17 chars	0.896ms	47 lines
Active Time	14.124us	100 chars	11.441ms	600 lines
Blank Time	4.944us	35 chars	1.716ms	90 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1024x768 87Hz INTERLACED

GEN_CNTL		0x02		
CLOCK_CNTL		0X07		
DOT_CLOCK		44.90MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x9D	V_TOTAL	0x330
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x81	V_SYNC_STRT	0x300
Sync Width	H_SYNC_WID	0x16	V_SYNC_WID	0x8
Resolution	1024		768	
Scan Frequency	35.522KHz		86.96Hz	
Polarity	(+)		(+)	
Sync Width	3.920us	22 chars	0.113ms	8 lines
Front Porch	0.356us	2 chars	0.014ms	1 lines
Back Porch	1.069us	6 chars	0.563ms	40 lines
Active Time	22.806us	128 chars	10.810ms	768 lines
Blank Time	5.345us	30 chars	0.690ms	49 lines

1024x768 56Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x05		
DOT_CLOCK		56.64MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0x9C	V_TOTAL	0x323
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x89	V_SYNC_STRT	0x30D
Sync Width	H_SYNC_WID	0x7	V_SYNC_WID	0x9
Resolution	1024		768	
Scan Frequency	45.096KHz		56.09Hz	
Polarity	(+)		(+)	
Sync Width	0.989us	7 chars	0.200ms	9 lines
Front Porch	1.412us	10 chars	0.310ms	14 lines
Back Porch	1.695us	12 chars	0.288ms	13 lines
Active Time	18.079us	128 chars	17.031ms	768 lines
Blank Time	4.096us	29 chars	0.798ms	36 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1024x768 60Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0F		
DOT_CLOCK		65.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xA7	V_TOTAL	0x31F
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x85	V_SYNC_STRT	0x300
Sync Width	H_SYNC_WID	0x8	V_SYNC_WID	0x4
Resolution	1024		768	
Scan Frequency	48.363KHz		60.45Hz	
Polarity	(+)		(+)	
Sync Width	0.985us	8 chars	0.083ms	4 lines
Front Porch	0.738us	6 chars	0.021ms	1 lines
Back Porch	3.200us	26 chars	0.558ms	27 lines
Active Time	15.754us	128 chars	15.880ms	768 lines
Blank Time	4.923us	40 chars	0.662ms	32 lines

1024x768 66Hz NON-INTERLACED/75

GEN_CNTL		0x00		
CLOCK_CNTL		0x0E		
DOT_CLOCK		75.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xad	V_TOTAL	0x32f
Screen Display	H_DISP	0x7f	V_DISP	0x2ff
Sync Start	H_SYNC_STRT	0x85	V_SYNC_STRT	0x307
Sync Width	H_SYNC_WID	0x16	V_SYNC_WID	0x4
Resolution	1024		768	
Scan Frequency	53.879KHz		66.03Hz	
Polarity	(-)		(-)	
Sync Width	2.347us	22 chars	0.074ms	4 lines
Front Porch	0.640us	6 chars	0.148ms	8 lines
Back Porch	1.920us	18 chars	0.668ms	36 lines
Active Time	13.653us	128 chars	14.254ms	768 lines
Blank Time	4.907us	46 chars	0.891ms	48 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1024X768 66Hz NON-INTERLACED/72

GEN_CNTL		0x00		
CLOCK_CNTL		0x06		
DOT_CLOCK		71.66MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xA5	V_TOTAL	0x32F
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x81	V_SYNC_STRT	0x307
Sync Width	H_SYNC_WID	0x16	V_SYNC_WID	0x4
Resolution	1024		768	
Scan Frequency	53.963KHz		66.13Hz	
Polarity	(+)		(+)	
Sync Width	2.456us	22 chars	0.074ms	4 lines
Front Porch	0.223us	2 chars	0.148ms	8 lines
Back Porch	1.563us	14 chars	0.667ms	36 lines
Active Time	14.289us	128 chars	14.232ms	768 lines
Blank Time	4.242us	38 chars	0.889ms	48 lines

1024X768 70Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0E		
DOT_CLOCK		75.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xA6	V_TOTAL	0x323
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x83	V_SYNC_STRT	0x301
Sync Width	H_SYNC_WID	0x16	V_SYNC_WID	0x8
Resolution	1024		768	
Scan Frequency	56.138KHz		69.82Hz	
Polarity	(+)		(+)	
Sync Width	2.347us	22 chars	0.143ms	8 lines
Front Porch	0.427us	4 chars	0.036ms	2 lines
Back Porch	1.387us	13 chars	0.463ms	26 lines
Active Time	13.653us	128 chars	13.681ms	768 lines
Blank Time	4.160us	39 chars	0.641ms	36 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1024x768 72Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0E		
DOT_CLOCK		75.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xA1	V_TOTAL	0x325
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x82	V_SYNC_STRT	0x302
Sync Width	H_SYNC_WID	0x32	V_SYNC_WID	0x26
Resolution	10224		768	
Scan Frequency	57.870KHz		71.80Hz	
Polarity	(-)		(-)	
Sync Width	1.920us	18 chars	0.104ms	6 lines
Front Porch	0.320us	3 chars	0.052ms	3 lines
Back Porch	1.387us	13 chars	0.501ms	29 lines
Active Time	13.653us	128 chars	13.271ms	768 lines
Blank Time	3.627us	34 chars	0.657ms	38 lines

1024x768 76Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0B		
DOT_CLOCK		80.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xA2	V_TOTAL	0x326
Screen Display	H_DISP	0x7F	V_DISP	0x2FF
Sync Start	H_SYNC_STRT	0x87	V_SYNC_STRT	0x307
Sync Width	H_SYNC_WID	0xB	V_SYNC_WID	0x4
Resolution	1024		768	
Scan Frequency	61.350KHz		76.02Hz	
Polarity	(+)		(+)	
Sync Width	1.100us	11 chars	0.065ms	4 lines
Front Porch	0.800us	8 chars	0.130ms	8 lines
Back Porch	1.600us	16 chars	0.440ms	27 lines
Active Time	12.800us	128 chars	12.518ms	768 lines
Blank Time	3.500us	35 chars	0.636ms	39 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1120x750 87Hz INTERLACED

GEN_CNTL		0x02		
CLOCK_CNTL		0x04		
DOT_CLOCK		50.35MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xB0	V_TOTAL	0x330
Screen Display	H_DISP	0x8B	V_DISP	0x2ED
Sync Start	H_SYNC_STRT	0x8D	V_SYNC_STRT	0x2EE
Sync Width	H_SYNC_WID	0x19	V_SYNC_WID	0x8
Resolution	1120		750	
Scan Frequency	35.558KHz		87.05Hz	
Polarity	(+)		(+)	
Sync Width	3.972us	25 chars	0.112ms	8 lines
Front Porch	0.318us	2 chars	0.014ms	1 lines
Back Porch	1.589us	10 chars	0.816ms	58 lines
Active Time	22.244us	140 chars	10.546ms	750 lines
Blank Time	5.879us	37 chars	0.942ms	67 lines

1120X750 60Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0F		
DOT_CLOCK		65.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xAB	V_TOTAL	0x312
Screen Display	H_DISP	0x8B	V_DISP	0x2ED
Sync Start	H_SYNC_STRT	0x91	V_SYNC_STRT	0x2F3
Sync Width	H_SYNC_WID	0xC	V_SYNC_WID	0x4
Resolution	1120		750	
Scan Frequency	47.238KHz		60.02Hz	
Polarity	(+)		(+)	
Sync Width	1.477us	12 chars	0.085ms	4 lines
Front Porch	0.738us	6 chars	0.127ms	6 lines
Back Porch	1.723us	14 chars	0.572ms	27 lines
Active Time	17.231us	140 chars	15.877ms	750 lines
Blank Time	3.938us	32 chars	0.783ms	37 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1120X750 70Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0B		
DOT_CLOCK		80.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xAE	V_TOTAL	0x32D
Screen Display	H_DISP	0x8B	V_DISP	0x2ED
Sync Start	H_SYNC_STRT	0x95	V_SYNC_STRT	0x2FC
Sync Width	H_SYNC_WID	0xF	V_SYNC_WID	0xA
Resolution	1120		750	
Scan Frequency	57.143KHz		70.20Hz	
Polarity	(+)		(+)	
Sync Width	1.50us	15 chars	0.175ms	10 lines
Front Porch	1.00us	10 chars	0.263ms	15 lines
Back Porch	1.00us	10 chars	0.683ms	39 lines
Active Time	14.00us	140 chars	13.125ms	750 lines
Blank Time	3.50us	35 chars	1.120ms	64 lines

1280x1024 87Hz INTERLACED

GEN_CNTL		0x02		
CLOCK_CNTL		0x0B		
DOT_CLOCK		80.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xC7	V_TOTAL	0x47C
Screen Display	H_DISP	0x9F	V_DISP	0x3FF
Sync Start	H_SYNC_STRT	0xA9	V_SYNC_STRT	0x431
Sync Width	H_SYNC_WID	0xA	V_SYNC_WID	0xA
Resolution	1024		1024	
Scan Frequency	50.000KHz		87.03Hz	
Polarity	(+)		(+)	
Sync Width	1.000us	10 chars	0.100ms	10 lines
Front Porch	1.000us	10 chars	0.500ms	50 lines
Back Porch	2.000us	20 chars	0.650ms	65 lines
Active Time	16.000us	160 chars	10.240ms	1024 lines
Blank Time	4.000us	40 chars	1.250ms	125 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1280x1024 95Hz INTERLACED

GEN_CNTL		0x02		
CLOCK_CNTL		0x0B		
DOT_CLOCK		80.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xC7	V_TOTAL	0x41C
Screen Display	H_DISP	0x9F	V_DISP	0x3FF
Sync Start	H_SYNC_STRT	0xA9	V_SYNC_STRT	0x400
Sync Width	H_SYNC_WID	0xA	V_SYNC_WID	0xA
Resolution	1280		1024	
Scan Frequency	50.000KHz		94.97Hz	
Polarity	(+)		(+)	
Sync Width	1.000us	10 chars	0.100ms	10 lines
Front Porch	1.000us	10 chars	0.010ms	1 lines
Back Porch	2.000us	20 chars	0.180ms	18 lines
Active Time	16.000us	160 chars	10.240ms	1024 lines
Blank Time	4.000us	40 chars	0.290ms	29 lines

1280x1024 60Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x0A		
DOT_CLOCK		110.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xD6	V_TOTAL	0x42A
Screen Display	H_DISP	0x9F	V_DISP	0x3FF
Sync Start	H_SYNC_STRT	0xA9	V_SYNC_STRT	0x400
Sync Width	H_SYNC_WID	0xE	V_SYNC_WID	0x5
Resolution	1280		1024	
Scan Frequency	63.953KHz		59.94Hz	
Polarity	(+)		(+)	
Sync Width	1.018us	14 chars	0.078ms	5 lines
Front Porch	0.727us	10 chars	0.016ms	1 lines
Back Porch	2.255us	31 chars	0.579ms	37 lines
Active Time	11.636us	160 chars	16.012ms	1024 lines
Blank Time	4.000us	55 chars	0.672ms	43 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

1280x1024 70Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x01		
DOT_CLOCK		126.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xD2	V_TOTAL	0x429
Screen Display	H_DISP	0x9F	V_DISP	0x3FF
Sync Start	H_SYNC_STRT	0xA9	V_SYNC_STRT	0x400
Sync Width	H_SYNC_WID	0xA	V_SYNC_WID	0x5
Resolution	1280		1024	
Scan Frequency	74.645KHz		70.02Hz	
Polarity	(+)		(+)	
Sync Width	0.889us	14 chars	0.067ms	5 lines
Front Porch	0.635us	10 chars	0.013ms	1 lines
Back Porch	1.714us	27 chars	0.482ms	36 lines
Active Time	10.159us	60 chars	13.718ms	1024 lines
Blank Time	3.238us	51 chars	0.563ms	42 lines

1280x1024 74Hz NON-INTERLACED

GEN_CNTL		0x00		
CLOCK_CNTL		0x08		
DOT_CLOCK		135.00MHz		
	Horizontal		**Vertical**	
Screen Total	H_TOTAL	0xD5	V_TOTAL	0x427
Screen Display	H_DISP	0x9F	V_DISP	0x3FF
Sync Start	H_SYNC_STRT	0xA3	V_SYNC_STRT	0x3FF
Sync Width	H_SYNC_WID	0x12	V_SYNC_WID	0x1E
Resolution	1280		1024	
Scan Frequency	78.855KHz		74.11Hz	
Polarity	(+)		(+)	
Sync Width	1.067us	18 chars	0.380ms	30 lines
Front Porch	0.237us	4 chars	0.000ms	0 lines
Back Porch	1.896us	32 chars	0.127ms	10 lines
Active Time	9.481us	160 chars	12.986ms	1024 lines
Blank Time	3.200us	54 chars	0.507ms	40 lines

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Appendix D
Clock Chip Reference

ATI18811-0	
Select	**PCLK (MHz)**
0h	42.95
1h	48.77
2h	92.40
3h	36.00
4h	50.35
5h	56.64
6h	External
7h	44.90
8h	30.24
9h	32.00
Ah	110.00
Bh	80.00
Ch	39.91
Dh	44.90
Eh	75.00
Fh	65.00

ATI18811-1	
Select	**PCLK (MHz)**
0h	100.00
1h	126.00
2h	92.40
3h	36.00
4h	50.35
5h	56.64
6h	External
7h	44.90
8h	135.00
9h	32.00
Ah	110.00
Bh	80.00
Ch	39.91
Dh	44.90
Eh	75.00
Fh	65.00

ATI18811-2	
Select	**PCLK (MHz)**
0h	100.00
1h	126.00
2h	92.40
3h	36.00
4h	50.35
5h	56.64
6h	External
7h	44.90
8h	135.00
9h	32.00
Ah	110.00
Bh	80.00
Ch	39.91
Dh	44.90
Eh	75.00
Fh	65.00

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Appendix E
Register Summary

The *mach64* ASIC has the following five register classes:

- **VGA registers**
- **Setup and control registers**
- **Accelerator CRTC and DAC registers**
- **Draw engine context control registers**
- **Draw engine trajectory control registers.**

VGA REGISTERS

VGA registers are completely segregated from the accelerator registers. Their functions are mutually exclusive. They are addressed at I/O ports 1CE-1CFh, 3B0-3BFh, 3C0-3CFh, 3D0-3DFh. See the *mach64 Accelerator VGA Registers Guide* for more details.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

SETUP AND CONTROL REGISTERS

Setup and control registers are memory-mapped, and are also aliased at I/O base address 2EC-2EFh. Most of these registers are initialized once only, at boot time.

Setup and Control Registers			
Name	I/O Address	Memory Offset	R/W
SCRATCH_REG0	42ECh	20h	R/W
SCRATCH_REG1	46ECh	21h	R/W
BUS_CNTL	4EECh	28h	R/W
MEM_CNTL	52ECh	2Ch	R/W
MEM_VGA_WP_SEL	56ECh	2Dh	R/W
MEM_VGA_RP_SEL	5AECh	2Eh	R/W
GEN_TEST_CNTL	66ECh	34h	R/W
CONFIG_CNTL	6AECh	-	R/W
CONFIG_CHIP_ID	6EECh	38h	R
CONFIG_STAT	72ECh	39h	R

© 1993 ATI Technologies Inc.
Proprietary and Confidential

ACCELERATOR CRTC AND DAC REGISTERS

The CRTC and DAC registers are memory-mapped, and are also aliased at I/O base address 2EC-2EFh. The accelerator CRTC registers are separate from the VGA CRTC registers.

Accelerator CRTC and DAC Registers			
Name	I/O Address	Memory Offset	R/W
CRTC_H_TOTAL_DISP	7EECh	0h	R/W
CRTC_H_SYNC_STRT_WID	6ECh	1h	R/W
CRTC_V_TOTAL_DISP	AECh	2h	R/W
CRTC_V_SYNC_START_WID	EECh	3h	R/W
CRTC_VLINE_CRNT_VLINE	12ECh	4h	R/W
CRTC_OFF_PITCH	16ECh	5h	R/W
CRTC_INT_CNTL	1AECh	6h	R/W
CRTC_GEN_CNTL	1EECh	7h	R/W
OVR_CLR	22ECh	10h	R/W
OVR_WID_LEFT_RIGHT	26ECh	11h	R/W
OVR_WID_TOP_BOTTOM	2AECh	12h	R/W
CUR_CLR0	2EECh	18h	R/W
CUR_CLR1	32ECh	19h	R/W
CUR_OFFSET	36ECh	1Ah	R/W
CUR_HORZ_VERT_POSN	3AECh	1Bh	R/W
CUR_HORZ_VERT_OFFSET	3EECh	1Ch	R/W
CLOCK_SEL_CNTL	4AECh	24h	R/W
DAC_REGS	5EECh	30h	R/W
DAC_CNTL	62ECh	31h	R/W

© 1993 ATI Technologies Inc.
Proprietary and Confidential

DRAW ENGINE CONTEXT CONTROL REGISTERS

Draw engine context control registers are memory-mapped. They are used to set up the draw engine data path and destination mixing logic.

Draw Engine Context Control and Status Registers			
Name	Offset	R/W	Description
PAT_REG0	A0h	R/W	Pattern register 0, for fixed patterns
PAT_REG1	A1h	R/W	Pattern register 1, for fixed patterns
PAT_CNTL	A2h	R/W	Pattern control, for enabling fixed patterns
SC_LEFT	A8h	R/W	Scissor left
SC_RIGHT	A9h	R/W	Scissor right
SC_LEFT_RIGHT	AAh	W	Scissor left and right
SC_TOP	ABh	R/W	Scissor top
SC_BOTTOM	ACh	R/W	Scissor bottom
SC_TOP_BOTTOM	ADh	W	Scissor top and bottom
DP_BKGD_CLR	B0h	R/W	Background color
DP_FRGD_CLR	B1h	R/W	Foreground color
DP_WRITE_MASK	B2h	R/W	Write mask
DP_CHAIN_MASK	B3h	R/W	ALU carry chain mask (only used for (D+S)/2)
DP_PIX_WIDTH	B4h	R/W	Pixel width, for setting source, destination, and host pixel widths
DP_MIX	B5h	R/W	Mix, for setting foreground and background mixes
DP_SRC	B6h	R/W	Source, for setting mono source, and foreground and background sources
CLR_CMP_CLR	C0h	R/W	Compare color
CLR_CMP_MSK	C1h	R/W	Compare mask
CLR_CMP_CNTL	C2h	R/W	Compare control, for setting compare function and compare source
FIFO_STAT	C4h	R	Fifo status
CONTEXT_MASK	C8	R/W	Context load mask, for selectively loading draw engine registers
CONTEXT_SAVE_PTR	CAh	R/W	Context save pointer, in units of 64 DWORDs. Writing to this register saves the engine context
CONTEXT_LOAD_PTR	CBh	R/W	Context load pointer, for restoring an engine context with an option to perform a draw operation
GUI_TRAJ_CNTL	CCh	R/W	Trajectory control, for setting DST_CNTL, SRC_CNTL, HOST_CNTL, and PAT_CNTL with a single memory access
GUI_STAT	CEh	R	Engine status

© 1993 ATI Technologies Inc.
Proprietary and Confidential

DRAW ENGINE TRAJECTORY CONTROL REGISTERS

Draw engine trajectory control registers are memory-mapped. They are used to set up the source and destination trajectories and to initiate draw operations.

Draw Engine Trajectory Control Registers				
Name	Offset	R/W	Init	Description
DST_OFF_PITCH	40h	R/W		Destination offset and pitch
DST_X	41h	R/W		Destination X
DST_Y	42h	R/W		Destination Y
DST_Y_X	43h	W		Destination Y and X
DST_WIDTH	44h	R/W	Yes	Destination width
DST_HEIGHT	45h	R/W		Destination height
DST_HEIGHT_WIDTH	46h	W	Yes	Destination height and width.
DST_X_WIDTH	47h	W	Yes	Destination X and width
DST_BRES_LNTH	48h	R/W	Yes	Bresenham line length
DST_BRES_ERR	49h	R/W		Bresenham error term
DST_BRES_INC	4Ah	R/W		Bresenham axial step term
DST_BRES_DEC	4Bh	R/W		Bresenham diagonal step term
DST_CNTL	4Ch	R/W		Destination control, for setting destination trajectory direction, destination side effects, line and polygon options, and packed 24 bit initial rotation value
SRC_OFF_PITCH	60h	R/W		Source offset and pitch
SRC_X	61h	R/W		Source X
SRC_Y	62h	R/W		Source Y
SRC_Y_X	63h	W		Source Y and source X
SRC_WIDTH1	64h	R/W		Source width1 — for setting the source width for unbounded Y and general pattern sources, for setting the minor source width for general patterns with rotation
SRC_HEIGHT1	65h	R/W		Source height1 — for setting the source height for general pattern sources, for setting the minor source height for general patterns with rotation
SRC_HEIGHT1_WIDTH1	66h	W		Source height1 and source width1
SRC_X_START	67h	R/W		Source X start — for setting the starting X location for general patterns with rotation

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Draw Engine Trajectory Control Registers				
Name	Offset	R/W	Init	Description
SRC_Y_START	68h	R/W		Source Y start — for setting the starting Y location for general patterns with rotation
SRC_Y_X_START	69h	W		Source Y start and source X start
SRC_WIDTH2	6Ah	R/W		Source width2 — for setting the major source width for general patterns with rotation
SRC_HEIGHT2	6Bh	R/W		Source height2 — for setting the major source height for general patterns with rotation
SRC_HEIGHT2_WIDTH2	6Ch	W		Source height 2 and source width2
SRC_CNTL	6Dh	R/W		Source control — for setting source trajectory type and source trajectory modifiers
HOST_DATA[0-15]	80:8Fh	W		Host registers 0 to 15 are identical but mapped to 16 separate locations
HOST_CNTL	90h	R/W		Host control — for setting host consumption modifiers

MEMORY MAPPING

All memory-mapped registers reside at offset 3FFC00h from the base aperture address on 4MB boards or 7FFC00h on 8MB boards. If VGA is enabled, these registers are aliased at offset FC00h from B0000h.

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Appendix F
Sample Code

mach64 SAMPLE CODE

Microsoft® C compilers and MASM compilers

All **C source sample code** is compatible with Microsoft® C compilers (version 5.1 and newer). All **assembler source sample code** is compatible with Microsoft® MASM compiler version 5.1.

ATIM64.INC

```
; ============================================================================
; ATIM64.INC
;
; ASM include file that contains ATI MACH64 register definitions
;
; Copyright (c) 1993 ATI Technologies Inc.  All rights reserved
; ============================================================================

; NON-GUI IO MAPPED Registers

ioCRTC_H_TOTAL_DISP      equ  02ECh
ioCRTC_H_SYNC_STRT_WID   equ  06ECh
ioCRTC_V_TOTAL_DISP      equ  0AECh
ioCRTC_V_SYNC_STRT_WID   equ  0EECh
ioCRTC_VLINE_CRNT_VLINE  equ  12ECh
ioCRTC_OFF_PITCH         equ  16ECh
ioCRTC_INT_CNTL          equ  1AECh
ioCRTC_GEN_CNTL          equ  1EECh

ioOVR_CLR                equ  22ECh
ioOVR_WID_LEFT_RIGHT     equ  26ECh
ioOVR_WID_TOP_BOTTOM     equ  2AECh

ioCUR_CLR0               equ  2EECh
ioCUR_CLR1               equ  32ECh
ioCUR_OFFSET             equ  36ECh
```

```
ioCUR_HORZ_VERT_POSN    equ  3AECh
ioCUR_HORZ_VERT_OFF     equ  3EECh

ioSCRATCH_REG0          equ  42ECh
ioSCRATCH_REG1          equ  46ECh

ioCLOCK_SEL_CNTL        equ  4AECh

ioBUS_CNTL              equ  4EECh

ioMEM_CNTL              equ  52ECh
ioMEM_VGA_WP_SEL        equ  56ECh
ioMEM_VGA_RP_SEL        equ  5AECh

ioDAC_REGS              equ  5EECh
ioDAC_CNTL              equ  62ECh

ioGEN_TEST_CNTL         equ  66ECh

ioCONFIG_CNTL           equ  6AECh
ioCONFIG_CHIP_ID        equ  6EECh
ioCONFIG_STAT0          equ  72ECh
ioCONFIG_STAT1          equ  76ECh


; NON-GUI MEMORY MAPPED Registers - expressed in BYTE offsets */

CRTC_H_TOTAL_DISP       equ  0000h  // Dword offset 00h
CRTC_H_SYNC_STRT_WID    equ  0004h  // Dword offset 01h
CRTC_V_TOTAL_DISP       equ  0008h  // Dword offset 02h
CRTC_V_SYNC_STRT_WID    equ  000Ch  // Dword offset 03h
CRTC_VLINE_CRNT_VLINE   equ  0010h  // Dword offset 04h
CRTC_OFF_PITCH          equ  0014h  // Dword offset 05h
CRTC_INT_CNTL           equ  0018h  // Dword offset 06h
CRTC_GEN_CNTL           equ  001Ch  // Dword offset 07h

OVR_CLR                 equ  0040h  // Dword offset 10h
OVR_WID_LEFT_RIGHT      equ  0044h  // Dword offset 11h
OVR_WID_TOP_BOTTOM      equ  0048h  // Dword offset 12h

CUR_CLR0                equ  0060h  // Dword offset 18h
CUR_CLR1                equ  0064h  // Dword offset 19h
CUR_OFFSET              equ  0068h  // Dword offset 1Ah
CUR_HORZ_VERT_POSN      equ  006Ch  // Dword offset 1Bh
CUR_HORZ_VERT_OFF       equ  0070h  // Dword offset 1Ch

SCRATCH_REG0            equ  0080h  // Dword offset 20h
SCRATCH_REG1            equ  0084h  // Dword offset 21h

CLOCK_SEL_CNTL          equ  0090h  // Dword offset 24h

BUS_CNTL                equ  00A0h  // Dword offset 28h
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
MEM_CNTL                equ  00B0h  // Dword offset 2Ch

MEM_VGA_WP_SEL          equ  00B4h  // Dword offset 2Dh
MEM_VGA_RP_SEL          equ  00B8h  // Dword offset 2Eh

DAC_REGS                equ  00C0h  // Dword offset 30h
DAC_CNTL                equ  00C4h  // Dword offset 31h

GEN_TEST_CNTL           equ  00D0h  // Dword offset 34h

CONFIG_CHIP_ID          equ  00E0h  // Dword offset 38h
CONFIG_STAT0            equ  00E4h  // Dword offset 39h
CONFIG_STAT1            equ  00E8h  // Dword offset 3Ah


; GUI MEMORY MAPPED Registers

DST_OFF_PITCH           equ  0100h  // Dword offset 40h
DST_X                   equ  0104h  // Dword offset 41h
DST_Y                   equ  0108h  // Dword offset 42h
DST_Y_X                 equ  010Ch  // Dword offset 43h
DST_WIDTH               equ  0110h  // Dword offset 44h
DST_HEIGHT              equ  0114h  // Dword offset 45h
DST_HEIGHT_WIDTH        equ  0118h  // Dword offset 46h
DST_X_WIDTH             equ  011Ch  // Dword offset 47h
DST_BRES_LNTH           equ  0120h  // Dword offset 48h
DST_BRES_ERR            equ  0124h  // Dword offset 49h
DST_BRES_INC            equ  0128h  // Dword offset 4Ah
DST_BRES_DEC            equ  012Ch  // Dword offset 4Bh
DST_CNTL                equ  0130h  // Dword offset 4Ch

SRC_OFF_PITCH           equ  0180h  // Dword offset 60h
SRC_X                   equ  0184h  // Dword offset 61h
SRC_Y                   equ  0188h  // Dword offset 62h
SRC_Y_X                 equ  018Ch  // Dword offset 63h
SRC_WIDTH1              equ  0190h  // Dword offset 64h
SRC_HEIGHT1             equ  0194h  // Dword offset 65h
SRC_HEIGHT1_WIDTH1      equ  0198h  // Dword offset 66h
SRC_X_START             equ  019Ch  // Dword offset 67h
SRC_Y_START             equ  01A0h  // Dword offset 68h
SRC_Y_X_START           equ  01A4h  // Dword offset 69h
SRC_WIDTH2              equ  01A8h  // Dword offset 6Ah
SRC_HEIGHT2             equ  01ACh  // Dword offset 6Bh
SRC_HEIGHT2_WIDTH2      equ  01B0h  // Dword offset 6Ch
SRC_CNTL                equ  01B4h  // Dword offset 6Dh

HOST_DATA0              equ  0200h  // Dword offset 80h
HOST_DATA1              equ  0204h  // Dword offset 81h
HOST_DATA2              equ  0208h  // Dword offset 82h
HOST_DATA3              equ  020Ch  // Dword offset 83h
HOST_DATA4              equ  0210h  // Dword offset 84h
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
HOST_DATA5              equ  0214h  // Dword offset 85h
HOST_DATA6              equ  0216h  // Dword offset 86h
HOST_DATA7              equ  021Ch  // Dword offset 87h
HOST_DATA8              equ  0220h  // Dword offset 88h
HOST_DATA9              equ  0224h  // Dword offset 89h
HOST_DATAA              equ  0228h  // Dword offset 8Ah
HOST_DATAB              equ  022Ch  // Dword offset 8Bh
HOST_DATAC              equ  0230h  // Dword offset 8Ch
HOST_DATAD              equ  0234h  // Dword offset 8Dh
HOST_DATAE              equ  0238h  // Dword offset 8Eh
HOST_DATAF              equ  023Ch  // Dword offset 8Fh
HOST_CNTL               equ  0240h  // Dword offset 90h

PAT_REG0                equ  0280h  // Dword offset A0h
PAT_REG1                equ  0284h  // Dword offset A1h
PAT_CNTL                equ  0288h  // Dword offset A2h

SC_LEFT                 equ  02A0h  // Dword offset A8h
SC_RIGHT                equ  02A4h  // Dword offset A9h
SC_LEFT_RIGHT           equ  02A8h  // Dword offset AAh
SC_TOP                  equ  02ACh  // Dword offset ABh
SC_BOTTOM               equ  02B0h  // Dword offset ACh
SC_TOP_BOTTOM           equ  02B4h  // Dword offset ADh

DP_BKGD_CLR             equ  02C0h  // Dword offset B0h
DP_FRGD_CLR             equ  02C4h  // Dword offset B1h
DP_WRITE_MASK           equ  02C8h  // Dword offset B2h
DP_CHAIN_MASK           equ  02CCh  // Dword offset B3h
DP_PIX_WIDTH            equ  02D0h  // Dword offset B4h
DP_MIX                  equ  02D4h  // Dword offset B5h
DP_SRC                  equ  02D8h  // Dword offset B6h

CLR_CMP_CLR             equ  0300h  // Dword offset C0h
CLR_CMP_MASK            equ  0304h  // Dword offset C1h
CLR_CMP_CNTL            equ  0308h  // Dword offset C2h

FIFO_STAT               equ  0310h  // Dword offset C4h

CONTEXT_MASK            equ  0320h  // Dword offset C8h
CONTEXT_LOAD_CNTL       equ  032Ch  // Dword offset CBh

GUI_TRAJ_CNTL           equ  0330h  // Dword offset CCh
GUI_STAT                equ  0338h  // Dword offset CEh
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

ATIM64.H

```
/*=========================================================================
  ATIM64.H

  C include file that contains ATI MACH64 register definitions

  Copyright (c) 1993 ATI Technologies Inc.  All rights reserved
 =========================================================================*/

/* NON-GUI IO MAPPED Registers */

#define ioCRTC_H_TOTAL_DISP      0x02EC
#define ioCRTC_H_SYNC_STRT_WID   0x06EC
#define ioCRTC_V_TOTAL_DISP      0x0AEC
#define ioCRTC_V_SYNC_STRT_WID   0x0EEC
#define ioCRTC_VLINE_CRNT_VLINE  0x12EC
#define ioCRTC_OFF_PITCH         0x16EC
#define ioCRTC_INT_CNTL          0x1AEC
#define ioCRTC_GEN_CNTL          0x1EEC

#define ioOVR_CLR                0x22EC
#define ioOVR_WID_LEFT_RIGHT     0x26EC
#define ioOVR_WID_TOP_BOTTOM     0x2AEC

#define ioCUR_CLR0               0x2EEC
#define ioCUR_CLR1               0x32EC
#define ioCUR_OFFSET             0x36EC
#define ioCUR_HORZ_VERT_POSN     0x3AEC
#define ioCUR_HORZ_VERT_OFF      0x3EEC

#define ioSCRATCH_REG0           0x42EC
#define ioSCRATCH_REG1           0x46EC

#define ioCLOCK_SEL_CNTL         0x4AEC

#define ioBUS_CNTL               0x4EEC

#define ioMEM_CNTL               0x52EC
#define ioMEM_VGA_WP_SEL         0x56EC
#define ioMEM_VGA_RP_SEL         0x5AEC

#define ioDAC_REGS               0x5EEC
#define ioDAC_CNTL               0x62EC

#define ioGEN_TEST_CNTL          0x66EC

#define ioCONFIG_CNTL            0x6AEC
#define ioCONFIG_CHIP_ID         0x6EEC
#define ioCONFIG_STAT0           0x72EC
#define ioCONFIG_STAT1           0x76EC
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
/* NON-GUI MEMORY MAPPED Registers - expressed in BYTE offsets */

#define CRTC_H_TOTAL_DISP       0x0000  // Dword offset 00
#define CRTC_H_SYNC_STRT_WID    0x0004  // Dword offset 01
#define CRTC_V_TOTAL_DISP       0x0008  // Dword offset 02
#define CRTC_V_SYNC_STRT_WID    0x000C  // Dword offset 03
#define CRTC_VLINE_CRNT_VLINE   0x0010  // Dword offset 04
#define CRTC_OFF_PITCH          0x0014  // Dword offset 05
#define CRTC_INT_CNTL           0x0018  // Dword offset 06
#define CRTC_GEN_CNTL           0x001C  // Dword offset 07

#define OVR_CLR                 0x0040  // Dword offset 10
#define OVR_WID_LEFT_RIGHT      0x0044  // Dword offset 11
#define OVR_WID_TOP_BOTTOM      0x0048  // Dword offset 12

#define CUR_CLR0                0x0060  // Dword offset 18
#define CUR_CLR1                0x0064  // Dword offset 19
#define CUR_OFFSET              0x0068  // Dword offset 1A
#define CUR_HORZ_VERT_POSN      0x006C  // Dword offset 1B
#define CUR_HORZ_VERT_OFF       0x0070  // Dword offset 1C

#define SCRATCH_REG0            0x0080  // Dword offset 20
#define SCRATCH_REG1            0x0084  // Dword offset 21

#define CLOCK_SEL_CNTL          0x0090  // Dword offset 24

#define BUS_CNTL                0x00A0  // Dword offset 28

#define MEM_CNTL                0x00B0  // Dword offset 2C

#define MEM_VGA_WP_SEL          0x00B4  // Dword offset 2D
#define MEM_VGA_RP_SEL          0x00B8  // Dword offset 2E

#define DAC_REGS                0x00C0  // Dword offset 30
#define DAC_CNTL                0x00C4  // Dword offset 31

#define GEN_TEST_CNTL           0x00D0  // Dword offset 34

#define CONFIG_CHIP_ID          0x00E0  // Dword offset 38
#define CONFIG_STAT0            0x00E4  // Dword offset 39
#define CONFIG_STAT1            0x00E8  // Dword offset 3A


/* GUI MEMORY MAPPED Registers */

#define DST_OFF_PITCH           0x0100  // Dword offset 40
#define DST_X                   0x0104  // Dword offset 41
#define DST_Y                   0x0108  // Dword offset 42
#define DST_Y_X                 0x010C  // Dword offset 43
#define DST_WIDTH               0x0110  // Dword offset 44
#define DST_HEIGHT              0x0114  // Dword offset 45
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
#define DST_HEIGHT_WIDTH        0x0118  // Dword offset 46
#define DST_X_WIDTH             0x011C  // Dword offset 47
#define DST_BRES_LNTH           0x0120  // Dword offset 48
#define DST_BRES_ERR            0x0124  // Dword offset 49
#define DST_BRES_INC            0x0128  // Dword offset 4A
#define DST_BRES_DEC            0x012C  // Dword offset 4B
#define DST_CNTL                0x0130  // Dword offset 4C

#define SRC_OFF_PITCH           0x0180  // Dword offset 60
#define SRC_X                   0x0184  // Dword offset 61
#define SRC_Y                   0x0188  // Dword offset 62
#define SRC_Y_X                 0x018C  // Dword offset 63
#define SRC_WIDTH1              0x0190  // Dword offset 64
#define SRC_HEIGHT1             0x0194  // Dword offset 65
#define SRC_HEIGHT1_WIDTH1      0x0198  // Dword offset 66
#define SRC_X_START             0x019C  // Dword offset 67
#define SRC_Y_START             0x01A0  // Dword offset 68
#define SRC_Y_X_START           0x01A4  // Dword offset 69
#define SRC_WIDTH2              0x01A8  // Dword offset 6A
#define SRC_HEIGHT2             0x01AC  // Dword offset 6B
#define SRC_HEIGHT2_WIDTH2      0x01B0  // Dword offset 6C
#define SRC_CNTL                0x01B4  // Dword offset 6D

#define HOST_DATA0              0x0200  // Dword offset 80
#define HOST_DATA1              0x0204  // Dword offset 81
#define HOST_DATA2              0x0208  // Dword offset 82
#define HOST_DATA3              0x020C  // Dword offset 83
#define HOST_DATA4              0x0210  // Dword offset 84
#define HOST_DATA5              0x0214  // Dword offset 85
#define HOST_DATA6              0x0218  // Dword offset 86
#define HOST_DATA7              0x021C  // Dword offset 87
#define HOST_DATA8              0x0220  // Dword offset 88
#define HOST_DATA9              0x0224  // Dword offset 89
#define HOST_DATAA              0x0228  // Dword offset 8A
#define HOST_DATAB              0x022C  // Dword offset 8B
#define HOST_DATAC              0x0230  // Dword offset 8C
#define HOST_DATAD              0x0234  // Dword offset 8D
#define HOST_DATAE              0x0238  // Dword offset 8E
#define HOST_DATAF              0x023C  // Dword offset 8F
#define HOST_CNTL               0x0240  // Dword offset 90

#define PAT_REG0                0x0280  // Dword offset A0
#define PAT_REG1                0x0284  // Dword offset A1
#define PAT_CNTL                0x0288  // Dword offset A2

#define SC_LEFT                 0x02A0  // Dword offset A8
#define SC_RIGHT                0x02A4  // Dword offset A9
#define SC_LEFT_RIGHT           0x02A8  // Dword offset AA
#define SC_TOP                  0x02AC  // Dword offset AB
#define SC_BOTTOM               0x02B0  // Dword offset AC
#define SC_TOP_BOTTOM           0x02B4  // Dword offset AD
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
#define DP_BKGD_CLR             0x02C0  // Dword offset B0
#define DP_FRGD_CLR             0x02C4  // Dword offset B1
#define DP_WRITE_MASK           0x02C8  // Dword offset B2
#define DP_CHAIN_MASK           0x02CC  // Dword offset B3
#define DP_PIX_WIDTH            0x02D0  // Dword offset B4
#define DP_MIX                  0x02D4  // Dword offset B5
#define DP_SRC                  0x02D8  // Dword offset B6

#define CLR_CMP_CLR             0x0300  // Dword offset C0
#define CLR_CMP_MASK            0x0304  // Dword offset C1
#define CLR_CMP_CNTL            0x0308  // Dword offset C2

#define FIFO_STAT               0x0310  // Dword offset C4

#define CONTEXT_MASK            0x0320  // Dword offset C8
#define CONTEXT_LOAD_CNTL       0x032C  // Dword offset CB

#define GUI_TRAJ_CNTL           0x0330  // Dword offset CC
#define GUI_STAT                0x0338  // Dword offset CE
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

ATTR.C

```
/*============================================================================
  ATTR.C

  Commonly used MACH 64 draw functions such as filled rectangles and lines.

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  ============================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

/* ----------------------------------------------------------------------------
  GET_COLOR_CODE - return the color code for the given generic color.

  A "generic" color value is converted into a real color value for sample
  functions set_fg_color() and get_fg_color() to make color setting
  video mode independant. Typically, the returned value is used as the
  input value to set_fg_color() or set_bg_color().
---------------------------------------------------------------------------- */
unsigned long get_color_code(int generic_color)
{
    unsigned long color;

    color = BLACK;
    switch (generic_color)
    {
        case BLACK:
            color = BLACK;
            break;

        case DARKBLUE:
            if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
            {
                color = DARKBLUE8;
            }
            if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
            {
                color = DARKBLUE15;
            }
            if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
            {
                color = DARKBLUE16;
            }
```

```
        if (modeinfo.bpp == 24)
        {
            color = DARKBLUE24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = DARKBLUE32_RGBA;
        }
        break;

    case DARKGREEN:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = DARKGREEN8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = DARKGREEN15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = DARKGREEN16;
        }
        if (modeinfo.bpp == 24)
        {
            color = DARKGREEN24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = DARKGREEN32_RGBA;
        }
        break;

    case DARKCYAN:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = DARKCYAN8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = DARKCYAN15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = DARKCYAN16;
        }
        if (modeinfo.bpp == 24)
        {
            color = DARKCYAN24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            color = DARKCYAN32_RGBA;
        }
        break;

    case DARKRED:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = DARKRED8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = DARKRED15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = DARKRED16;
        }
        if (modeinfo.bpp == 24)
        {
            color = DARKRED24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = DARKRED32_RGBA;
        }
        break;

    case DARKMAGENTA:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = DARKMAGENTA8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = DARKMAGENTA15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = DARKMAGENTA16;
        }
        if (modeinfo.bpp == 24)
        {
            color = DARKMAGENTA24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = DARKMAGENTA32_RGBA;
        }
        break;

    case BROWN:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
```

```
        {
            color = BROWN8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = BROWN15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = BROWN16;
        }
        if (modeinfo.bpp == 24)
        {
            color = BROWN24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = BROWN32_RGBA;
        }
        break;

    case LIGHTGRAY:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = LIGHTGRAY8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = LIGHTGRAY15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = LIGHTGRAY16;
        }
        if (modeinfo.bpp == 24)
        {
            color = LIGHTGRAY24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = LIGHTGRAY32_RGBA;
        }
        break;

    case DARKGRAY:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = DARKGRAY8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = DARKGRAY15;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = DARKGRAY16;
        }
        if (modeinfo.bpp == 24)
        {
            color = DARKGRAY24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = DARKGRAY32_RGBA;
        }
        break;

    case LIGHTBLUE:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = LIGHTBLUE8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = LIGHTBLUE15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = LIGHTBLUE16;
        }
        if (modeinfo.bpp == 24)
        {
            color = LIGHTBLUE24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = LIGHTBLUE32_RGBA;
        }
        break;

    case LIGHTGREEN:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = LIGHTGREEN8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = LIGHTGREEN15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = LIGHTGREEN16;
        }
        if (modeinfo.bpp == 24)
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        {
            color = LIGHTGREEN24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = LIGHTGREEN32_RGBA;
        }
        break;

    case LIGHTCYAN:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = LIGHTCYAN8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = LIGHTCYAN15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = LIGHTCYAN16;
        }
        if (modeinfo.bpp == 24)
        {
            color = LIGHTCYAN24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = LIGHTCYAN32_RGBA;
        }
        break;

    case LIGHTRED:
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            color = LIGHTRED8;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
        {
            color = LIGHTRED15;
        }
        if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
        {
            color = LIGHTRED16;
        }
        if (modeinfo.bpp == 24)
        {
            color = LIGHTRED24_RGB;
        }
        if (modeinfo.bpp == 32)
        {
            color = LIGHTRED32_RGBA;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    }
    break;

case LIGHTMAGENTA:
    if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
    {
        color = LIGHTMAGENTA8;
    }
    if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
    {
        color = LIGHTMAGENTA15;
    }
    if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
    {
        color = LIGHTMAGENTA16;
    }
    if (modeinfo.bpp == 24)
    {
        color = LIGHTMAGENTA24_RGB;
    }
    if (modeinfo.bpp == 32)
    {
        color = LIGHTMAGENTA32_RGBA;
    }
    break;

case YELLOW:
    if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
    {
        color = YELLOW8;
    }
    if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
    {
        color = YELLOW15;
    }
    if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
    {
        color = YELLOW16;
    }
    if (modeinfo.bpp == 24)
    {
        color = YELLOW24_RGB;
    }
    if (modeinfo.bpp == 32)
    {
        color = YELLOW32_RGBA;
    }
    break;

case WHITE:
    if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
    {
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                color = WHITE8;
            }
            if ((modeinfo.bpp == 16) && (modeinfo.depth == 555))
            {
                color = WHITE15;
            }
            if ((modeinfo.bpp == 16) && (modeinfo.depth == 565))
            {
                color = WHITE16;
            }
            if (modeinfo.bpp == 24)
            {
                color = WHITE24_RGB;
            }
            if (modeinfo.bpp == 32)
            {
                color = WHITE32_RGBA;
            }
            break;
    }

    return (color);
}

/* ------------------------------------------------------------------------
  SET_FG_COLOR - set foreground color
------------------------------------------------------------------------ */
void set_fg_color(unsigned long color)
{
    wait_for_fifo(1);
    regw(DP_FRGD_CLR, color);
}

/* ------------------------------------------------------------------------
  GET_FG_COLOR - get foreground color
------------------------------------------------------------------------ */
unsigned long get_fg_color(void)
{
    wait_for_idle();  // insure engine is idle before reading a GUI register
    return (regr(DP_FRGD_CLR));
}

/* ------------------------------------------------------------------------
  SET_BG_COLOR - set background color
------------------------------------------------------------------------ */
void set_bg_color(unsigned long color)
{
    wait_for_fifo(1);
    regw(DP_BKGD_CLR, color);
}

/* ------------------------------------------------------------------------
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
  GET_BG_COLOR - get background color
------------------------------------------------------------------------ */
unsigned long get_bg_color(void)
{
    wait_for_idle();  // insure engine is idle before reading a GUI register
    return (regr(DP_BKGD_CLR));
}

/* ------------------------------------------------------------------------
  SET_FG_MIX - set the current foreground mix
------------------------------------------------------------------------ */
void set_fg_mix(int mix)
{
    unsigned long temp;

    wait_for_idle();  // insure engine is idle before reading a GUI register
    temp = (unsigned long)(mix);
    temp = temp << 16;
    temp = (unsigned long)(temp | (regr(DP_MIX) & 0xffff));
    regw(DP_MIX, temp);
}

/* ------------------------------------------------------------------------
  GET_FG_MIX - get the current foreground mix
------------------------------------------------------------------------ */
int get_fg_mix(void)
{
    wait_for_idle();  // insure engine is idle before reading a GUI register
    return ((int)(regr(DP_MIX) >> 16));
}

/* ------------------------------------------------------------------------
  SET_BG_MIX - set the current background mix
------------------------------------------------------------------------ */
void set_bg_mix(int mix)
{
    unsigned long temp;

    wait_for_idle();  // insure engine is idle before reading a GUI register
    temp = regr(DP_MIX) & 0xffff0000;
    regw(DP_MIX, (unsigned long)(temp | mix));
}

/* ------------------------------------------------------------------------
  GET_BG_MIX - get the current background mix
------------------------------------------------------------------------ */
int get_bg_mix(void)
{
    wait_for_idle();  // insure engine is idle before reading a GUI register
    return ((int)(regr(DP_MIX) & 0xffff));
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
/* ------------------------------------------------------------------------
  get_primary_color - separate the primary color component from a given
                      color (i.e. separate the RED, GREEN, or BLUE
                      component value

  This function is only useful for non-palettized modes such as 15, 16, 24,
  and 32 bpp color depths. If this function is called for 4 or 8 bpp modes,
  the index value will be returned without modification.
------------------------------------------------------------------------ */
unsigned long get_primary_color(PRIMARYCOLOR primarycolor,
                                unsigned long color)
{
    unsigned long primary;

    switch (modeinfo.bpp)
            {
        case 4:
        case 8:
            primary = color;
            break;

        case 16:
            if (modeinfo.depth == 555)
            {
                switch (primarycolor)
                {
                    case RED:   primary = (color >> 10) & 0x1f; break;
                    case GREEN: primary = (color >> 5) & 0x1f;  break;
                    case BLUE:  primary = color & 0x1f;         break;
                }
            }
            else   // 565 weight
            {
                switch (primarycolor)
                {
                    case RED:   primary = (color >> 11) & 0x1f; break;
                    case GREEN: primary = (color >> 5) & 0x3f;  break;
                    case BLUE:  primary = color & 0x1f;         break;
                }
            }
            break;

        case 24:
            switch (primarycolor)
            {
                case RED:   primary = (color >> 16) & 0xff; break;
                case GREEN: primary = (color >> 8) & 0xff;  break;
                case BLUE:  primary = color & 0xff;         break;
                }
            break;

        case 32:
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            switch (primarycolor)
            {
                case RED:   primary = (color >> 24) & 0xff; break;
                case GREEN: primary = (color >> 16) & 0xff; break;
                case BLUE:  primary = (color >> 8) & 0xff;  break;
                }
            break;
    }

    return (primary);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

HWCURSOR.C

```
/*============================================================================
  HWCURSOR.C

  Functions to define, enable, disable, and position the Mach64 hardware
  cursor.

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  ============================================================================*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

/* ----------------------------------------------------------------------------
  SET_HWCURSOR - define hardware cursor bitmap

  This function sets up the hardware cursor data region according to the
  given bitmap data. The data region is located at (0, y).

  The hardware cursor be vary in size from 1x1 to 64x64. The expected bitmap
  format is LSB to MSB. The LSB will be drawn first in a left to right
  direction. Since the cursor position is NOT set in this routine, the
  position should be set (set_hwcursor_pos) prior to enabling the cursor.
---------------------------------------------------------------------------- */
void set_hwcursor(int y, int width, int height,
                  unsigned long color0, unsigned long color1,
                  unsigned int *bitmap)
{
    unsigned long cur_offset, cur_size, cur_pitch;
    unsigned long temp1, temp2, temp3, temp4, temp5, temp6;
    unsigned long red1, green1, blue1, red2, green2, blue2;
    unsigned long rshift, gshift, bshift;
    unsigned long bitdata;
    int i, index, dataindex, start, widthwords;
    PALETTE entry;

    // Check that cursor size is within limits
    if ((width < 1) || (width > 64)) return;
    if ((height < 1) || (height > 64)) return;

    // set cursor dimensions
    cursordata.y = y;
    cursordata.width = width;
    cursordata.height = height;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    cursordata.color0 = color0;
    cursordata.color1 = color1;

    // set hwcursor bitmap to transparent
    for (index = 0; index < (HWCURHEIGHT * HWCURWIDTH); index++)
                {
        cursordata.bitmap[index] = 0xaaaa;
                }

    // load user hwcursor data into bitmap
    dataindex = 0;
    widthwords = width / 8;
    if (width > widthwords * 8)
    {
        widthwords++;
    }
    start = HWCURWIDTH - widthwords;
   for (index = start; index < (HWCURWIDTH * height); index = index + HWCURWIDTH)
                {
        i = 0;
        do
        {
            cursordata.bitmap[index + i] = *(bitmap + dataindex);
            if (width < 8)
            {
               cursordata.bitmap[index + i] = cursordata.bitmap[index + i] <<
((8 - width) * 2);
            }
            dataindex++;
            i++;
        } while (i < widthwords);
    }

    // calculate offset in bytes
    cur_offset = (unsigned long)(modeinfo.pitch);
    if (modeinfo.bpp == 4)
    {
        cur_offset = (unsigned long)(cur_offset / 2);
    }
    else
    {
        cur_offset = (unsigned long)(cur_offset * (modeinfo.bpp / 8));
    }
    cur_offset = (unsigned long)(cur_offset * y);

    // calculate cursor pitch (assuming 16 bpp)
    cur_pitch = HWCURWIDTH / 8;
    cur_pitch = cur_pitch << 22;

    // convert byte offset to qword offset and limit to 20 bits (used DWORD
    //   format used in DST_OFF_PITCH
    cur_offset = (cur_offset / 8) & 0x000fffff;
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// Use 16 bpp to setup hardware cursor bitmap

// save vital registers
wait_for_idle();
temp1 = regr(DP_PIX_WIDTH);
temp2 = regr(DP_CHAIN_MASK);
temp3 = regr(DST_OFF_PITCH);
temp4 = regr(DP_SRC);
temp5 = regr(DP_MIX);
temp6 = regr(DST_CNTL);

// load bitmap data to hardware cursor bitmap data area
regw(DP_PIX_WIDTH, (temp1 & 0xffc00000) | BYTE_ORDER_LSB_TO_MSB |
                   HOST_16BPP | SRC_16BPP | DST_16BPP);
regw(DP_CHAIN_MASK, 0x0410);  // chain mask for 16 bpp
regw(DST_OFF_PITCH, cur_pitch | cur_offset);
regw(DP_SRC, FRGD_SRC_HOST);
regw(DP_MIX, FRGD_MIX_S | BKGD_MIX_D);
regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);
regw(DST_X, 0);
regw(DST_Y, 0);
regw(DST_HEIGHT, HWCURHEIGHT);
regw(DST_WIDTH, HWCURWIDTH);
for (index = 0; index < (HWCURHEIGHT * HWCURWIDTH * 2); index = index + 2)
{
    wait_for_fifo(1);
    bitdata = (unsigned long)(cursordata.bitmap[index+1]);
    bitdata = (bitdata << 16) | (cursordata.bitmap[index]);
    regw(HOST_DATA0, bitdata);
}
wait_for_idle();

// set cursor size offsets
cur_size = (unsigned long)(64 - height);
cur_size = (unsigned long)((cur_size << 16) | (64 - width));
regw(CUR_HORZ_VERT_OFF, cur_size);

// set cursor colors
if (modeinfo.bpp > 8)
{
    // for 15, 16, 24, 32 color modes
    switch(modeinfo.bpp)
    {
        case 16:
            if (modeinfo.depth == 555)
            {
                rshift = 3;
                gshift = 3;
                bshift = 3;
            }
            else  // 565 weight
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            {
                rshift = 3;
                gshift = 2;
                bshift = 3;
            }
            break;

        case 24:
        case 32:
            rshift = 0;
            gshift = 0;
            bshift = 0;
            break;
    }

    // cursor color 0
    red1 = get_primary_color(RED, color0) << rshift;
    green1 = get_primary_color(GREEN, color0) << gshift;
    blue1 = get_primary_color(BLUE, color0) << bshift;

    // cursor color 1
    red2 = get_primary_color(RED, color1) << rshift;
    green2 = get_primary_color(GREEN, color1) << gshift;
    blue2 = get_primary_color(BLUE, color1) << bshift;

    if (querydata.dac_type != DAC_ATI68860)
    {
        // standard setup for other dacs
        regw(CUR_CLR0, (red1 << 24) | (green1 << 16) | (blue1 << 8));
        regw(CUR_CLR1, (red2 << 24) | (green2 << 16) | (blue2 << 8));
    }
}
else
{
    // for 4, 8 bpp color modes
    if (querydata.dac_type == DAC_ATI68860)
    {
        entry = get_palette((int)(color0 & 0xff));
        red1 = (unsigned long)(entry.red);
        green1 = (unsigned long)(entry.green);
        blue1 = (unsigned long)(entry.blue);
        entry = get_palette((int)(color1 & 0xff));
        red2 = (unsigned long)(entry.red);
        green2 = (unsigned long)(entry.green);
        blue2 = (unsigned long)(entry.blue);
    }
    else
    {
        regw(CUR_CLR0, color0 & 0xff);
        regw(CUR_CLR1, color1 & 0xff);
    }
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    if (querydata.dac_type == DAC_ATI68860)
    {
        // special setup for ATI68860/880 dac for cursor colors
        outp(ioDAC_CNTL, 1);
        outp(ioDAC_REGS, 0);

        outp(ioDAC_REGS+1, (int)red1);
        outp(ioDAC_REGS+1, (int)green1);
        outp(ioDAC_REGS+1, (int)blue1);

        outp(ioDAC_REGS+1, (int)red2);
        outp(ioDAC_REGS+1, (int)green2);
        outp(ioDAC_REGS+1, (int)blue2);

        outp(ioDAC_CNTL, 0);
    }

    // set offset to cursor data region
    regw(CUR_OFFSET, cur_offset);

    // restore vital registers
    regw(DP_PIX_WIDTH, temp1);
    regw(DP_CHAIN_MASK, temp2);
    regw(DST_OFF_PITCH, temp3);
    regw(DP_SRC, temp4);
    regw(DP_MIX, temp5);
    regw(DST_CNTL, temp6);
}

/* ------------------------------------------------------------------------
  ENABLE_HWCURSOR - turn on the hardware cursor
------------------------------------------------------------------------ */
void enable_hwcursor(void)
{
    // enable hardware cursor
    outpw(ioGEN_TEST_CNTL, inpw(ioGEN_TEST_CNTL) | 0x80);
}

/* ------------------------------------------------------------------------
  DISABLE_HWCURSOR - turn off the hardware cursor
------------------------------------------------------------------------ */
void disable_hwcursor(void)
{
    // disable hardware cursor
    outpw(ioGEN_TEST_CNTL, inpw(ioGEN_TEST_CNTL) & 0xff7f);
}

/* ------------------------------------------------------------------------
  SET_HWCURSOR_POS - set the hardware cursor position
------------------------------------------------------------------------ */
void set_hwcursor_pos(int x, int y)
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
{
    unsigned long cur_pos;

    // check for coordinate violations
    if (x < 0) x = 0;
    if (y < 0) y = 0;

    // set cursor position
    cur_pos = (unsigned long)y;
    cur_pos = (unsigned long)((cur_pos << 16) | x);
    regw(CUR_HORZ_VERT_POSN, cur_pos);
}

/* ------------------------------------------------------------------------
  get_hwcursor_pos - get the hardware cursor position
------------------------------------------------------------------------ */
void get_hwcursor_pos(POINT *position)
{
    unsigned long cur_pos;

    // get cursor position
    cur_pos = regr(CUR_HORZ_VERT_POSN);

    position->x = (int)(cur_pos & 0xffff);
    position->y = (int)((cur_pos >> 16) & 0xffff);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

GLOB.C

```
/*=========================================================================
  GLOB.C

  Global structures for sample code.

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"


/* Global structure declarations */

MODECFG modeinfo;
HWCURSOR cursordata;
QUERY_STRUCTURE querydata;
int mem_cntl;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

SAMPLE.H

```
/*===========================================================================
  SAMPLE.H

  C include file for sample functions

  Copyright (c) 1993-94 ATI Technologies Inc.  All rights reserved
 ===========================================================================*/

/* Function constants */

// BIOS interface constants
#define MODE_640x480     0x12
#define MODE_800x600     0x6a
#define MODE_1024x768    0x55
#define MODE_1280x1024   0x83

#define PITCH_1024       0
#define PITCH_NOCHANGE   1
#define PITCH_XRES       2

#define COLOR_DEPTH_4    1
#define COLOR_DEPTH_8    2
#define COLOR_DEPTH_15   3
#define COLOR_DEPTH_16   4
#define COLOR_DEPTH_24   5
#define COLOR_DEPTH_32   6

#define ACCELERATOR_MODE 1
#define VGA_MODE         0

#define NO_ERROR         0
#define YES_ERROR        1
#define NOT_SUPPORTED    2

// Detection contants
#define NO_MACH64        0
#define YES_MACH64       1

// Query constants
#define HEADER_ONLY      0
#define HEADER_AND_MODE  1

// Other constants
#define MEM_WRITE        1
#define MEM_READ         0

// Waitforidle() and Waitforfifo() errors and constants
#define BAD_IDLE         0
#define BAD_FIFO         1
```

```
#define DOS_TICK_ADDRESS 0x0000046c
#define IDLE_TIMEOUT     50
#define FIFO_TIMEOUT     50

// VGA aperture address constants
#define VGA_REGISTER_BASE  0xb000fc00
#define LOW_APERTURE_BASE  0xa0000000
#define HIGH_APERTURE_BASE 0xa8000000

// Mix control values for setfgmix() and setbgmix() functions
#define NOT_D_MIX           0
#define ZERO_MIX            1
#define ONE_MIX             2
#define D_MIX               3
#define NOT_S_MIX           4
#define D_XOR_S_MIX         5
#define NOT_D_XOR_S_MIX     6
#define S_MIX               7
#define NOT_D_OR_NOT_S_MIX  8
#define D_OR_NOT_S_MIX      9
#define NOT_D_OR_S_MIX      10
#define D_OR_S_MIX          11
#define D_AND_S_MIX         12
#define NOT_D_AND_S_MIX     13
#define D_AND_NOT_S_MIX     14
#define NOT_D_AND_NOT_S_MIX 15
#define D_PLUS_S_DIV2_MIX   23

// Status codes used in MODECFG structure */
#define LINEAR_APERTURE_ENABLED   1
#define LINEAR_APERTURE_DISABLED  0
#define VGA_APERTURE_ENABLED      1
#define VGA_APERTURE_DISABLED     0

// Generic color definitions for use in getcolorcode() function
#define BLACK               0
#define DARKBLUE            1
#define DARKGREEN           2
#define DARKCYAN            3
#define DARKRED             4
#define DARKMAGENTA         5
#define BROWN               6
#define LIGHTGRAY           7
#define DARKGRAY            8
#define LIGHTBLUE           9
#define LIGHTGREEN          10
#define LIGHTCYAN           11
#define LIGHTRED            12
#define LIGHTMAGENTA        13
#define YELLOW              14
#define WHITE               15
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// Color codes for 4 & 8 bpp modes - palette entry based
#define DARKBLUE8           0x00000001
#define DARKGREEN8          0x00000002
#define DARKCYAN8           0x00000003
#define DARKRED8            0x00000004
#define DARKMAGENTA8        0x00000005
#define BROWN8              0x00000006
#define LIGHTGRAY8          0x00000007
#define DARKGRAY8           0x00000008
#define LIGHTBLUE8          0x00000009
#define LIGHTGREEN8         0x0000000a
#define LIGHTCYAN8          0x0000000b
#define LIGHTRED8           0x0000000c
#define LIGHTMAGENTA8       0x0000000d
#define YELLOW8             0x0000000e
#define WHITE8              0x0000000f
#define HIWHITE8            0x000000ff

// Color codes for 15 bpp (16 bpp 555 color weight) modes - direct mapped
#define DARKBLUE15          0x00000014
#define DARKGREEN15         0x00000280
#define DARKCYAN15          0x00000294
#define DARKRED15           0x00005000
#define DARKMAGENTA15       0x00005014
#define BROWN15             0x00005280
#define LIGHTGRAY15         0x00005294
#define DARKGRAY15          0x0000294a
#define LIGHTBLUE15         0x0000001f
#define LIGHTGREEN15        0x000003e0
#define LIGHTCYAN15         0x000003ff
#define LIGHTRED15          0x00007c00
#define LIGHTMAGENTA15      0x00007c1f
#define YELLOW15            0x00007fe0
#define WHITE15             0x0000ffff

// Color codes for 16 bpp (16 bpp 565 color weight) modes - direct mapped
#define DARKBLUE16          0x00000014
#define DARKGREEN16         0x000004c0
#define DARKCYAN16          0x000004d4
#define DARKRED16           0x0000a000
#define DARKMAGENTA16       0x0000a014
#define BROWN16             0x0000a4c0
#define LIGHTGRAY16         0x0000a4d4
#define DARKGRAY16          0x0000528a
#define LIGHTBLUE16         0x0000001f
#define LIGHTGREEN16        0x000007e0
#define LIGHTCYAN16         0x000007ff
#define LIGHTRED16          0x0000f800
#define LIGHTMAGENTA16      0x0000f81f
#define YELLOW16            0x0000ffe0
#define WHITE16             0x0000ffff
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// Color codes for 24 bpp RGB color weight modes - direct mapped
#define DARKBLUE24_RGB      0x0000009e
#define DARKGREEN24_RGB     0x00009e00
#define DARKCYAN24_RGB      0x00009e9e
#define DARKRED24_RGB       0x009e0000
#define DARKMAGENTA24_RGB   0x009e009e
#define BROWN24_RGB         0x009e9e00
#define LIGHTGRAY24_RGB     0x009e9e9e
#define DARKGRAY24_RGB      0x00555555
#define LIGHTBLUE24_RGB     0x000000ff
#define LIGHTGREEN24_RGB    0x0000ff00
#define LIGHTCYAN24_RGB     0x0000ffff
#define LIGHTRED24_RGB      0x00ff0000
#define LIGHTMAGENTA24_RGB  0x00ff00ff
#define YELLOW24_RGB        0x00ffff00
#define WHITE24_RGB         0x00ffffff

// Color codes for 32 bpp RGBA color weight modes - direct mapped
#define DARKBLUE32_RGBA     0x00009e00
#define DARKGREEN32_RGBA    0x009e0000
#define DARKCYAN32_RGBA     0x009e9e00
#define DARKRED32_RGBA      0x9e000000
#define DARKMAGENTA32_RGBA  0x9e009e00
#define BROWN32_RGBA        0x9e9e0000
#define LIGHTGRAY32_RGBA    0x9e9e9e00
#define DARKGRAY32_RGBA     0x55555500
#define LIGHTBLUE32_RGBA    0x0000ff00
#define LIGHTGREEN32_RGBA   0x00ff0000
#define LIGHTCYAN32_RGBA    0x00ffff00
#define LIGHTRED32_RGBA     0xff000000
#define LIGHTMAGENTA32_RGBA 0xff00ff00
#define YELLOW32_RGBA       0xffff0000
#define WHITE32_RGBA        0xffffff00

// Macro for 24 bpp alignment - x is expressed in 24 bpp
#define GET24BPPROTATION(x)  (unsigned long)(((x * 3) / 4) % 6)


/* Engine bit constants - these are typically ORed together */

// GEN_TEST_CNTL register constants
#define HWCURSOR_ENABLE         0x80
#define GUI_ENGINE_ENABLE       0x100

// CONFIG_CNTL register constants
#define APERTURE_4M_ENABLE      1
#define APERTURE_8M_ENABLE      2
#define VGA_APERTURE_ENABLE     4

// DST_CNTL register constants
#define DST_X_RIGHT_TO_LEFT     0
#define DST_X_LEFT_TO_RIGHT     1
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
#define DST_Y_BOTTOM_TO_TOP      0
#define DST_Y_TOP_TO_BOTTOM      2
#define DST_X_MAJOR              0
#define DST_Y_MAJOR              4
#define DST_X_TILE               8
#define DST_Y_TILE               0x10
#define DST_LAST_PEL             0x20
#define DST_POLYGON_ENABLE       0x40
#define DST_24_ROTATION_ENABLE   0x80

// SRC_CNTL register constants
#define SRC_PATTERN_ENABLE       1
#define SRC_ROTATION_ENABLE      2
#define SRC_LINEAR_ENABLE        4
#define SRC_BYTE_ALIGN           8
#define SRC_LINE_X_RIGHT_TO_LEFT 0
#define SRC_LINE_X_LEFT_TO_RIGHT 0x10

// HOST_CNTL register constants
#define HOST_BYTE_ALIGN          1

// PAT_CNTL register constants
#define PAT_MONO_8x8_ENABLE      1
#define PAT_CLR_4x2_ENABLE       2
#define PAT_CLR_8x1_ENABLE       4

// DP_PIX_WIDTH register constants
#define DST_1BPP                 0
#define DST_4BPP                 1
#define DST_8BPP                 2
#define DST_15BPP                3
#define DST_16BPP                4
#define DST_32BPP                6
#define SRC_1BPP                 0
#define SRC_4BPP                 0x100
#define SRC_8BPP                 0x200
#define SRC_15BPP                0x300
#define SRC_16BPP                0x400
#define SRC_32BPP                0x600
#define HOST_1BPP                0
#define HOST_4BPP                0x10000
#define HOST_8BPP                0x20000
#define HOST_15BPP               0x30000
#define HOST_16BPP               0x40000
#define HOST_32BPP               0x60000
#define BYTE_ORDER_MSB_TO_LSB    0
#define BYTE_ORDER_LSB_TO_MSB    0x1000000

// DP_MIX register constants
#define BKGD_MIX_NOT_D              0
#define BKGD_MIX_ZERO               1
#define BKGD_MIX_ONE                2
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
#define BKGD_MIX_D                  3
#define BKGD_MIX_NOT_S              4
#define BKGD_MIX_D_XOR_S            5
#define BKGD_MIX_NOT_D_XOR_S        6
#define BKGD_MIX_S                  7
#define BKGD_MIX_NOT_D_OR_NOT_S     8
#define BKGD_MIX_D_OR_NOT_S         9
#define BKGD_MIX_NOT_D_OR_S         10
#define BKGD_MIX_D_OR_S             11
#define BKGD_MIX_D_AND_S            12
#define BKGD_MIX_NOT_D_AND_S        13
#define BKGD_MIX_D_AND_NOT_S        14
#define BKGD_MIX_NOT_D_AND_NOT_S    15
#define BKGD_MIX_D_PLUS_S_DIV2      0x17
#define FRGD_MIX_NOT_D              0
#define FRGD_MIX_ZERO               0x10000
#define FRGD_MIX_ONE                0x20000
#define FRGD_MIX_D                  0x30000
#define FRGD_MIX_NOT_S              0x40000
#define FRGD_MIX_D_XOR_S            0x50000
#define FRGD_MIX_NOT_D_XOR_S        0x60000
#define FRGD_MIX_S                  0x70000
#define FRGD_MIX_NOT_D_OR_NOT_S     0x80000
#define FRGD_MIX_D_OR_NOT_S         0x90000
#define FRGD_MIX_NOT_D_OR_S         0xa0000
#define FRGD_MIX_D_OR_S             0xb0000
#define FRGD_MIX_D_AND_S            0xc0000
#define FRGD_MIX_NOT_D_AND_S        0xd0000
#define FRGD_MIX_D_AND_NOT_S        0xe0000
#define FRGD_MIX_NOT_D_AND_NOT_S    0xf0000
#define FRGD_MIX_D_PLUS_S_DIV2      0x170000

// DP_SRC register constants
#define BKGD_SRC_BKGD_CLR           0
#define BKGD_SRC_FRGD_CLR           1
#define BKGD_SRC_HOST               2
#define BKGD_SRC_BLIT               3
#define BKGD_SRC_PATTERN            4
#define FRGD_SRC_BKGD_CLR           0
#define FRGD_SRC_FRGD_CLR           0x100
#define FRGD_SRC_HOST               0x200
#define FRGD_SRC_BLIT               0x300
#define FRGD_SRC_PATTERN            0x400
#define MONO_SRC_ONE                0
#define MONO_SRC_PATTERN            0x10000
#define MONO_SRC_HOST               0x20000
#define MONO_SRC_BLIT               0x30000

// CLR_CMP_CNTL register constants
#define COMPARE_FALSE               0
#define COMPARE_TRUE                1
#define COMPARE_NOT_EQUAL           4
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
#define COMPARE_EQUAL                5
#define COMPARE_DESTINATION          0
#define COMPARE_SOURCE               0x1000000

// FIFO_STAT register constants
#define FIFO_ERR                     0x80000000

// CONTEXT_LOAD_CNTL constants
#define CONTEXT_NO_LOAD              0
#define CONTEXT_LOAD                 0x10000
#define CONTEXT_LOAD_AND_DO_FILL     0x20000
#define CONTEXT_LOAD_AND_DO_LINE     0x30000
#define CONTEXT_EXECUTE              0
#define CONTEXT_CMD_DISABLE          0x80000000

// GUI_STAT register constants
#define ENGINE_IDLE                  0
#define ENGINE_BUSY                  1
#define SCISSOR_LEFT_FLAG            0x10
#define SCISSOR_RIGHT_FLAG           0x20
#define SCISSOR_TOP_FLAG             0x40
#define SCISSOR_BOTTOM_FLAG          0x80

// Bus types from CONFIG_STAT0 bits 2:0
#define BUS_ISA                      0
#define BUS_EISA                     1
#define BUS_VLB                      6
#define BUS_PCI                      7

// Dac types from CONFIG_STAT0 bits 11:9
#define DAC_TVP3020                  1
#define DAC_ATI68875                 2
#define DAC_BT476                    3
#define DAC_BT481                    4
#define DAC_ATI68860                 5
#define DAC_STG1700                  6
#define DAC_SC15021                  7

// Vga type from CONFIG_STAT0 bit 23
#define VGA_ENABLE                   1
#define VGA_DISABLE                  0

/* Hardware cursor bitmap dimensions */
#define HWCURWIDTH          8    // width of hwcursor in words
#define HWCURHEIGHT         64   // height of hwcursor in scan lines


/* Ensure that all structures are BYTE aligned -- same as /Zp1 CL option */
#pragma pack(1)

/* Global user structures */
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
typedef enum {BLUE, GREEN, RED} PRIMARYCOLOR;

typedef struct
{
    int y;
    int width;
    int height;
    unsigned long color0;
    unsigned long color1;
    unsigned int bitmap[HWCURHEIGHT * HWCURWIDTH];
} HWCURSOR;

typedef struct
{
    int red;
    int green;
    int blue;
} PALETTE;

typedef struct
{
    int x, y;
} POINT;

typedef struct
{
    int left, top, right, bottom;
} SCISSOR;

typedef struct
{
    int xres;
    int yres;
    int bpp;                          /* 4, 8, 16, 24, 32 */
    int depth;                        /* 555 for 15 bpp, 565 for 16 bpp */
    int pitch;
   int vga_aperture_status;      /* VGA_APERTURE_ENABLE, VGA_APERTURE_DISABLE */
    int linear_aperture_status;     /* APERTURE_4M_ENABLE, APERTURE_8M_ENABLE */
    int linear_aperture_size;       /* APERTURE_4M_ENABLE, APERTURE_8M_ENABLE */
    unsigned long aperture_address;       /* linear aperture address */
    unsigned long vga_memreg_offset;
    unsigned long linear_memreg_offset;
} MODECFG;

typedef struct                        /* See MACH 64 BIOS extension info */
{
    unsigned int  size;
    unsigned char revision;
    unsigned char mode_tables;
    unsigned int  mode_table_offset;
    unsigned char mode_table_size;
    unsigned char vga_type;           /* See VGA types */
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    unsigned int  asic_id;
    unsigned char vga_boundary;
    unsigned char memory_size;
    unsigned char dac_type;          /* See DAC types */
    unsigned char memory_type;
    unsigned char bus_type;          /* See BUS types */
    unsigned char monitor_cntl;
    unsigned int  aperture_addr;
    unsigned char aperture_cfg;
    unsigned char color_depth_support;
    unsigned char ramdac_feature_support;
    unsigned char reserved1;
    unsigned int  reserved2;
    unsigned int  io_base;
    unsigned char reserved3[6];
} QUERY_STRUCTURE;


/* Resume word alignment */
#pragma pack()

/* Function declarations */

// Routines from ATTR.C
unsigned long get_color_code(int generic_color);
void set_fg_color(unsigned long color);
unsigned long get_fg_color(void);
void set_bg_color(unsigned long color);
unsigned long get_bg_color(void);
void set_fg_mix(int mix);
int  get_fg_mix(void);
void set_bg_mix(int mix);
int  get_bg_mix(void);
unsigned long get_primary_color(PRIMARYCOLOR primarycolor,
                                unsigned long color);

// Routines from DRAW.C
void blit(int x1, int y1, int x2, int y2, int width, int height);
void clear_screen(int x, int y, int width, int height);
void draw_rectangle(int x, int y, int width, int height);
void draw_line(int x1, int y1, int x2, int y2);
void draw_line24(int x1, int y1, int x2, int y2);

// Routines from HWCURSOR.C
void set_hwcursor(int y, int width, int height,
                  unsigned long color0, unsigned long color1,
                  unsigned int *bitmap);
void enable_hwcursor(void);
void disable_hwcursor(void);
void set_hwcursor_pos(int x, int y);
void get_hwcursor_pos(POINT *position);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// Routines from INIT.C
int  detect_mach64(void);
void init_4bpp(void);
void init_8bpp(void);
void init_15bpp(void);
void init_16bpp(void);
void init_24bpp(void);
void init_32bpp(void);
void reset_engine(void);
void init_engine(void);
int  open_mode(int mode_code, int pitch_code, int color_code);
void close_mode(void);
void enable_vga_aperture(void);
void disable_vga_aperture(void);
void enable_linear_aperture(int aperture_size);
void disable_linear_aperture(void);
void init_aperture(void);
void update_aperture_status(void);
int  query_hardware(void);
void set_packed_pixel(void);

// Routines from MEMREG.C
void vga_regw(unsigned int regindex, unsigned long regdata);
unsigned long vga_regr(unsigned int regindex);
void app_regw(unsigned int regindex, unsigned long regdata);
unsigned long app_regr(unsigned int regindex);
void regw(unsigned int regindex, unsigned long regdata);
unsigned long regr(unsigned int regindex);
void iow(unsigned int ioaddr, unsigned long iodata);
unsigned long ior(unsigned int ioaddr);

// Routines from PALETTE.C
void set_palette(int index, PALETTE entry);
PALETTE get_palette(int index);
void init_palette(void);

// Routines from WAIT.C
void wait_for_idle(void);
void wait_for_fifo(int entries);
void terminate(int idle_problem);

// Routines from MOVEMEM.ASM
int movemem(void far *ptr, unsigned long phyaddr, unsigned int nwords, int
direction);

// Routines from ROMCALLS.ASM
int is_ati_rom(void);
int load_mode_parms(int resolution_code, int pitch_code, int color_depth_code);
int set_display_mode(int display_mode);
int load_and_set_mode(int resolution_code, int pitch_code, int
color_depth_code);
int get_query_size(int info_type);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
int fill_query_structure(int info_type, unsigned char far *ptr);

/* External variable and array declarations */

extern MODECFG modeinfo;
extern HWCURSOR cursordata;
extern QUERY_STRUCTURE querydata;
extern int mem_cntl;
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

DRAW.C

```
/*=========================================================================
  DRAW.C

  Sample code draw functions such as rectangles and lines.

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

/* -------------------------------------------------------------------------
  DRAW_RECTANGLE - draw a filled rectangle

  A filled rectangle is drawn at (x, y) of size (width x height) using the
  current engine settings. For 24 bpp modes, the engine is actually in 8
  bpp mode and the CRTC is in 24 bpp. For this reason, all horizontal
  parameters must be multiplied by 3. Also, the 24 bpp alignment must
  be determine for the engine to draw pixels in the correct color. Note that
  for 24 bpp modes, the input parameters for this routine are in 24 bpp.
  The rectangle source is determined by the current setting of the DP_SRC
  register.
------------------------------------------------------------------------- */
void draw_rectangle(int x, int y, int width, int height)
{
    unsigned long temp, rotation;

    wait_for_idle();      // wait for idle before reading GUI registers

    // save used registers
    temp = regr(DST_CNTL);

    if (modeinfo.bpp == 24)
    {
        // adjust horizontal parameters
        x = x * 3;
        width = width * 3;

        // set 24 bpp alignment while maintaining direction bits
        rotation = GET24BPPROTATION(x);
        regw(DST_CNTL, (temp & 0xdf) |
                       DST_24_ROTATION_ENABLE |
                       (rotation << 8));
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    }

    // perform rectangle fill
    regw(DST_X, (unsigned long)x);
    regw(DST_Y, (unsigned long)y);
    regw(DST_HEIGHT, (unsigned long)height);
    regw(DST_WIDTH, (unsigned long)width);

    // restore
    regw(DST_CNTL, temp);
}

/* ---------------------------------------------------------------------------
  CLEAR_SCREEN - clear a region of video memory

  A BLACK filled rectangle is drawn at (x, y) of size (width x height).
  The current engine mix setting will not affect the operation.
--------------------------------------------------------------------------- */
void clear_screen(int x, int y, int width, int height)
{
    unsigned long temp;

    wait_for_idle();      // wait for idle before reading GUI registers

    // save used registers
    temp = regr(DP_MIX);

    // perform clear screen
    regw(DP_MIX, FRGD_MIX_ZERO | BKGD_MIX_ZERO);
    draw_rectangle(x, y, width, height);

    // wait for clear screen operation to finish before exiting
    wait_for_idle();

    // restore DP_MIX register
    regw(DP_MIX, temp);
}

/* ---------------------------------------------------------------------------
  DRAW_LINE - draw a line from (x1, y1) to (x2, y2)

  The drawing of the last pixel in the line is determined by the current
  setting of the DST_CNTL register (LAST_PEL bit).
--------------------------------------------------------------------------- */
void draw_line(int x1, int y1, int x2, int y2)
{
    int dx, dy;
    int small, large;
    int x_dir, y_dir, y_major;
    unsigned long err, inc, dec, temp;

    /* call specific routine if mode is in 24 bpp */
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    if (modeinfo.bpp == 24)
    {
        draw_line24(x1, y1, x2, y2);
        return;
    }

    /* determine x & y deltas and x & y direction bits */
    if (x1 < x2)
    {
        dx = x2 - x1;
        x_dir = 1;
    }
    else
    {
        dx = x1 - x2;
        x_dir = 0;
    }

    if (y1 < y2)
    {
        dy = y2 - y1;
        y_dir = 2;
    }
    else
    {
        dy = y1 - y2;
        y_dir = 0;
    }

    /* determine x & y min and max values; also determine y major bit */
    if (dx < dy)
    {
        small = dx;
        large = dy;
        y_major = 4;
    }
    else
    {
        small = dy;
        large = dx;
        y_major = 0;
    }

    /* calculate bressenham parameters and draw line */
    err = (unsigned long)((2 * small) - large);
    inc = (unsigned long)(2 * small);
    dec = 0x3ffff - ((unsigned long)(2 * (large - small)));

    wait_for_idle();      // wait for idle before reading GUI registers

    // save used registers
    temp = regr(DST_CNTL);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    // draw bressenham line
    regw(DST_X, (unsigned long)x1);
    regw(DST_Y, (unsigned long)y1);

    // allow setting of last pel bit and polygon outline bit for line drawing

    regw(DST_CNTL, (temp & 0x60) | (unsigned long)(y_major | y_dir | x_dir));
    regw(DST_BRES_ERR, err);
    regw(DST_BRES_INC, inc);
    regw(DST_BRES_DEC, dec);
    regw(DST_BRES_LNTH, (unsigned long)(large + 1));

    // restore
    regw(DST_CNTL, temp);
}


/* ------------------------------------------------------------------------
  DRAW_LINE24 - draw a bressenham line from (x1, y1) to (x2, y2).

  Since the engine does not directly support 24 bpp modes, it is set
  to 8 bpp while the CRTC is set to 24 bpp display mode (RGB). For
  rectangle drawing, all X coordinates and widths must be converted
  to 8 bpp sizes. This is done by taking the 24 bpp value and
  multipling it by 3.
------------------------------------------------------------------------ */
void draw_line24(int x1, int y1, int x2, int y2)
{
    int x, y, xend, yend, dx, dy;
    int d, incr1, incr2, incr3;
    unsigned long rotation, temp1, temp2;

    // save register
    wait_for_idle();
    temp1 = regr(DST_CNTL);
    temp2 = 0xa3;

    // ------------------ Bressenham line routine ----------------------

    dx = abs(x2 - x1);
    dy = abs(y2 - y1);

    // check slope
    if (dy <= dx) // slope <= 1

    {
        if (x1 > x2)
        {
            x = x2;
            y = y2;
            xend = x1;
```

```
        dy = y1 - y2;
    }
    else
    {
        x = x1;
        y = y1;
        xend = x2;
        dy = y2 - y1;
    }

    d = (2 * dy) - dx;
    incr1 = 2 * dy;
    incr2 = 2 * (dy - dx);
    incr3 = 2 * (dy + dx);

    regw(DST_HEIGHT, 1);
    regw(DST_Y, y);

    do
    {
        wait_for_fifo(4);
        rotation = GET24BPPROTATION(x);
        regw(DST_CNTL, temp2 | (rotation << 8));
        regw(DST_X, x * 3);
        regw(DST_WIDTH, 3);

        x++;

        if (d >= 0)
        {
            if (dy <= 0)
            {
                d = d + incr1;
            }
            else
            {
                y++;
                regw(DST_Y, y);
                d = d + incr2;
            }
        }
        else
        {
            if (dy >= 0)
            {
                d = d + incr1;
            }
            else
            {
                y--;
                regw(DST_Y, y);
                d = d + incr3;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            }
        }
    } while (x <= xend);
}
else          // slope > 1
{
    if (y1 > y2)
    {
        y = y2;
        x = x2;
        yend = y1;
        dx = x1 - x2;
    }
    else
    {
        y = y1;
        x = x1;
        yend = y2;
        dx = x2 - x1;
    }

    d = (2 * dx) - dy;
    incr1 = 2 * dx;
    incr2 = 2 * (dx - dy);
    incr3 = 2 * (dx + dy);

    regw(DST_HEIGHT, 1);

    do
    {
        wait_for_fifo(3);
        rotation = GET24BPPROTATION(x);
        regw(DST_CNTL, temp2 | (rotation << 8));
        regw(DST_Y_X, ((unsigned long)(x * 3) << 16) | y);
        regw(DST_WIDTH, 3);

        y++;

        if (d >= 0)
        {
            if (dx <= 0)
            {
                d = d + incr1;
            }
            else
            {
                x++;
                d = d + incr2;
            }
        }
        else
        {
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                if (dx >= 0)
                {
                    d = d + incr1;
                }
                else
                {
                    x--;
                    d = d + incr3;
                }
            }
        } while (y <= yend);
    }

    // restore register
    wait_for_fifo(1);
    regw(DST_CNTL, temp1);
}

/* ------------------------------------------------------------------------
  BLIT - perform a screen to screen blit

  Copy the contents of screen memory at (x1, y1) of size (width x height) to
  (x2, y2) using the current engine settings. This is known as an unbounded
  Y source trajectory blit. For 24 bpp modes, the engine is in 8 bpp and
  the CRTC is in 24 bpp. For this reason, all horizontal parameters must be
  mulitplied by 3. The blit source is determined by the current setting
  of the DP_SRC register.
------------------------------------------------------------------------ */
void blit(int x1, int y1, int x2, int y2, int width, int height)
{
    unsigned long temp, rotation;

    wait_for_idle();      // wait for idle before reading GUI registers

    // save used registers
    temp = regr(DST_CNTL);

    if (modeinfo.bpp == 24)
    {
        // adjust horizontal parameters
        x1 = x1 * 3;
        x2 = x2 * 3;
        width = width * 3;

        // get 24 bpp alignment rotation for destination
        rotation = GET24BPPROTATION(x2);

        // set 24 bpp alignment while maintaining direction bits
        regw(DST_CNTL, (temp & 0xdf) |
                       DST_24_ROTATION_ENABLE |
                       (rotation << 8));
    }
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    // perform a blit
    regw(SRC_X, x1);
    regw(SRC_Y, y1);
    regw(SRC_HEIGHT1, height);
    regw(SRC_WIDTH1, width);

    regw(DST_X, x2);
    regw(DST_Y, y2);
    regw(DST_HEIGHT, height);
    regw(DST_WIDTH, width);

    // restore
    regw(DST_CNTL, temp);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

INIT.C

```
/*=========================================================================
  INIT.C

  MACH 64 functions to initialize and set a standard engine context.

  Copyright (c) 1993-94 ATI Technologies Inc. All rights reserved
  =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

/* -------------------------------------------------------------------------
  detect_mach64 - determine if a mach64 based video adapter is installed.

  This routine identifies if a mach64 based video adapter is installed. This
  is done by writing and reading the SCRATCH_REG0 or SCRATCH_REG1 io based
  registers. These registers must be saved and restored since the mach64
  BIOS uses their contents.

  Returns YES_MACH64 if detected, NO_MACH64 if not.
------------------------------------------------------------------------- */
int detect_mach64(void)
{
    unsigned long save_value;
    int result;

    // assume failure
    result = NO_MACH64;

    // check for ATI rom signature
    if (is_ati_rom() == 0)
    {
        return (NO_MACH64);
    }

    // save old value
    save_value = ior(ioSCRATCH_REG0);

    // test odd bits for readability
    iow(ioSCRATCH_REG0, 0x55555555);
    if (ior(ioSCRATCH_REG0) == 0x55555555)
    {
        // test even bits for readability
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
        iow(ioSCRATCH_REG0, 0xaaaaaaaa);
        if (ior(ioSCRATCH_REG0) == 0xaaaaaaaa)
        {
            result = YES_MACH64;
        }
    }

    // restore old value
    iow(ioSCRATCH_REG0, save_value);

    return (result);
}

/* ------------------------------------------------------------------------
  INIT_4BPP - set the MACH64 engine to a 4bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_4bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_4BPP | SRC_4BPP | DST_4BPP |
                       BYTE_ORDER_MSB_TO_LSB);
    regw(DP_CHAIN_MASK, 0x8888);
}

/* ------------------------------------------------------------------------
  INIT_8BPP - set the MACH64 engine to a 8bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_8bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_8BPP | SRC_8BPP | DST_8BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x8080);
}

/* ------------------------------------------------------------------------
  INIT_15BPP - set the MACH64 engine to a 15bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_15bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_15BPP | SRC_15BPP | DST_15BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x4210);
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
/* ------------------------------------------------------------------------
  INIT_16BPP - set the MACH64 engine to a 16bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_16bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_16BPP | SRC_16BPP | DST_16BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x0410);
}

/* ------------------------------------------------------------------------
  INIT_24BPP - set the MACH64 engine to a 24bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_24bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_8BPP | SRC_8BPP | DST_8BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x8080);
}

/* ------------------------------------------------------------------------
  INIT_32BPP - set the MACH64 engine to a 32bpp standard context.

  This routine is used in conjuction with INIT_ENGINE().
------------------------------------------------------------------------ */
void init_32bpp(void)
{
    wait_for_fifo(2);
    regw(DP_PIX_WIDTH, HOST_32BPP | SRC_32BPP | DST_32BPP |
                       BYTE_ORDER_LSB_TO_MSB);
    regw(DP_CHAIN_MASK, 0x8080);
}

/* ------------------------------------------------------------------------
  RESET_ENGINE - reset engine and clear any FIFO errors

  This function resets the GUI engine and clears any FIFO errors.
------------------------------------------------------------------------ */
void reset_engine(void)
{
    // reset engine
    iow(ioGEN_TEST_CNTL, 0);
    iow(ioGEN_TEST_CNTL, GUI_ENGINE_ENABLE);

    // Ensure engine is not locked up by clearing any FIFO errors
    iow(ioBUS_CNTL, (ior(ioBUS_CNTL) & 0xff00ffff) | 0x00ae0000);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
}

/* ---------------------------------------------------------------------------
  INIT_ENGINE - set the MACH64 engine to a standard context.

  This routine configures the MACH64 engine to a known typical context.
  The context consists of:

     -engine reset and enabling

     -fifo control including clearing fifo overflow errors

     -source and destination pitch

     -source and destination offset into video memory

     -color depth

     -host data control

     -pattern data control

     -line and rectangle control

     -color source, mix, and compare control

     -scissor settings

     -write mask

--------------------------------------------------------------------------- */
void init_engine(void)
{
    unsigned long pitch_value, xres, yres;

    // determine modal information from global mode structure
    xres = (unsigned long)(modeinfo.xres);
    yres = (unsigned long)(modeinfo.yres);
    pitch_value = (unsigned long)(modeinfo.pitch);
    if (modeinfo.bpp == 24)
    {
        // In 24 bpp, the engine is in 8 bpp - this requires that all
        // horizontal coordinates and widths must be adjusted
        pitch_value = pitch_value * 3;
    }

    // Reset engine and clear any errors
    reset_engine();

    // Ensure that vga page pointers are set to zero - the upper page
    // pointers are set to 1 to handle overflows in the lower page
    iow(ioMEM_VGA_WP_SEL, 0x00010000);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    iow(ioMEM_VGA_RP_SEL, 0x00010000);

    // setup standard engine context
    wait_for_fifo(14);

    regw(CONTEXT_MASK, 0xFFFFFFFF);

    regw(DST_OFF_PITCH, (pitch_value / 8) << 22);
    regw(DST_Y_X, 0);
    regw(DST_HEIGHT, 0);
    regw(DST_BRES_ERR, 0);
    regw(DST_BRES_INC, 0);
    regw(DST_BRES_DEC, 0);
    regw(DST_CNTL, DST_LAST_PEL | DST_Y_TOP_TO_BOTTOM | DST_X_LEFT_TO_RIGHT);

    regw(SRC_OFF_PITCH, (pitch_value / 8) << 22);
    regw(SRC_Y_X, 0);
    regw(SRC_HEIGHT1_WIDTH1, 0);
    regw(SRC_Y_X_START, 0);
    regw(SRC_HEIGHT2_WIDTH2, 0);
    regw(SRC_CNTL, SRC_LINE_X_LEFT_TO_RIGHT);

    wait_for_fifo(13);

    regw(HOST_CNTL, 0);

    regw(PAT_REG0, 0);
    regw(PAT_REG1, 0);
    regw(PAT_CNTL, 0);

    regw(SC_LEFT, 0);
    regw(SC_TOP, 0);
    regw(SC_BOTTOM, yres-1);
    regw(SC_RIGHT, pitch_value-1);

    regw(DP_BKGD_CLR, 0);
    regw(DP_FRGD_CLR, 0xFFFFFFFF);
    regw(DP_WRITE_MASK, 0xFFFFFFFF);
    regw(DP_MIX, FRGD_MIX_S | BKGD_MIX_D);
    regw(DP_SRC, FRGD_SRC_FRGD_CLR);

    wait_for_fifo(3);

    regw(CLR_CMP_CLR, 0);
    regw(CLR_CMP_MASK, 0xFFFFFFFF);
    regw(CLR_CMP_CNTL, 0);

    switch(modeinfo.bpp)
    {
        case 4 : init_4bpp();  break;
        case 8 : init_8bpp();  break;
        case 16:
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            if (modeinfo.depth == 555)
            {
                init_15bpp();   // 555 color weighting
            }
            else
            {
                init_16bpp();   // 565 color weighting
            }
            break;
        case 24: init_24bpp(); break;
        case 32: init_32bpp(); break;
    }

    wait_for_idle();        // insure engine is idle before leaving
}

/* ------------------------------------------------------------------------
  OPEN_MODE - set an accelerator mode for the Mach 64

  This function sets an accelerator mode, initializes a global mode
  structure, and determines the access method for the GUI memory mapped
  registers of the Mach 64. This function should be called before any other
  engine function since they depend on the global mode information structure
  that is filled by this function. It is assumed that a Mach 64 based video
  adapter has been detected before calling this function (detect_mach64()).

  Inputs :

    Mode code:  MODE_640x480,
                MODE_800x600,
                MODE_1024x768,
                MODE_1280x1024

    Pitch code: PITCH_1024,
                PITCH_XRES

    Color code: COLOR_DEPTH_4,     // palettized 4 bpp
                COLOR_DEPTH_8,     // palettized 8 bpp
                COLOR_DEPTH_15,    // direct color 15 bpp - 555 weighting
                COLOR_DEPTH_16,    // direct color 16 bpp - 565 weighting
                COLOR_DEPTH_24,    // direct color 24 bpp - RGB weighting
                COLOR_DEPTH_32     // direct color 32 bpp - RGBa weighting

  Returns:

    NO_ERROR       - the accelerator mode has been set successfully
    YES_ERROR      - an error has occurred
    NOT_SUPPORTED - the accelerator mode is not supported
------------------------------------------------------------------------ */
int open_mode(int mode_code, int pitch_code, int color_code)
{
    int retval;
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
// Setup apertures
init_aperture();
update_aperture_status();

// Insure that memory is shared between VGA and acelerator for full
// memory access
mem_cntl = inpw(ioMEM_CNTL+2);
outpw(ioMEM_CNTL+2, 0);

// attempt to set an accelerator mode
retval = load_and_set_mode(mode_code, pitch_code, color_code);
if (retval == NO_ERROR)
{
    // Fill modal information structure
    modeinfo.depth = 0;
    switch(mode_code)
    {
        case MODE_640x480:
            modeinfo.xres = 640;
            modeinfo.yres = 480;
            break;

        case MODE_800x600:
            modeinfo.xres = 800;
            modeinfo.yres = 600;
            break;

        case MODE_1024x768:
            modeinfo.xres = 1024;
            modeinfo.yres = 768;
            break;

        case MODE_1280x1024:
            modeinfo.xres = 1280;
            modeinfo.yres = 1024;
            break;
    }
    switch(pitch_code)
    {
        case PITCH_1024:
            modeinfo.pitch = 1024;
            break;

        case PITCH_XRES:
            modeinfo.pitch = modeinfo.xres;
            break;
    }
    switch(color_code)
    {
        case COLOR_DEPTH_4:
            modeinfo.bpp = 4;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                break;

            case COLOR_DEPTH_8:
                modeinfo.bpp = 8;
                break;

            case COLOR_DEPTH_15:
                modeinfo.bpp = 16;
                modeinfo.depth = 555;
                break;

            case COLOR_DEPTH_16:
                modeinfo.bpp = 16;
                modeinfo.depth = 565;
                break;

            case COLOR_DEPTH_24:
                modeinfo.bpp = 24;
                break;

            case COLOR_DEPTH_32:
                modeinfo.bpp = 32;
                break;
        }

        // Insure that engine is clear of any idle or fifo errors
        reset_engine();

        // Setup palette if 4 or 8 bpp mode
        if ((modeinfo.bpp == 4) || (modeinfo.bpp == 8))
        {
            init_palette();
        }
    }
    else
    {
        // Insure VGA apeture is disabled for proper operation of VGA
        // controller (in text mode) before leaving
        disable_vga_aperture();
    }

    return (retval);
}

/* ------------------------------------------------------------------------
  CLOSE_MODE - switch back to VGA mode from an accelerator mode

  This function closes a Mach 64 accelerator mode and switches the display
  back to VGA control.
------------------------------------------------------------------------ */
void close_mode(void)
{
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    union REGS regs;

    // Disable accelerator mode and go back to VGA mode
    set_display_mode(VGA_MODE);

    // Disable VGA aperture BEFORE reinitializing the VGA controller
    disable_vga_aperture();

    // Since the accelerator and VGA controllers share the same video memory,
    // it is necessary to reinitialize the VGA text mode before exiting. Text
    // mode 3 is used here.
    regs.x.ax = 3;
    int86(0x10, &regs, &regs);

    // restore MEM_CNTL register
    outpw(ioMEM_CNTL+2, mem_cntl);
}

/* ------------------------------------------------------------------------
  ENABLE_VGA_APERTURE - enable the vga aperture

  This function enables the VGA aperture and sets the modeinfo
  structure contents accordingly. The sample code functions will use the new
  settings.
------------------------------------------------------------------------ */
void enable_vga_aperture(void)
{
    outpw(ioCONFIG_CNTL, inpw(ioCONFIG_CNTL) | VGA_APERTURE_ENABLE);
    modeinfo.vga_aperture_status = VGA_APERTURE_ENABLED;
}

/* ------------------------------------------------------------------------
  DISABLE_VGA_APERTURE - disable the vga aperture

  This function disables the VGA aperture and sets the modeinfo
  structure contents accordingly. The sample code functions will use the new
  settings. It is important to note that if both the linear aperture and
  the vga aperture are disabled, the sample code functions will fail to
  operate correctly. It is the application's responsibility to insure that
  this does not occur.
------------------------------------------------------------------------ */
void disable_vga_aperture(void)
{
    outpw(ioCONFIG_CNTL, inpw(ioCONFIG_CNTL) & (~VGA_APERTURE_ENABLE));
    modeinfo.vga_aperture_status = VGA_APERTURE_DISABLED;
}

/* ------------------------------------------------------------------------
  ENABLE_LINEAR_APERTURE - enable the linear aperture

  This function enables the linear aperture for 4M or 8M. The linear aperture
  address is set by the ROM at boot time. The address is assumed to be
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
  correct. The sample code functions will use the new settings.
---------------------------------------------------------------------------- */
void enable_linear_aperture(int aperture_size)
{
    if (aperture_size == APERTURE_4M_ENABLE) // 4M size
    {
      outpw(ioCONFIG_CNTL, (inpw(ioCONFIG_CNTL) & 0xfffc) | APERTURE_4M_ENABLE);
        modeinfo.linear_memreg_offset = 0x3ffc00;
        modeinfo.linear_aperture_size = APERTURE_4M_ENABLE;
    }
    else                                        // 8M size
    {
      outpw(ioCONFIG_CNTL, (inpw(ioCONFIG_CNTL) & 0xfffc) | APERTURE_8M_ENABLE);
        modeinfo.linear_memreg_offset = 0x7ffc00;
        modeinfo.linear_aperture_size = APERTURE_8M_ENABLE;
    }
}

/* ---------------------------------------------------------------------------
  DISABLE_LINEAR_APERTURE - disable the linear aperture

  This function disables the linear aperture. It is important to note that
  if both the linear aperture and the vga aperture are disabled, the PGL
  functions will fail to operate correctly. It is the application's
  responsibility to insure that this does not occur.
---------------------------------------------------------------------------- */
void disable_linear_aperture(void)
{
    outpw(ioCONFIG_CNTL,
          inpw(ioCONFIG_CNTL) & (~(APERTURE_4M_ENABLE | APERTURE_8M_ENABLE)));
}

/* ---------------------------------------------------------------------------
  INIT_APERTURE - set the apertures according to Mach 64 hardware

  This function selects the aperture according to the installed hardware
  for performance and functionality. This function is called by open_mode()
  each time a new mode is set.
---------------------------------------------------------------------------- */
void init_aperture(void)
{
    // If the VGA controller is enabled, use VGA aperture. If it is disabled,
    // use the linear aperture.
    if (querydata.vga_type == VGA_ENABLE)
    {
        // Enable VGA aperture
        outpw(ioCONFIG_CNTL, inpw(ioCONFIG_CNTL) | VGA_APERTURE_ENABLE);
    }
    else
    {
        // Enable linear aperture
        outpw(ioCONFIG_CNTL, inpw(ioCONFIG_CNTL) | APERTURE_4M_ENABLE);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    }
}

/* ------------------------------------------------------------------------
  UPDATE_APERTURE_STATUS - update aperture status for PGL register access

  This function is always called when a new mode is set through open_mode().
  The aperture status information in the MODECFG structure is updated
  according the current aperture settings of the Mach 64 hardware. This
  function is also useful if the aperture settings change during an active
  session. The aperture status information is used by the sample functions to
  determine the method of access for the Mach 64 memory mapped registers.
------------------------------------------------------------------------ */
void update_aperture_status(void)
{
    int config1, config2;

    /* ---- determine status of VGA and linear apertures ---- */
    config1 = inpw(ioCONFIG_CNTL);
    config2 = inpw(ioCONFIG_CNTL+2);

    // determine addresses and offsets for linear aperture
    modeinfo.aperture_address = (unsigned long)(config1 & 0x3ff0);
    modeinfo.aperture_address = modeinfo.aperture_address << 18;
    if ((config1 & APERTURE_4M_ENABLE) == APERTURE_4M_ENABLE)
    {
        // 4M size
        modeinfo.linear_memreg_offset = 0x3ffc00;
        modeinfo.linear_aperture_size = APERTURE_4M_ENABLE;
    }
    else
    {
        // 8M size
        modeinfo.linear_memreg_offset = 0x7ffc00;
        modeinfo.linear_aperture_size = APERTURE_8M_ENABLE;
    }

    // determine linear aperture status and address
    if ((config1 & 3) != 0)
    {
        modeinfo.linear_aperture_status = LINEAR_APERTURE_ENABLED;
    }
    else
    {
        modeinfo.linear_aperture_status = LINEAR_APERTURE_DISABLED;
    }

    // determine addresses and offsets for vga aperture
    modeinfo.vga_memreg_offset = 0xb000fc00;    // segment:offset

    // determine vga aperture status
    if ((config1 & VGA_APERTURE_ENABLE) == VGA_APERTURE_ENABLE)
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    {
        modeinfo.vga_aperture_status = VGA_APERTURE_ENABLED;
    }
    else
    {
        modeinfo.vga_aperture_status = VGA_APERTURE_DISABLED;
    }
}

/* ------------------------------------------------------------------------
  QUERY_HARDWARE - query the Mach 64 hardware and fill a global query
                   structure

  This function calls the Mach 64 ROM to fill a global query structure that
  is used by other functions. It should be called before using any other
  function including init_aperture() and open_mode(). The return code should
  be checked before proceeding.

  Returns:

    NO_ERROR      - the query structure has been successfully filled
    YES_ERROR     - an error has occurred
------------------------------------------------------------------------ */
int query_hardware(void)
{
    int structure_size, error;
    unsigned char far *ptr;
    QUERY_STRUCTURE far *query_ptr;

    // assume error
    error = YES_ERROR;

    // call the BIOS query functions to obtain its size and contents
    structure_size = get_query_size(HEADER_ONLY);
    if (structure_size != 0)
    {
        // allocate memory buffer for query structure
        ptr = (unsigned char far *) malloc (structure_size);
        if (ptr != NULL)
        {
            // call the BIOS to fill memory
            if (fill_query_structure(HEADER_ONLY, ptr) == NO_ERROR)
            {
                // fill global query structure from memory buffer
                query_ptr = (QUERY_STRUCTURE far *)(ptr);
                memcpy(&querydata, query_ptr, sizeof(QUERY_STRUCTURE));

                // set no error condition
                error = NO_ERROR;
            }
            free(ptr);
        }
```

```
    }

    return (error);
}

/* ---------------------------------------------------------------------------
  SET_PACKED_PIXEL - set VGA controller into packed pixel mode

  The VGA controller is normally set in planar mode. Data transfer
  through the VGA aperture (low and high 32K pages) requires that the
  VGA controller be set in a packed pixel mode where the pixel data
  is arranged contigiously.
--------------------------------------------------------------------------- */
void set_packed_pixel(void)
{
    union REGS regs;

    // set VGA controller into packed pixel mode by setting superVGA mode 62h
    regs.x.ax = 0x62;
    int86(0x10, &regs, &regs);

    // set VGA controller space to 128k (A0000h-BFFFFh) - this is necessary
    // to allow access to GUI registers in VGA space (BFC00h) since setting
    // mode 62h sets VGA controller space to 64K (A0000h-AFFFFh).
    outp(0x3ce, 6);
    outp(0x3cf, 1);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

MEMREG.C

```
/*=========================================================================
  MEMREG.C

  MACH 64 functions to access the memory mapped registers.

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

/* -------------------------------------------------------------------------
  APP_REGW - write to a memory mapped register through the linear aperture.

  See REGW() for method description.
------------------------------------------------------------------------- */
void app_regw(unsigned int regindex, unsigned long regdata)
{
    unsigned long appaddr;
    unsigned long *dataptr;

    // calculate aperture address
    appaddr = modeinfo.aperture_address + modeinfo.linear_memreg_offset +
regindex;

    // setup dataptr to write from
    dataptr = &regdata;

    movemem(dataptr, appaddr, 2, 1);
}

/* -------------------------------------------------------------------------
  APP_REGR - read from a memory mapped register through the linear aperture.

  See REGR() for method description.
------------------------------------------------------------------------- */
unsigned long app_regr(unsigned int regindex)
{
    unsigned long appaddr, regdata;
    unsigned long *dataptr;

    // calculate aperture address
    appaddr = modeinfo.aperture_address + modeinfo.linear_memreg_offset +
```

```
regindex;

    // setup dataptr to read to
    dataptr = &regdata;

    movemem(dataptr, appaddr, 2, 0);

    return (regdata);
}

/* ---------------------------------------------------------------------
  REGW - write to a memory mapped register through an available aperture.

  This function will provide write access to the memory mapped registers.
  Each register is 32 bits wide. The appropriate method is selected based
  which aperture is enabled. Preference is given to the small VGA aperture
  since this method will work on all bus types. It is assumed that one of
  the two apertures is enabled.


  VGA aperture method:

    This method will provide write access on all bus types. It is assumed
    that the VGA aperture is enabled. The base address of the memory mapped
    registers is B000:FC00h. Each register occupies 4 bytes. This is also
    the fastest method for real mode applications.


  Linear aperture method:

    This method will provide write access on all bus types except ISA. ISA
    will work if the linear aperture can be enabled without mapping to
    existing extended memory. It is assumed that the linear memory aperture
    is enabled. The base address of the memory mapped registers is (base
    aperture address + 3FFC00h) for a 4M aperture size and (base aperture
    address + 7FFC00h) for an 8M aperture size. Each register occupies 4
    bytes. This method will impose a heavy performance hit since the memory
    mapped registers exist in extended memory above real mode space.
--------------------------------------------------------------------- */
void regw(unsigned int regindex, unsigned long regdata)
{
    if (modeinfo.vga_aperture_status == VGA_APERTURE_ENABLED)
    {
        *((unsigned long far *)(VGA_REGISTER_BASE + regindex)) = regdata;
    }
    else if (modeinfo.linear_aperture_status == LINEAR_APERTURE_ENABLED)
    {
        app_regw(regindex, regdata);
    }
}

/* ---------------------------------------------------------------------
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
  REGR - read from a memory mapped register through an available aperture.

  This function will provide read access to the memory mapped registers.
  Each register is 32 bits wide. The appropriate method is selected based
  which aperture is enabled. Preference is given to the small VGA aperture
  since this method will work on all bus types. It is assumed that one of
  the two apertures is enabled.


  VGA aperture method:

    This method will provide read access on all bus types. It is assumed
    that the VGA aperture is enabled. The base address of the memory mapped
    registers is B000:FC00h. Each register occupies 4 bytes. This is also
    the fastest method for real mode applications.


  Linear aperture method:

    This method will provide read access on all bus types except ISA. ISA
    will work if the linear aperture can be enabled without mapping to
    existing extended memory. It is assumed that the linear memory aperture
    is enabled. The base address of the memory mapped registers is (base
    aperture address + 3FFC00h) for a 4M aperture size and (base aperture
    address + 7FFC00h) for an 8M aperture size. Each register occupies 4
    bytes.
---------------------------------------------------------------------------- */
unsigned long regr(unsigned int regindex)
{
    if (modeinfo.vga_aperture_status == VGA_APERTURE_ENABLED)
    {
        return (*((unsigned long far *)(VGA_REGISTER_BASE + regindex)));
    }
    else if (modeinfo.linear_aperture_status == LINEAR_APERTURE_ENABLED)
    {
        return(app_regr(regindex));
    }
}

/* ----------------------------------------------------------------------------
  IOW - write to an io mapped register in 32 bit wide data.

  This function will write to an io register in 32 bits. This is done in
  two 16 bit writes - first to the low word io address, then to the upper
  word address.
---------------------------------------------------------------------------- */
void iow(unsigned int ioaddr, unsigned long iodata)
{
    outpw(ioaddr, (unsigned int)(iodata & 0xffff));
    outpw(ioaddr+2, (unsigned int)((iodata >> 16) & 0xffff));
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
/* ------------------------------------------------------------------------
  IOR - read from an io mapped register in 32 bit wide data.

  This function will read an io register in 32 bits. This is done in
  two 16 bit read - first from the low word io address, then from the upper
  word address.
------------------------------------------------------------------------ */
unsigned long ior(unsigned int ioaddr)
{
    unsigned long regdata;

    regdata = (unsigned long)(inpw(ioaddr+2));
    regdata = (unsigned long)((regdata << 16) | (inpw(ioaddr)));

    return (regdata);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

WAIT.C

```
/*=========================================================================
  WAIT.C

  MACH 64 functions to wait for engine idle

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

unsigned long fifo_stat;
unsigned long gui_stat;

/* -------------------------------------------------------------------------
  WAIT_FOR_IDLE - wait until engine active bit is idle.

  This function uses the DOS tick counter to serve as a timeout clock. If
  the engine is in a lockup condition, the busy bit may stay set. In this
  case, a timeout will occur, an error message will occur, and the program
  will terminate.
------------------------------------------------------------------------- */
void wait_for_idle(void)
{
    unsigned int starttick, endtick;

    // insure fifo is empty before waiting for engine idle
    wait_for_fifo(16);

    starttick = *((unsigned int far *)(DOS_TICK_ADDRESS));
    endtick = starttick;
    while ((regr(GUI_STAT) & ENGINE_BUSY) != ENGINE_IDLE)
    {
        endtick = *((unsigned int far *)(DOS_TICK_ADDRESS));
        if (abs(endtick - starttick) > IDLE_TIMEOUT)
        {
            fifo_stat = regr(FIFO_STAT);
            gui_stat = regr(GUI_STAT);
            terminate(BAD_IDLE);
        }
    }
}
```

```
/* ------------------------------------------------------------------------
  WAIT_FOR_FIFO - wait n empty FIFO entries.

  The FIFO contains upto 16 empty entries. The 'entries' value must be
  1 to 16. This function implements the same timeout mechanism as the
  Waitforidle() function.
------------------------------------------------------------------------ */
void wait_for_fifo(int entries)
{
    unsigned int starttick, endtick;

    starttick = *((unsigned int far *)(DOS_TICK_ADDRESS));
    endtick = starttick;
    while ((regr(FIFO_STAT) & 0xffff) > ((unsigned int)(0x8000 >> entries)))
    {
        endtick = *((unsigned int far *)(DOS_TICK_ADDRESS));
        if (abs(endtick - starttick) > FIFO_TIMEOUT)
        {
            fifo_stat = regr(FIFO_STAT);
            gui_stat = regr(GUI_STAT);
            terminate(BAD_FIFO);
        }
    }
}

/* ------------------------------------------------------------------------
  TERMINATE - terminate the program due to a timeout from Waitforidle() or
              Waitforfifo().
------------------------------------------------------------------------ */
void terminate(int idle_problem)
{
    // disable accelerator and switch back to VGA mode
    close_mode();

    // print error message
    if (idle_problem == BAD_IDLE)
    {
        printf("Error. Timeout exceeded for engine idle.\n");
    }
    else  // BAD_FIFO
    {
        printf("Error. Timeout exceeded for engine fifo.\n");
    }

    printf("Status at error:\n");
    printf("  Idle status  : %08lx\n", gui_stat & 1);
    printf("  Fifo status  : %08lx\n", fifo_stat & 0xffff);
    printf("  Fifo overflow: %08lx\n", fifo_stat >> 31);

    exit (1);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

MOVEMEM.ASM

```
; ============================================================================
; MOVEMEM.ASM
;
; Routine to move a block of data from or to extended memory. This routine
; will typically be used to write and read from the MACH64 aperture memory.
;
; Compiling:
;   masm /Ml /D<memory model> movemem.asm;
;       <memory model> = mem_S for SMALL model,
;                        mem_M for MEDIUM model,
;                        mem_L for LARGE model
;
; Copyright (c) 1993 ATI Technologies Inc.  All rights reserved
; ============================================================================

IFDEF mem_S
PARM        equ     4   ; passed parameters start at bp+4 for small model
ELSE
PARM        equ     6   ; passed parameters start at bp+6 for other models
ENDIF

IFDEF mem_S
.MODEL  SMALL, C
ELSEIFDEF mem_M
.MODEL  MEDIUM, C
ELSE
.MODEL  LARGE, C
ENDIF

.DATA

EVEN
                ; gdt structure required for INT 15h, function 87h call
gdt             dw      0,0,0,0
                dw      0,0,0,0
source_len      dw      0ffffh              ; set to 0ffffh
source_addr_low dw      ?                   ; 24 bit address
source_addr_hi  db      ?
source_access   db      93h                 ; set to 93h
source_addr_ext dw      0                   ; 386/486 address extensions
target_len      dw      0ffffh              ; set to 0ffffh
target_addr_low dw      ?                   ; 24 bit address
target_addr_hi  db      ?
target_access   db      93h                 ; set to 93h
target_addr_ext dw      0                   ; 386/486 address extensions
                dw      0,0,0,0
                dw      0,0,0,0

.CODE
```

```
.286

; Macro for 'call' model handling
Mcall       macro   routine
IFDEF mem_S
            call    NEAR PTR routine
ELSE
            call    FAR PTR routine
ENDIF
                        endm

; ---------------------------------------------------------------------------
; MOVEMEM - Move a block of data to or from an extended address (at or above
;           the 1Meg memory boundary). Int 15h, function 87h is used for this
;           function. The input parameters are fetched from the stack.
;
; Inputs : FAR PTR: memory buffer address (segment:offset),
;          DWORD  : extended physical memory address,
;          WORD   : number of words to transfer (<= 32767 words),
;          WORD   : transfer direction flag
;                     0 -> READ  (extended memory to memory buffer)
;                     1 -> WRITE (memory buffer to extended memory)
;
; Outputs: Return value from Int 15h, function 87h in ax
;              0 for success,
;             >0 for error code
;
; C declaration for this function:
;
;     int movemem(void far *ptr,
;                 unsigned long dest,
;                 unsigned int nwords,
;                 int direction);
;
; The memory buffer address pointer is expected in segment:offset format. The
; 'dest' parameter is expected in physical address format.
; ---------------------------------------------------------------------------
            public  movemem

IFDEF mem_S
movemem     proc    near
ELSE
movemem     proc    far
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    bx
            push    cx
            push    dx
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
                push    di
                push    si
                push    es

                ; retrieve direction flag in di
                mov     di, WORD PTR [bp+PARM+10]

                ; setup gdt structure
                mov     source_len, 0ffffh
                mov     target_len, 0ffffh
                mov     source_access, 93h
                mov     target_access, 93h

                ; calculate physical address from PTR address for GDT in dx:cx
                mov     dx, WORD PTR [bp+PARM+2] ; segment of PTR address
                mov     cx, dx
                clc
                shr     dx, 0ch
                shl     cx, 4
                mov     ax, WORD PTR [bp+PARM]   ; load offset of PTR address
                add     cx, ax
                jnc     mm1
                inc     dx
mm1:
                ; put extended physical memory address in bx:ax
                mov     ax, WORD PTR [bp+PARM+4] ; load low word of address
                mov     bx, WORD PTR [bp+PARM+6] ; load high word of address

                ; fill in address depending on direction flag
                cmp     di, 0                    ; 0 = read
                je      mm2

                ; WRITE: source = mem buffer, target = extended mem address
                mov     source_addr_low, cx
                mov     source_addr_hi, dl
                xor     dl, dl
                mov     source_addr_ext, dx
                mov     target_addr_low, ax
                mov     target_addr_hi, bl
                xor     bl, bl
                mov     target_addr_ext, bx
                jmp     mm3
mm2:
                ; READ: source = extended mem address, target = mem buffer
                mov     source_addr_low, ax
                mov     source_addr_hi, bl
                xor     bl, bl
                mov     source_addr_ext, bx
                mov     target_addr_low, cx
                mov     target_addr_hi, dl
                xor     dl, dl
                mov     target_addr_ext, dx
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
mm3:
            ; setup registers and call INT 15h, function 87h, gdt = es:si
            mov     cx, WORD PTR [bp+PARM+8] ; load word transfer count
            mov     si, offset ds:gdt
            push    ds
            pop     es
            clc
            mov     ah, 87h
            int     15h

            ; setup return value
            mov     al, ah
            xor     ah, ah

            ; restore saved registers
            pop     es
            pop     si
            pop     di
            pop     dx
            pop     cx
            pop     bx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

movemem     endp

            end
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

PALETTE.C

```
/*=============================================================================
  PALETTE.C

  Functions to set, get, and initialize the palette.

  Copyright (c) 1993 ATI Technologies Inc. All rights reserved
  ============================================================================*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

#include "atim64.h"
#include "sample.h"

/* ---------------------------------------------------------------------------
  SET_PALETTE - set a palette entry.

  This function sets a specific palette entry. Each component has a range
  of 0 to 255. The index has a range of 0 to 255.
--------------------------------------------------------------------------- */
void set_palette(int index, PALETTE entry)
{
    /* set DAC write index */
    outp(ioDAC_REGS, index);

    /* set red component */
    outp(ioDAC_REGS+1, entry.red);

    /* set green component */
    outp(ioDAC_REGS+1, entry.green);

    /* set blue component */
    outp(ioDAC_REGS+1, entry.blue);
}

/* ---------------------------------------------------------------------------
  GET_PALETTE - get a palette entry.

  This function gets a specific palette entry. Each component has a range
  of 0 to 255. The index has a range of 0 to 255.
--------------------------------------------------------------------------- */
PALETTE get_palette(int index)
{
    PALETTE entry;

    /* set DAC read index */
```

```
    outp(ioDAC_REGS+3, index);

    /* get red component */
    entry.red = inp(ioDAC_REGS+1);

    /* get green component */
    entry.green = inp(ioDAC_REGS+1);

    /* get blue component */
    entry.blue = inp(ioDAC_REGS+1);

    return (entry);
}

/* ---------------------------------------------------------------------------
  INIT_PALETTE - Set the palette to default values.

  This function initializes the palette table by setting the entries to a
  set of default values. The first 16 entries are set to EGA/VGA colors. Each
  EGA/VGA color is replicated 16 times to fill the 256 palette entries.
--------------------------------------------------------------------------- */
void init_palette(void)
{
    int i, j, index;
    PALETTE entry[16] =
    {
        { 0, 0, 0},          // black
        { 0, 0, 168},        // blue
        { 0, 168, 0},        // green
        { 0, 168, 168},      // cyan
        { 168, 0, 0},        // red
        { 168, 0, 168},      // magenta
        { 168, 168, 0},      // brown
        { 168, 168, 168},    // light gray
        { 84, 84, 84},       // gray
        { 0, 0, 255},        // light blue
        { 0, 255, 0},        // light green
        { 0, 255, 255},      // light cyan
        { 255, 0, 0},        // light red
        { 255, 0, 255},      // light magenta
        { 255, 255, 0},      // yellow
        { 255, 255, 255}     // white
    };

    // set first 16 entries
    for (index = 0; index < 16; index++)
    {
        set_palette(index, entry[index]);
    }

    // set other entries by replicating the first 16 entries
    index = 16;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    for (i = 1; i < 16; i++)
    {
        for (j = 0; j < 16; j++)
        {
            set_palette(index, entry[i]);
            index++;
        }
    }
}
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

ROMCALLS.ASM

```
; ============================================================================
; ROMCALLS.ASM
;
; Interface routines to access MACH64 ROM services.
;
; Compiling:
;   masm /Ml /D<memory model> romcalls.asm;
;       <memory model> = mem_S for SMALL model,
;                        mem_M for MEDIUM model,
;                        mem_L for LARGE model
;
; Copyright (c) 1993 ATI Technologies Inc. All rights reserved
; ============================================================================

include atim64.inc

IFDEF mem_S
PARM        equ     4   ; passed parameters start at bp+4 for small model
ELSE
PARM        equ     6   ; passed parameters start at bp+6 for other models
ENDIF

IFDEF mem_S
.MODEL  SMALL, C
ELSEIFDEF mem_M
.MODEL  MEDIUM, C
ELSE
.MODEL  LARGE, C
ENDIF

.DATA

ati_sig     db      '761295520', 0
rom_addr    dw      64h
            dw      0c000h
            db      0

.CODE
.286

; Macro for 'call' model handling
Mcall       macro   routine
IFDEF mem_S
            call    NEAR PTR routine
ELSE
            call    FAR PTR routine
ENDIF
                        endm
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; ------------------------------------------------------------------------
; IS_ATI_ROM
;
; Check if ROM has ATI signature. Normally, the starting location of the 9
; character signature is at ROM_SEGMENT:0031h. In an MCA system, this is not
; the case and can be anywhere in the first 128 bytes of the ROM. For this
; reason, a signature scan method is used.
;
; Inputs : none
;
; Outputs: Returns 1 in ax if signature was found, 0 if not
; ------------------------------------------------------------------------
            public  is_ati_rom

IFDEF mem_S
is_ati_rom  proc    near
ELSE
is_ati_rom  proc    far
ENDIF
            ; save registers used
            push    bx
            push    cx
            push    dx
            push    di
            push    si
            push    es

            ; setup ati signature string scan
            Mcall   rom_base            ; get rom segment in ax
            mov     dx, 0               ; assume no ATI ROM
            push    ax                  ; setup ROM search segment es
            pop     es
            mov     si, 0               ; start search at rom_seg:0000h
            mov     di, offset ati_sig  ; compare with ati_sig string
            mov     bl, byte ptr ds:[di] ; load first character of string
            mov     cx, 128             ; search first 128 bytes of ROM
match_chk:
            mov     al, byte ptr es:[si] ; get byte from ROM
            cmp     al, bl              ; does first character match
            jne     next_char

            ; compare with ati signature string
            push    si                  ; save si
            mov     dx, 1               ; assume ATI ROM
            mov     di, offset ati_sig  ; compare with ati_sig string
            inc     di                  ; start at second character
            inc     si
sig_check:
            mov     al, byte ptr es:[si]
            mov     ah, byte ptr ds:[di]
            inc     si
            inc     di
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            cmp     ah, 0               ; 0 = end of string
            je      end_check
            cmp     al, ah
            je      sig_check
            mov     dx, 0               ; not ATI ROM
end_check:
            pop     si                  ; restore si
            cmp     dx, 0               ; check for string match
            jne     sig_found

next_char:
            inc     si
            loop    match_chk

sig_found:
            mov     ax, dx              ; 0 = not found, 1 = found

            ; restore saved registers
            pop     es
            pop     si
            pop     di
            pop     dx
            pop     cx
            pop     bx

            ret

is_ati_rom  endp

; ---------------------------------------------------------------------------
; ROM_BASE
;
; Retrieve base segment of MACH64 ROM.
;
; Inputs : none
;
; Outputs: Returns base rom segment in ax (usually C000h or C800h)
; ---------------------------------------------------------------------------
            public  rom_base

IFDEF mem_S
rom_base    proc    near
ELSE
rom_base    proc    far
ENDIF
            ; save registers used
            push    cx
            push    dx

            ; retrieve rom segment address from MACH64 register
            mov     dx, ioSCRATCH_REG1
            in      al, dx
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            and     al, 7Fh
            mov     ah, 0
            mov     cl, 7
            shl     ax, cl
            add     ax, 0C000h

            ; restore saved registers
            pop     dx
            pop     cx

            ret

rom_base    endp

; ---------------------------------------------------------------------------
; LOAD_MODE_PARMS
;
; Load accelerator mode parameters to determine the mode to be invoked by a
; following call to SET_DISPLAY_MODE. Input parameters are fetched from the
; stack.
;
; Inputs : WORD resolution code
;            CH = 12h - 640x480
;            CH = 6Ah - 800x600
;            CH = 55h - 1024x768
;            CH = 83h - 1280x1024
;
;          WORD pitch code
;            CL [bits 7-6] = 0 - 1024
;            CL [bits 7-6] = 1 - don't change
;            CL [bits 7-6] = 2 - pitch size = resolution width
;
;          WORD deep color code
;            CL [bits 3-0] = 1 - 4 bpp
;            CL [bits 3-0] = 2 - 8 bpp
;            CL [bits 3-0] = 3 - 15 bpp (555)
;            CL [bits 3-0] = 4 - 16 bpp (565)
;            CL [bits 3-0] = 5 - 24 bpp
;            CL [bits 3-0] = 6 - 32 bpp
;
; Outputs: Returns error code in ax
;            AX = 0 - no error
;            AX = 1 - function complete with error
;            AX = 2 - function not supported
;
; ---------------------------------------------------------------------------
            public  load_mode_parms

IFDEF mem_S
load_mode_parms proc near
ELSE
load_mode_parms proc far
```

```
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    cx

            ; setup parameters for call to ATI rom
            Mcall   rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     ax, WORD PTR [bp+PARM]      ; get resolution code
            mov     ch, al
            mov     ax, WORD PTR [bp+PARM+2]    ; get pitch code
            shl     al, 6
            mov     cl, al
            mov     ax, WORD PTR [bp+PARM+4]    ; get deep color code
            and     al, 7
            or      cl, al
            mov     ax, 0                       ; function code 0
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; setup error code in AL
            mov     al, ah
            xor     ah, ah

            ; restore saved registers
            pop     cx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

load_mode_parms endp

; ---------------------------------------------------------------------------
; SET_DISPLAY_MODE
;
; Set display to accelerator (after a call to LOAD_MODE_PARMS()) or VGA mode.
; Input parameters are fetched from the stack.
;
; Inputs : WORD display mode
;            CL = 0 - VGA
;            CL = 1 - Accelerator
;
; Outputs: Returns error code in ax
;            AX = 0 - no error
;            AX = 1 - function complete with error
;            AX = 2 - function not supported
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
;
; ------------------------------------------------------------------------
            public  set_display_mode

IFDEF mem_S
set_display_mode proc near
ELSE
set_display_mode proc far
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    cx

            ; setup parameters for call to ATI rom
            Mcall   rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     ax, WORD PTR [bp+PARM]      ; get display mode flag
            mov     cl, al
            mov     ax, 1                       ; function code 1
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; setup error code in AL
            mov     al, ah
            xor     ah, ah

            ; restore saved registers
            pop     cx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

set_display_mode endp

; ------------------------------------------------------------------------
; LOAD_AND_SET_MODE
;
; Load accelerator mode parameters and set accelerator mode. Input parameters
; are fetched from the stack.
;
; Inputs : WORD resolution code
;            CH = 12h - 640x480
;            CH = 6Ah - 800x600
;            CH = 55h - 1024x768
;            CH = 83h - 1280x1024
;
```

```
;           WORD pitch code
;             CL [bits 7-6] = 0 - 1024
;             CL [bits 7-6] = 1 - don't change
;             CL [bits 7-6] = 2 - pitch size = resolution width
;
;           WORD deep color code
;             CL [bits 3-0] = 1 - 4 bpp
;             CL [bits 3-0] = 2 - 8 bpp
;             CL [bits 3-0] = 3 - 15 bpp (555)
;             CL [bits 3-0] = 4 - 16 bpp (565)
;             CL [bits 3-0] = 5 - 24 bpp
;             CL [bits 3-0] = 6 - 32 bpp
;
; Outputs: Returns error code in ax
;             AX = 0 - no error
;             AX = 1 - function complete with error
;             AX = 2 - function not supported
;
; ---------------------------------------------------------------------------
            public  load_and_set_mode

IFDEF mem_S
load_and_set_mode proc near
ELSE
load_and_set_mode proc far
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    cx

            ; setup parameters for call to ATI rom
            Mcall   rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     ax, WORD PTR [bp+PARM]      ; get resolution code
            mov     ch, al
            mov     ax, WORD PTR [bp+PARM+2]    ; get pitch code
            shl     al, 6
            mov     cl, al
            mov     ax, WORD PTR [bp+PARM+4]    ; get deep color code
            and     al, 7
            or      cl, al
            mov     ax, 2                       ; function code 2
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; setup error code in AL
            mov     al, ah
            xor     ah, ah
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            ; restore saved registers
            pop     cx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

load_and_set_mode endp

; ---------------------------------------------------------------------------
; GET_QUERY_SIZE
;
; Retrieve the query size in bytes to reserve for a subsequent call to
; FILL_QUERY_STRUCTURE. The size of the header or header & mode tables can be
; retrieved. Input parameters are fetched from the stack.
;
; Inputs : WORD information type
;            CL = 0 - header information only
;            CL = 1 - header and mode table information
;
; Outputs: Returns size in bytes in ax
;            AX = size in bytes
;               or
;            AX = 0 if the function returns with an error
; ---------------------------------------------------------------------------
            public  set_display_mode

IFDEF mem_S
get_query_size proc near
ELSE
get_query_size proc far
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    cx

            ; setup parameters for call to ATI rom
            Mcall   rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     ax, WORD PTR [bp+PARM]      ; get info type flag
            mov     cl, al
            mov     ax, 8                       ; function code 8
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; check for errors in AH
            cmp     ah, 0
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            je      no_error

            ; error: set ax to zero
            mov     ax, 0
            jmp     qsize_exit

no_error:
            ; no error: move size from cx to ax
            mov     ax, cx

qsize_exit:
            ; restore saved registers
            pop     cx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

get_query_size endp

; ----------------------------------------------------------------------------
; FILL_QUERY_STRUCTURE
;
; Fill a query structure containing information about the installed hardware.
; This function should be called after called GET_QUERY_SIZE and with the
; same information type flag. This insures that the structure is large
; enough to hold the fill information. Input parameters are fetched from the
; stack.
;
; Inputs : WORD information type
;            CL = 0 - header information only
;            CL = 1 - header and mode table information
;
;          FAR POINTER address of structure to be filled (DX:BX)
;
; Outputs: Returns error code in ax
;            AX = 0 - no error
;            AX = 1 - function complete with error
;            AX = 2 - function not supported
; ----------------------------------------------------------------------------
            public  set_display_mode

IFDEF mem_S
fill_query_structure proc near
ELSE
fill_query_structure proc far
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            ; save registers used
            push    bx
            push    cx
            push    dx

            ; setup parameters for call to ATI rom
            Mcall   rom_base                    ; get rom segment in ax
            mov     rom_addr+2, ax
            mov     cx, WORD PTR [bp+PARM]      ; get info type flag
            mov     bx, WORD PTR [bp+PARM+2]    ; get offset of address
            mov     dx, WORD PTR [bp+PARM+4]    ; get segment of address
            mov     ax, 9                       ; function code 9
            mov     rom_addr, 64h
            call    DWORD PTR rom_addr          ; call ROM

            ; setup error code in AL
            mov     al, ah
            xor     ah, ah

            ; restore saved registers
            pop     dx
            pop     cx
            pop     bx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

fill_query_structure endp

            end
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

VINT.ASM

```
; ===========================================================================
; VINT.ASM
;
; Interface routines to handle MACH64 CRTC vertical interrupts.
;
; Compiling:
;   masm /Ml /D<memory model> romcalls.asm;
;       <memory model> = mem_S for SMALL model,
;                        mem_M for MEDIUM model,
;                        mem_L for LARGE model
;
; Copyright (c) 1993 ATI Technologies Inc. All rights reserved
; ===========================================================================

include ..\util\atim64.inc

IFDEF mem_S
PARM        equ     4   ; passed parameters start at bp+4 for small model
ELSE
PARM        equ     6   ; passed parameters start at bp+6 for other models
ENDIF

IFDEF mem_S
.MODEL  SMALL, C
ELSEIFDEF mem_M
.MODEL  MEDIUM, C
ELSE
.MODEL  LARGE, C
ENDIF

.DATA

.CODE
.286

; Macro for 'call' model handling
Mcall       macro   routine
IFDEF mem_S
            call    NEAR PTR routine
ELSE
            call    FAR PTR routine
ENDIF
                        endm

; ---------------------------------------------------------------------------
; ENABLE_VBLANKINT
;
; Enable the vertical blank interrupt bit.
;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; Inputs : none
;
; Outputs: none
; ------------------------------------------------------------------------
            public  enable_vblankint

IFDEF mem_S
enable_vblankint proc    near
ELSE
enable_vblankint proc    far
ENDIF
            ; save registers used
            push    dx

            ; set vertical blank interrupt bit
            mov     dx, ioCRTC_INT_CNTL
            in      al, dx
            or      al, 2
            out     dx, al

            ; restore saved registers
            pop     dx

            ret

enable_vblankint endp

; ------------------------------------------------------------------------
; DISABLE_VBLANKINT
;
; Enable the vertical blank interrupt bit.
;
; Inputs : none
;
; Outputs: none
; ------------------------------------------------------------------------
            public  disable_vblankint

IFDEF mem_S
disable_vblankint proc    near
ELSE
disable_vblankint proc    far
ENDIF
            ; save registers used
            push    dx

            ; clear vertical blank interrupt bit
            mov     dx, ioCRTC_INT_CNTL
            in      al, dx
            and     al, 0fbh
            out     dx, al
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            ; restore saved registers
            pop     dx

            ret

disable_vblankint endp

; ---------------------------------------------------------------------------
; WAIT_FOR_VBLANK
;
; Wait until vertical blank interrupt bit is set. It is assumed that the
; vertical blank interrupt enable bit is set.
;
; Inputs : none
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  wait_for_vblank

IFDEF mem_S
wait_for_vblank  proc    near
ELSE
wait_for_vblank  proc    far
ENDIF
            ; save registers used
            push    dx

            ; ack vertical blank interrupt to clear it
            mov     dx, ioCRTC_INT_CNTL
            in      al, dx
            or      al, 6
            out     dx, al

            ; wait until it gets set again
chkvblank:
            in      al, dx
            and     al, 4
            cmp     al, 4
            jne     chkvblank

            ; restore saved registers
            pop     dx

            ret

wait_for_vblank  endp

; ---------------------------------------------------------------------------
; SET_VLINE
;
; Inputs : WORD vertical line to set interrupt bit at.
;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; Outputs: none
; ------------------------------------------------------------------------
            public  set_vline

IFDEF mem_S
set_vline    proc    near
ELSE
set_vline    proc    far
ENDIF
            ; create frame pointer
            push    bp
            mov     bp, sp

            ; save registers used
            push    dx

            ; set vertical line to set interrupt bit at
            mov     ax, WORD PTR [bp+PARM]
            mov     dx, ioCRTC_VLINE_CRNT_VLINE
            out     dx, ax

            ; restore saved registers
            pop     dx

            ; remove frame pointer
            mov     sp, bp
            pop     bp

            ret

set_vline    endp

; ------------------------------------------------------------------------
; ENABLEVLINEINT
;
; Enable the vertical line interrupt bit.
;
; Inputs : none
;
; Outputs: none
; ------------------------------------------------------------------------
            public  enable_vlineint

IFDEF mem_S
enable_vlineint proc    near
ELSE
enable_vlineint proc    far
ENDIF
            ; save registers used
            push    dx

            ; set vertical line interrupt bit
```

```
            mov     dx, ioCRTC_INT_CNTL
            in      al, dx
            or      al, 8
            out     dx, al

            ; restore saved registers
            pop     dx

            ret

enable_vlineint endp

; ---------------------------------------------------------------------------
; DISABLEVLINEINT
;
; Enable the vertical blank interrupt bit.
;
; Inputs : none
;
; Outputs: none
; ---------------------------------------------------------------------------
            public  disable_vlineint

IFDEF mem_S
disable_vlineint proc    near
ELSE
disable_vlineint proc    far
ENDIF
            ; save registers used
            push    dx

            ; clear vertical line interrupt bit
            mov     dx, ioCRTC_INT_CNTL
            in      al, dx
            and     al, 0f7h
            out     dx, al

            ; restore saved registers
            pop     dx

            ret

disable_vlineint endp

; ---------------------------------------------------------------------------
; WAIT_FOR_VLINE
;
; Wait until vertical line interrupt bit is set. It is assumed that the
; vertical line interrupt enable bit is set. Also, it is assumed that the
; CRTC_VLINE vertical line has been previously set in the
; CRTC_VLINE_CRNT_VLINE register.
;
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
; Inputs : none
;
; Outputs: none
; ------------------------------------------------------------------------
            public  wait_for_vline

IFDEF mem_S
wait_for_vline   proc    near
ELSE
wait_for_vline   proc    far
ENDIF
            ; save registers used
            push    dx

            ; ack vertical line interrupt to clear it
            mov     dx, ioCRTC_INT_CNTL
            in      al, dx
            or      al, 18h
            out     dx, al

            ; wait until it gets set again
chkvline:
            in      al, dx
            and     al, 10h
            cmp     al, 10h
            jne     chkvline

            ; restore saved registers
            pop     dx

            ret

wait_for_vline   endp

            end
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

VINT.H

```
/*=========================================================================
  VINT.H

  C include file for functions in VINT.C

  Copyright (c) 1993 ATI Technologies Inc.  All rights reserved
 =========================================================================*/

/* Routines in VINT.C */
void enable_vblankint(void);
void disable_vblankint(void);
void wait_for_vblank(void);

void set_vline(int vline);
void enable_vlineint(void);
void disable_vlineint(void);
void wait_for_vline(void);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

VTGA.C

```
/*=========================================================================
 VTGA.C

 Routines to bring in a TGA file to video memory for the sample code examples.

 Copyright (c) 1993 ATI Technologies Inc. All rights reserved
 =========================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#include "..\util\atim64.h"
#include "..\util\sample.h"
#include "vtga.h"

/*-------------------------------------------------------------------------
    GET_TARGA_HEADER - load targa file header information

    Inputs      : filename,
                  pointer to header structure

    Return codes: SUCCESS    - success
                  OPEN_ERROR - can't open file
-------------------------------------------------------------------------*/
int get_targa_header(char *filename, TARGA_HEADER *header)
{
    FILE *TargaFile;

    // open targa file
    TargaFile = fopen(filename, "rb");
    if (TargaFile == NULL)
    {
        return (OPEN_ERROR);
    }

    // check for support ability (type 1, 8 bpp, color map)
    fread(header, sizeof(TARGA_HEADER), 1, TargaFile);

    fclose(TargaFile);

    return (SUCCESS);
}

/*-------------------------------------------------------------------------
    SET_TARGA_PALETTE - load a targa image color table and set the palette

    Inputs      : filename of targa image file
```

```
    Return codes: SUCCESS          - success
                  OPEN_ERROR       - can't open file
                  IMAGE_TYPE_ERROR - targa file does not have color table

    Notes       : - a valid set accelerator mode is assumed
                  - the image size is intended to be fairly small
                  - accelerator mode is assumed to be 8 bpp
-------------------------------------------------------------------------*/
int set_targa_palette(char *filename)
{
    FILE *TargaFile;
    TARGA_HEADER header;
    PALETTE entry;
    int i;

    // open targa file
    TargaFile = fopen(filename, "rb");
    if (TargaFile == NULL)
    {
        return (OPEN_ERROR);
    }

    // check for support ability (type 1, 8 bpp, color map)
    fread(&header, sizeof(TARGA_HEADER), 1, TargaFile);

    // support check for type 1, 8 bpp, color map
    if ((header.cm_type != 1) ||
        (header.image_type != 1) ||
        (header.pixel_depth != 8) ||
        (header.image_coord == 1) ||
        (header.image_coord == 3))
    {
        // return error if image type is not supported (no color table)
        fclose(TargaFile);
        return (IMAGE_TYPE_ERROR);
    }

    // read color map and setup palette for 8 bpp
    for (i = header.cm_origin; i < header.cm_length; i++)
    {
        entry.blue = fgetc(TargaFile) >> 2;
        entry.green = fgetc(TargaFile) >> 2;
        entry.red = fgetc(TargaFile) >> 2;
        set_palette(i, entry);
    }

    fclose(TargaFile);

    // return to caller
    return (SUCCESS);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
/*-------------------------------------------------------------------------
    LOAD_TARGA - load a targa image into video memory

    Inputs      : filename of targa image file,
                  (x, y) starting position (top-left corner),
                  paletteflag (set or don't set for 8 bpp mode) -
                    NOSET_PALETTE or SET_PALETTE

    Return codes: SUCCESS          - success
                  OPEN_ERROR       - can't open file
                  MEMORY_ERROR     - not enough memory for input buffer
                  IMAGE_TYPE_ERROR - targa file is not supported

    Notes       : - a valid set accelerator mode is assumed
                  - the image size is intended to be fairly small
                  - accelerator mode is assumed to be 8 bpp
-------------------------------------------------------------------------*/
int load_targa(char *filename, int x, int y)
{
    FILE *TargaFile;
    TARGA_HEADER header;
    unsigned char *inbuffer;
    unsigned long temp1, temp2, temp3, temp4, data;
    unsigned long totalbytes, hostwrts;
    int drawx, drawy, j;
    int shifter, remainder;
    int adder, error;
    unsigned char r, g, b;
    unsigned char value;

    // open targa file
    TargaFile = fopen(filename, "rb");
    if (TargaFile == NULL)
    {
        return (OPEN_ERROR);
    }

    // check for support ability (type 1, 8 bpp, color map)
    fread(&header, sizeof(TARGA_HEADER), 1, TargaFile);

    // support check for type 1, 8 bpp, color map
    error = 0;
    if ((header.cm_type != 1) ||
        (header.image_type != 1) ||
        (header.pixel_depth != 8) ||
        (header.image_coord == 1) ||
        (header.image_coord == 3))
    {
        error++;
    }
```

```
    // support check for type 2, 24 bpp, no color map
    if ((header.cm_type != 0) ||
       (header.image_type != 2) ||
       (header.pixel_depth != 24) ||
       (header.image_coord == 1) ||
       (header.image_coord == 3))
    {
        error++;
    }

    // return error if image type is not supported
    if (error > 1)
    {
        fclose(TargaFile);
        return (IMAGE_TYPE_ERROR);
    }

    // skip over color table if 8 bpp image type
    if (header.pixel_depth == 8)
    {
        fseek(TargaFile, (unsigned long)(header.cm_length * 3), SEEK_CUR);
    }

    // open input buffer (image width * image pixel depth in bytes)
   inbuffer = (unsigned char *) malloc ((header.width + 1) * (header.pixel_depth
/ 8));
    if (inbuffer == NULL)
    {
        printf("Not enough memory to show targa file.\n");
        fclose(TargaFile);
        return (MEMORY_ERROR);
    }

    // determine if an additional host write is needed at the end of the loop
    remainder = 0;
   totalbytes = (unsigned long)(header.width) * (unsigned long)(header.height);
    hostwrts = totalbytes / 4;
    if ((hostwrts * 4) != totalbytes)
    {
        remainder++;
    }

    // setup engine for buffer to screen host data transfer
    wait_for_idle();
    temp1 = regr(DP_SRC);
    temp2 = regr(DP_MIX);
    temp3 = regr(SRC_CNTL);
    temp4 = regr(DST_CNTL);

    regw(DP_SRC, FRGD_SRC_HOST);
    regw(DP_MIX, FRGD_MIX_S | BKGD_MIX_S);
    regw(SRC_CNTL, 0);
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
    if (header.image_coord == 0)
    {
        // bottom left
        regw(DST_CNTL, DST_Y_BOTTOM_TO_TOP | DST_X_LEFT_TO_RIGHT);
        regw(DST_X, x);
        regw(DST_Y, y + header.height - 1);
        regw(DST_HEIGHT, header.height);
        regw(DST_WIDTH, header.width);
    }
    else
    {
        // top left
        regw(DST_CNTL, DST_Y_TOP_TO_BOTTOM | DST_X_RIGHT_TO_LEFT);
        regw(DST_X, x + header.width - 1);
        regw(DST_Y, y);
        regw(DST_HEIGHT, header.height);
        regw(DST_WIDTH, header.width);
    }

    // main draw loop
    if (header.pixel_depth == 24)
    {
        adder = 3;
    }
    else
    {
        adder = 1;
    }
    j = 0;
    shifter = 0;
    data = 0;
    for (drawy = 0; drawy < header.height; drawy++)
    {
        // get next image scanline
        fread((unsigned char *)inbuffer,
              header.width * (header.pixel_depth / 8), 1, TargaFile);

        // get next line of pixels from input buffer
       for (drawx = 0; drawx < (header.width * (header.pixel_depth / 8)); drawx
= drawx + adder)
        {
            if (header.pixel_depth == 24)
            {
                // get 24 bpp data and convert to 8 bpp
                g = (unsigned char)(*(inbuffer + drawx) & 0xff);
                b = (unsigned char)(*(inbuffer + drawx + 1) & 0xff);
                r = (unsigned char)(*(inbuffer + drawx + 2) & 0xff);
               value = (unsigned char)((r & 0xe0) | ((g >> 3) & 0x1c) | ((b >>
6) & 0x03));
            }
            else
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

```
            {
                // get 8 bit data
                value = (unsigned char)(*(inbuffer + drawx) & 0xff);
            }

            data = data | ((unsigned long)(value) << shifter);
            shifter = shifter + 8;

            // accumulate 32 bits at a time before writing
            j++;
            if (j > 3)
            {
                wait_for_fifo(1);
                regw(HOST_DATA0, data);
                j = 0;
                shifter = 0;
                data = 0;
            }
        }
    }

    // write remaining data if needed
    if (remainder != 0)
    {
        wait_for_fifo(1);
        regw(HOST_DATA0, data);
    }

    // restore main engine context
    wait_for_idle();
    regw(DP_SRC, temp1);
    regw(DP_MIX, temp2);
    regw(SRC_CNTL, temp3);
    regw(DST_CNTL, temp4);

    // close buffers and targa file
    free(inbuffer);
    fclose(TargaFile);

    // return to caller
    return (SUCCESS);
}
```


© 1993 ATI Technologies Inc.
Proprietary and Confidential

VTGA.H

```
/*=========================================================================
 VTGA.H

 Header for VTGA.C file.

 Copyright (c) 1993 ATI Technologies Inc. All rights reserved
 =========================================================================*/

/* Return codes for targa routines */
#define SUCCESS           0
#define OPEN_ERROR        1
#define MEMORY_ERROR      2
#define IMAGE_TYPE_ERROR  3

/* Use BYTE alignment */
#pragma pack(1)

typedef struct
{
    char id_length;     // file id length (follows header, if applicable)
    char cm_type;       // 0 = no color map, 1 = color map
    char image_type;    // 1 = uncompressed, color map
    int  cm_origin;     // starting color map index
    int  cm_length;     // number of color map entries
    char cm_entrysize; // bits/color map entry
    int  x_origin;      // starting x position
    int  y_origin;      // starting y position
    int  width;         // image width in pixels
    int  height;        // image height in pixels
    char pixel_depth;   // image bits per pixel (8, 16, 24)
    char image_coord;   // image start coordinates
                                //  0 - bottom left
                                //  1 - bottom right
                                //  2 - top left
                                //  3 - top right
} TARGA_HEADER;

/* Restore to WORD alignment */
#pragma pack()


/* Routines in VTGA.C */

int get_targa_header(char *filename, TARGA_HEADER *header);
int set_targa_palette(char *filename);
int load_targa(char *filename, int x, int y);
```

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Glossary of Terms

A

alternate-fill Xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx.

B

background

Bresenham

C

clipping

color source

context

CRTC

D

destination aligned

destination compare

destination trajectory

direct color

draw engine

F

foreground

H

host

host data

L

linear frame buffer

M

mix
monochrome
monochrome source

O

offset

P

packed monochrome
paged frame buffer
palette
pattern
pitch
pixel depth
pseudocolor

R

ROP

S

scissoring
side effect
source aligned
source compare
source trajectory

T

tiling
trajectory

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Index

© 1993 ATI Technologies Inc.
Proprietary and Confidential

D

© 1993 ATI Technologies Inc.
Proprietary and Confidential

E

F

G

H

I

L

M

N

O

P

© 1993 ATI Technologies Inc.
Proprietary and Confidential

T

U

V

W

© 1993 ATI Technologies Inc.
Proprietary and Confidential

Perfecting the PC

Printed in Canada