4860 MotherBoard Reference Manual

Revision 1.7 May 25, 1991

Hauppauge Computer Works, Inc.
91 Cabot Court
Hauppauge, NY 11788 U.S.A.
Telephone: 516-434-1600
Fax: 516-434-3198

Table of Contents

Installing the 4860 MotherBoard

A quick guide to first-time installation

Note: When using the EISA BIOS version 1.00 and above, the Power-On self test program will display the message "Slot 00 error: .." during the first-time installation. This message can be ignored, because it alerts us to the fact that the EISA configuration has not yet been set.

Also, note that the BIOS Setup program uses the cursor keypad and the + - keys to move through the BIOS Setup and to set the BIOS options.

Step 1 - Install the 4860 MotherBoard, connect the power cables, the keyboard, speaker, reset connector, and install just the video adapter. The board as shipped from the factory has been jumpered and tested with both ISA and EISA disk controllers and the MotherBoard's serial and parallel ports have also been tested. The only jumper that might need to be changed is the **monitor type** jumper. The factory setting is color. It needs to be changed when using a monochrome monitor. The chapter entitled "Guide to Jumpers, .." describes the 4860 MotherBoard jumpers.

Step 2 - Turn on the power, and look for the Power-On self test messages. A CMOS RAM error message, and several other error messages will appear:

PRESS F1 TO CONTINUE OR CTRL-ALT-ESC TO ENTER SETUP

This is normal since the BIOS and the EISA configurations have not yet been set. One common problem is the "Configuration error" message which appears if the **monitor type** has not been

set correctly. Check the BIOS Setup program or the 4860 MotherBoard **monitor type** jumper.

Run the BIOS Setup program

Step 3 - Run **Setup** by typing Control-Alt-ESC (at the same time). Set the time, date (remember to use the + - keys) and floppy diskette types. Check to see that all of the memory that has been installed has been detected. As an example, on a 4MByte system the following configuration will be seen:

Base memory:	640K
Extended memory:	3072K
Expanded memory:	0K
Other memory:	384K
Total:	4096K

If using an MFM or IDE disk drive, set the hard **drive type** to the correct value.

If you are using an ESDI controller, set **drive type** to 1 or 2 (check with the disk controller manufacturer for the correct entry).

If using a SCSI controller, normally the **drive type** should be set to NONE. But be aware that some SCSI controllers will require the **drive type** to be set to 1.

If your MFM or IDE controller does not have a drive type that matches its parameters, there are two configurable drive types, 48 and 49, where you can set custom parameters. After entering either 48 or 49, use the right arrow key to get to the different parameters for your hard disk drive. Remember, since this is a custom hard disk drive type, write these parameters on a label that can be placed on the outside of your PC just in case the CMOS RAM is erased.

Hit the "PgDn" key (page down) to go to the next BIOS Setup screen. Here you can enable or disable the CPU cache, the BIOS cache and the Video cache. We recommend that all items be **ENABLED**. You can experiment with the effects on

performance by disabling the settings later on, but for now enable everything.

Leave the Default speed setting at **HIGH** and the Slow speed at **FAST**.

Hit **F10,F5** to save the CMOS Setup and to exit the Setup program.

Step 4 - Turn the power off, and install the floppy disk controller (or combination floppy/hard disk controller). Connect the cables to the floppies and make sure the power is also connected to the floppies.

Install a DOS disk (DOS 3.3 recommended) and turn the power back on. The Power-On self test program might display an "Error intitalizing hard disk 0" since the hard disk is not connected. This is o.k. Boot DOS from the floppy disk.

Run the EISA Configuration Utility

Step 5 - Now that DOS has been booted from the floppy, the EISA configuration must be set. The Hauppauge 4860 MotherBoard is provided with an EISA Configuration diskette that contains a program called **CF**. This diskette must be put in the floppy disk drive and the program **CF** should be run.

In systems without an EISA disk controller, the key sequence to set the EISA configuration is simple: **enter, enter,F10,S,X** and then choose **Save Configuration and Exit**. This will save the minimum EISA information required for use with non-EISA disk controllers.

In systems with an EISA I/O controller, before the **CF** program is run, the configuration program provided by the controller vendor must be copied onto the the Hauppauge EISA Configuration diskette. Then run **CF**, following instructions from your I/O controller manual. To save the EISA configuration once it has been set in the **CF** program, type **F10,S,X** and then choose **Save Configuration and Exit**.

Step 6 - Now that the BIOS has been set, and the EISA configuration has been saved in both the CMOS RAM and the diskette, you can connect the hard disk drive and boot.

If the battery that holds the CMOS RAM ever drains while the power is turned off, you must run the BIOS Setup, then run **CF**, using the old parameters saved on the diskette.

If you add or remove EISA I/O cards from the system, you need to run **CF** to modify the system configuration.

Installing i860/APX with the EISA 1.00 (and above) BIOS

When installing i860/APX, there are two extra steps in the setup. These steps partition the memory system between the i860 and the 486 processor.

The memory partitioning puts a gap in the memory space between the i860 and the 486. This gap is necessary so that 486 programs such as UNIX or OS/2 do not try to "grab" all of system memory for themselves.

To partition the memory, run the program **CFG4860**, which can be found on the EISA Configuration diskette. On the second screen, move the cursor with the down arrow key to the i860 memory option. Hit **8M** (to select 8MBytes of memory for the i860; from 4M to 16MBytes can be selected for the i860), and then down arrow once more. Now hit **F10, F5** to save this memory configuration.

Once the memory configuration is saved, it is necessary to do a hard reset (push the reset button or turn off the computer). After the hard reset, during the Power-On-Self-Test a message will appear stating **MEMORY SIZE ERROR**. This is o.k. since we have just changed the amount of memory seen by the 486 processor. Just hit **F1** to remove this message.

To check the memory configuration, run **INFO4860** (also on the EISA Configuration diskette), and observe the i860 Dedicated Memory setting. Due to a bug in INFO4860, please ignore the

message: i860 Not Present.

Note: It is not necessary to change the i860 memory size in the program "CF" once this has been set using CFG4860. Though you will find an entry for i860 memory size in CF, it is not necessary to change this entry.

Setting the memory size of APX

Note: The following procedure is necessary only for i860/APX versions 1.7 and below. With version 1.8, the memory size is automatically configured.

i860/APX has the memory configuration set during the APX installation. If any other memory configuration is used, the APX driver in UNIX must be told of this new configuration. There is a file called "kernel.cfg" which has been installed during the APX installation procedure which must be modified. The file is typically found in the **/usr/apx5/gen/src/ker860** directory under UNIX, and the **/usr/apx5/ker860** directory in i860/APX-DOS.

The two entries that need to be changed are:

ev_memory_size = 8

host_memory_size = 8

The 8 is an example showing 8MBytes. For example, if the 486 memory is 16MBytes, then change **host_memory_size** to 16. If the i860 memory size changes, then change the **ev_memory_size** entry. Note that this must be done after **installapx** is run.

Installing the PS/2 style mouse cable

The cable for the PS/2 style mouse connector consists of two pieces:

- Cable: 10 pin header to female DB-9
- Adaptor: DB-9 male to PS/2 style DIN

The 10 pin header is installed on the 4860 MotherBoard in con-

ector J23 (located near the Power Connector). See diagram 4.3 in the 4860 Reference manual for its location. Pin 1 of the cable is near the red stripe. **The mouse will not work if the cable is plugged in backwards**. Make sure the red stripe is towards pin 1 of J23.

The DB-9 female connector on the 10 pin cable is normally mounted on a bracket of the system chassis. The DB-9 to PS/2 adaptor is then attached to DB-9 female connector from the outside of the chassis.

Guide to jumpers and connectors

Note - * indicates default setting

PS8 Power supply connection

1 = /PWRGOOD from power supply
2 = +5 Volts
3 = +12 Volts
4 = -12 Volts
5,6 = GND

PS9 Power supply connection

1,2 = GND
3 = -5 Volts
4,5,6 = +5 Volts

PS10 Auxiliary power connection

Connects to PC Power & Cooling Model TURBO-450 power supply. Key is at pin 2.

1,2,3 = +5 Volts
4,5,6 = GND

J19 Speaker output connector

1 = +SPKR
2 = N.C.
33 = -SPKR
4 = +5 volts

J9 Reset switch connector

*Open = run
Short = reset

J11 486 Turbo connector

1 = VCC
2 = Turbo LED output (sink)
3,4,5 = N.C.
6 = TURBO Control
7 = GND

J21 External CMOS battery connector

1 = +VBB (minimum +4.5v, max 6v)
2 = N.C.
3, 4 = -VBB (normally ground)

J23 PS/2 compatible mouse interface connector

Note: Requires special cable. See diagaram in Chapter 3.
1 = Clock
2 = Data
4 = Signal Ground
5 = +5 Volts
3,6,7,8,9,10 = Shield Ground

J1, J3-J8 EISA 8/16 bit connectors

(J1, J3-J6 and J8 are EISA MASTER capable,
J7 will not allow EISA bus masters)

J2 ISA compatible 8-bit connector

J10 64-Bit expansion connector

(to be supplied in a future revision of this manual)

J20 Keylock/Power LED connector

1 = GND
2 = KEYLOCK
3 = GND
4 = [KEY]
5 = +5 Volts

J22 **DIN-5 Keyboard connector**

1 = CLOCK 4 = GND
2 = DATA 5 = +5 Volts
3 = N.C.

JP1 **i860 timer interrupt jumper**

*2-3 (61.7911 msec)
1-2 (30.8955 msec)

JP3 **Mouse Interrupt (IRQ12) jumper**

*1-2 = No interrupt generated
2-3 = Enable mouse interrupt

JP11 **2x4 Keyboard configuration header**

Do not change unless instructed for a Keyboard BIOS upgrade

JP12 **Future i860 mode support**

*Closed = DO NOT INSTALL

JP13 **Delayed ADS jumper**

*2-3 = DO NOT CHANGE

JP14 **25/33MHz Jumper**

Short = 25MHz
Open = 33MHz

JP15 **Turbocache Module write protect**

Open = write protected item cached by 486 & Turbocache
Short = write protected item cached by Turbocache only

JP2 **Turbo Cache module**

*In = Turbo cache not installed
Out = Turbo cache installed

SW1 **Monitor select/Burn-in test jumper**

*Open selects Monochrome
1-2 selects burn-in test
2-3 selects color monitor

Parallel I/O port jumpers and connector

JP8 Parallel port address select jumper

1-2 = LPT1 (port 03BCh)

*2-3 = LPT2 (port 0378h)

JP9 Parallel port interrupt enable jumper

*Short = LPT1/2 generates IRQ7

Open = No interrupt generated

J15 Parallel port connector

NOTE: Cable straight from 2x13 header to DB25 connector

NOTE: Pin 1 is LEFT-BOTTOM and goes to pin 1 of DB25

1 = /STB	2 = /AFD
3 = D0	4 = /ERR
5 = D1	6 = /INIT
7 = D2	8 = /SLIN
9 = D3	10 = GND
11 = D4	12 = GND
13 = D5	14 = GND
15 = D6	16 = GND
17 = D7	18 = GND
19 = /ACK	20 = GND
21 = BUSY	22 = GND
23 = PE	24 = GND
25 = SLCT	26 = GND (not used on DB25 connector)

Serial I/O port jumpers and connectors

JP4 Serial port A address select jumper

*1-2 = Port A is COM1

2-3 = Port A is COM3

JP5 Serial port A interrupt enable jumper

*Short = COM1/3 generate IRQ4

Open = No interrupt generated

JP6 Serial port B address select jumper

*1-2 = Port B is COM2

2-3 = Port B is COM4

JP7 Serial Port B interrupt enable jumper

*Short = COM2/4 generate IRQ3

Open = No interrupt generated

J13 , J14 COM1/3 and COM2/4 connectors

NOTE: Cable straight from 2x5 header to DB9 connector

NOTE: Pin 1 is LEFT-BOTTOM and goes to pin 1 of DB9

1= DCD(I)	2= DSR(I)
3= RxD(I)	4= RTS(O)
5= TxD(O)	6= CTS(I)
7= DTR(O)	8= RI(I)
9= GND	10= GND (not used on DB9 connector)

4860 MotherBoard Revision C changes:

JP18 64-bit slot write select

1-2 CPUWR#

* 2-3 BCPUWR#

JP16 Reserved

Must be left open

JP17 Reserved

Must be left open

JP10 No longer needed

Adding additional memory

The 4860 EISA MotherBoard requires the RAM to be loaded in **pairs** of 36-bit SIMMs. The SIMM modules can be 2MByte, 4MByte or 8MBytes per module. The modules may be mixed and matched as long as they are paired.

We recommend using 70nsec SIMM modules for the 4860-33MHz MotherBoard. Some (but not all) qualified modules are:

2 MByte:	
Toshiba	THM365120S-70
Micron	MT8C36512DM-7
Samsung	KMM536512A-7
4 Mbyte:	
Toshiba	THM361020S-70
OKI	MSC2355-70YS12
NEC	MC421000A36BH70
8 MByte:	
Toshiba	THM362020S-70
OKI	MSC2356-70YS18
NEC	MC422000A36BH70

In the default BIOS configuration, RAM is populated in sockets **U20** and **U24**. RAM grows upwards from this point using sockets **U21** and **U25**, then **U22** and **U26**, and finally **U23** and **U27**. A sample configuration might be:

U20, U24	512K x 36 (2MB SIMM's)	4MBytes
U21, U25	1M x 36 (4MB SIMM's)	8MBytes
U22, U26	512K x 36 (2MB SIMM's)	4MBytes
U23, U27	2M x 36 (8MB SIMM's)	16MBytes

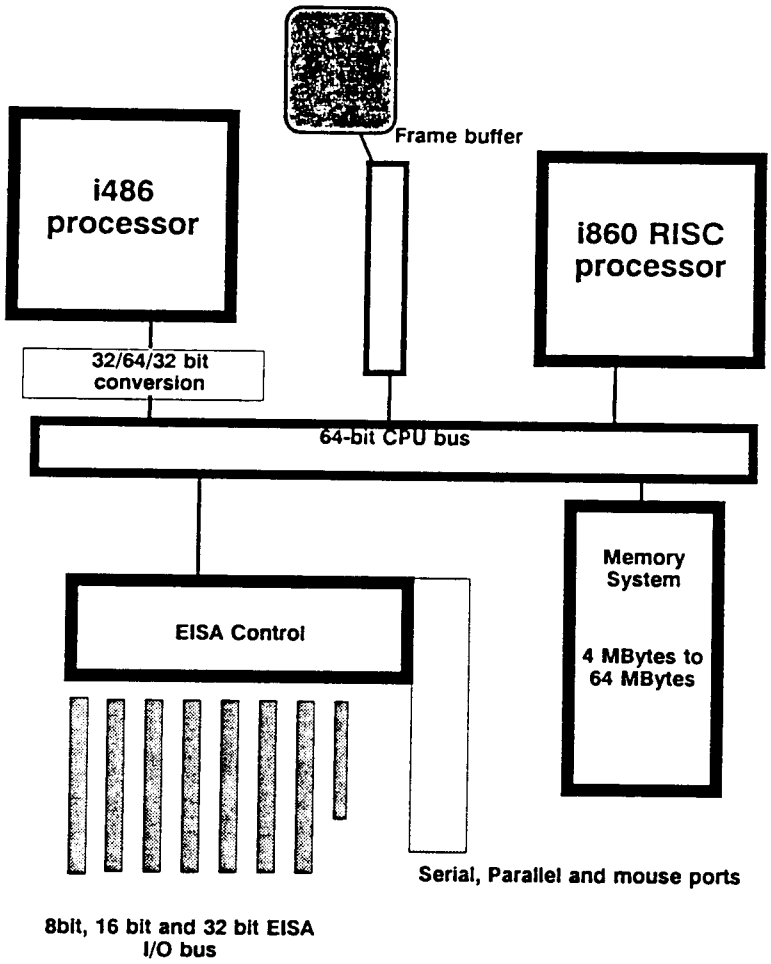
This configuration contains 32 Megabytes of memory. Any (or all) of this memory can be allocated to either the 486 or 860 by running a simple re-mapping program (see Chapter 1 for instructions on running **CFG4860**).

The following chart shows some sample memory configurations. Three configurations are shown; 4MB, 16MB and 32MB:

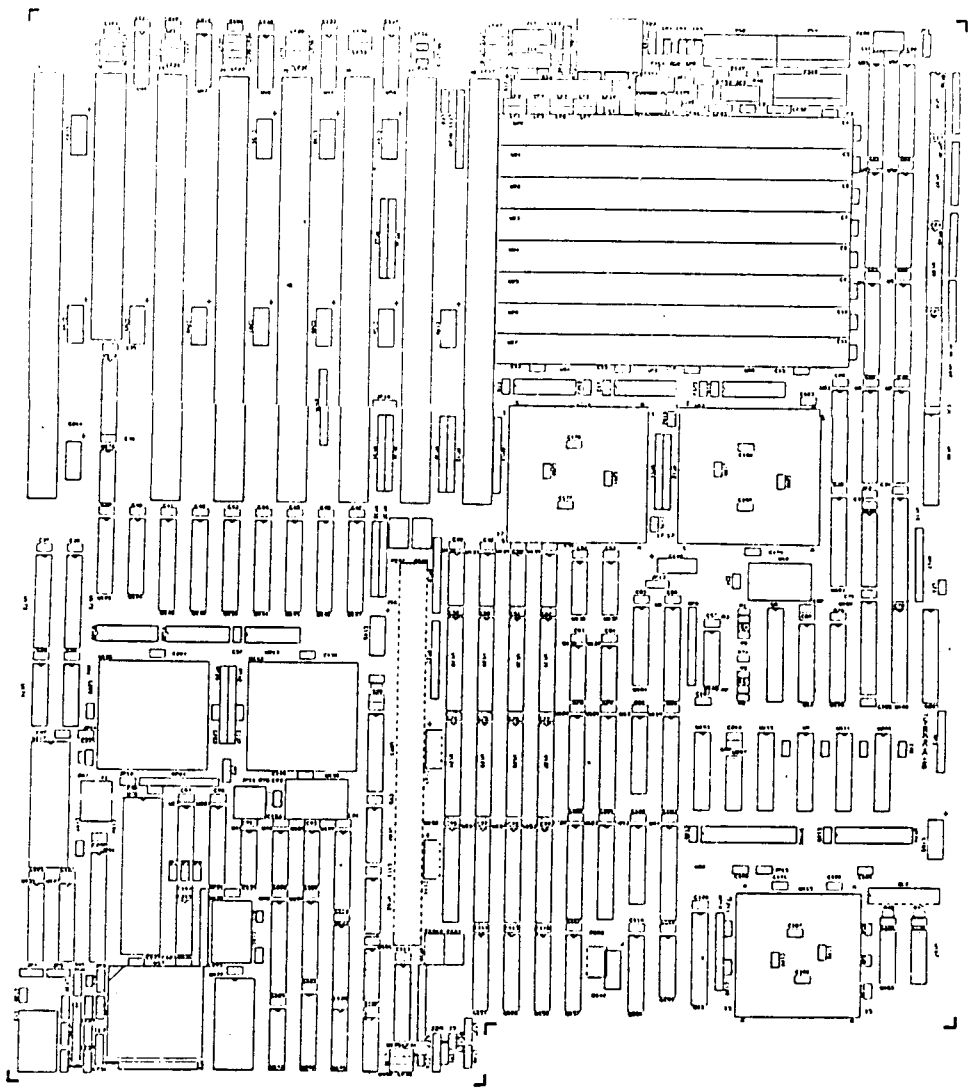
Socket	4MBytes	16MBytes	32 MBytes
U20	2MB SIMM	4MB SIMM	2MB SIMM
U21		4MB SIMM	4MB SIMM
U22			2MB SIMM
U23			8MB SIMM
U24	2MB SIMM	4MB SIMM	2MB SIMM
U25		4MB SIMM	4MB SIMM
U26			2MB SIMM
U27			8MB SIMM

4860 EISA MotherBoard Diagrams

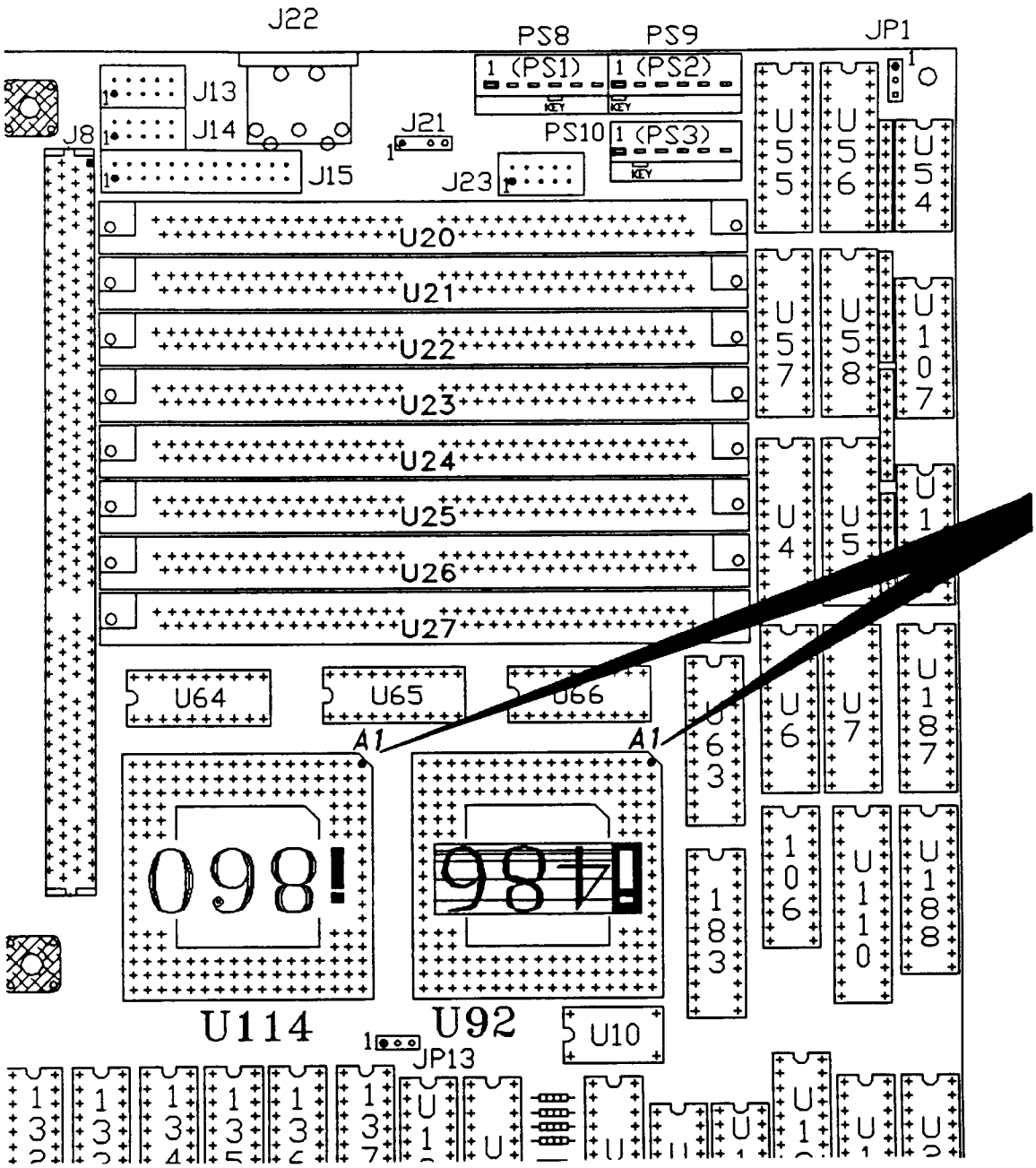
4860 MotherBoard Block Diagram



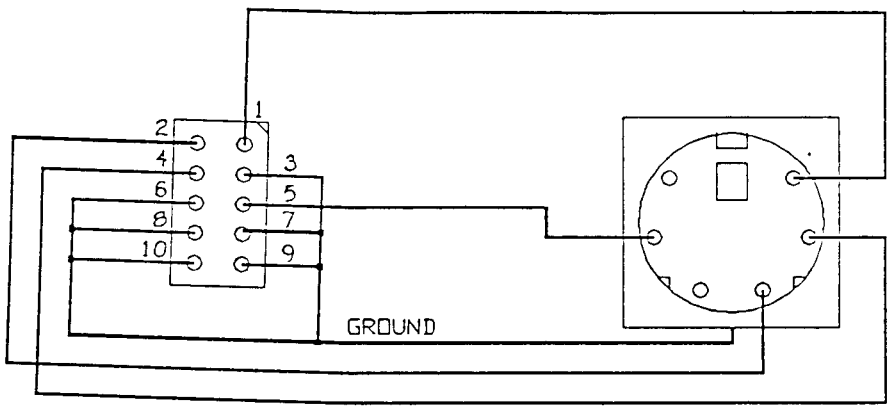
4860 EISA MotherBoard layout



486 /860 Processor orientations, SIMM connectors

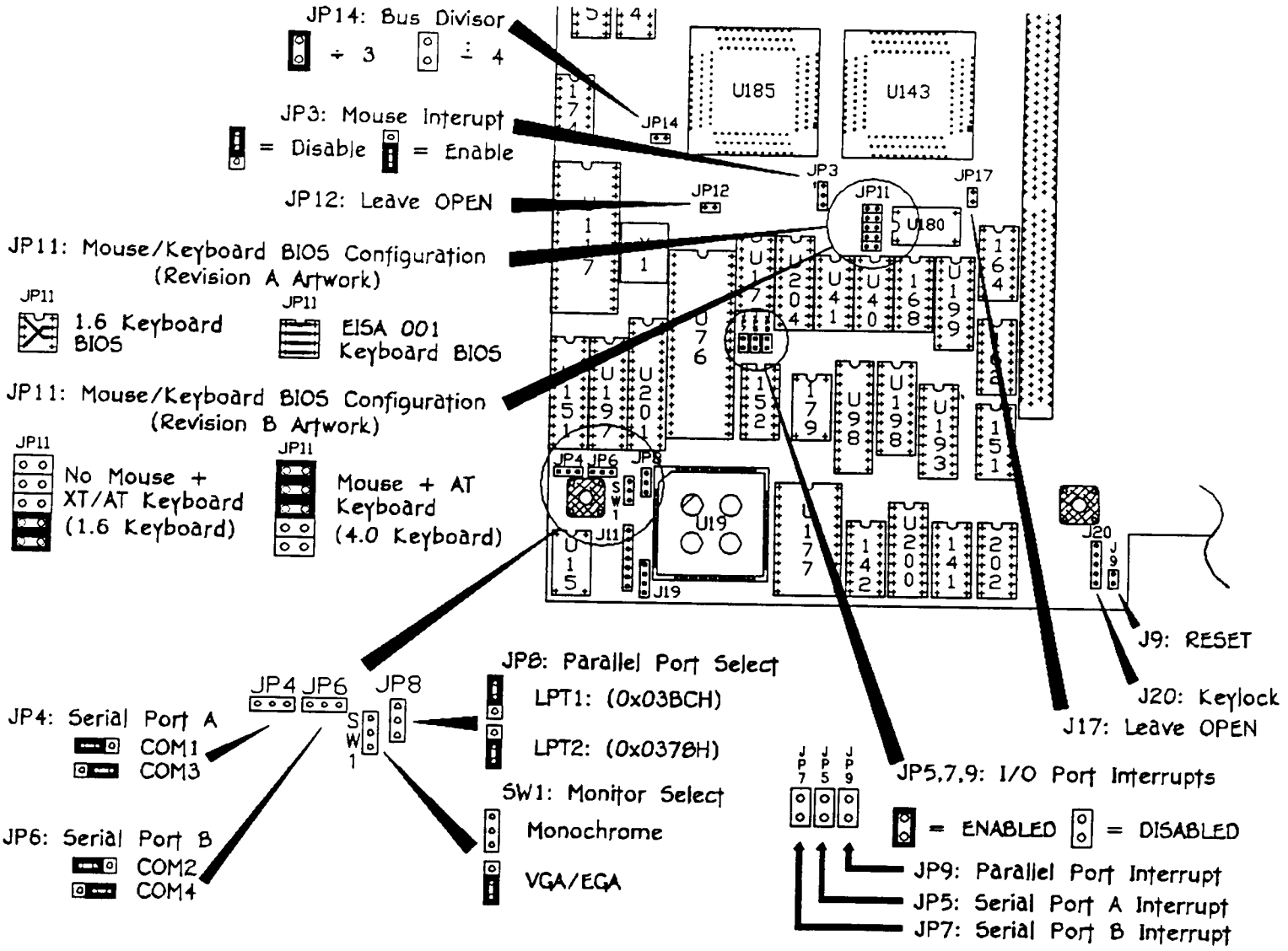


Mouse connector



Mouse Connector
(As seen from back
of computer)

Jumper locations



EISA Configuration Utility

Trademark Information

Configuration Utility User's Guide Version 1.10. Micro Computer Systems, Inc., Hauppauge Computer Works, Inc.

NOTICE: The information in this guide is subject to change without notice.

NEITHER MICRO COMPUTER SYSTEMS, INC. OR HAUPPAUGE COMPUTER WORKS, INC. SHALL NOT BE LIABLE FOR TECHNICAL OR EDITORIAL OMISSIONS MADE HEREIN; NOR FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL.

Introduction to the EISA Configuration Utility

The Extended Industry Standard Architecture Configuration Utility is a software utility for the configuration of EISA com-

puters. Delivered with all EISA machines, the EISA Configuration Utility automatically generates conflict-free configuration information for the system, and provides information to the user for the correct setting of switches and jumpers on older-design ISA cards.

The Problem

In the ISA architecture, one of the biggest challenges faced by a user is configuring the computer. The computer must be configured when it is initially set up and any time hardware is added to or removed from the system. This often requires reading several manuals to find the information needed to correctly set switches and jumpers as well as learning new concepts such as I/O addresses and interrupt levels. Once the user has configured the computer, the only way to know if it is configured correctly is to plug all the boards in and see if everything works. If not, there is usually little to assist the user in resolving the problem.

The Solution

In the new EISA architecture, option boards are designed without switches or jumpers. Configuration is achieved through a series of initialization commands that are stored in nonvolatile memory. The EISA Configuration Utility determines the configuration, creates the initialization commands, and ensures that the configuration is correct. If a conflict free configuration is not possible, the user is notified immediately.

At system power-up time, the EISA system ROM downloads system initialization information from nonvolatile memory to the slot-specific I/O addresses for each EISA board to declare system resource use and other manufacturer-defined options. For EISA boards, this port initialization process is automated, and it replaces the switch and jumper setting method used for ISA boards.

One advantage of EISA is the ability to accept existing ISA boards. The expansion bus of an EISA computer is defined to be a superset of the Industry Standard Architecture and main-

tains full upward compatibility for option boards originally designed for 8 and 16 bit ISA computers. Boards designed for the ISA architechture containing switches and jumpers can coexist with software programmable boards in an EISA system. In addition to generating initialization information for EISA boards, the EISA Configuration Utility is able to determine the switch settings for any ISA board in a system if a corresponding ISA CFG file is supplied.

Configuration (CFG) and Extension (OVL) Files

Each option board in an EISA system has a corresponding configuration file (CFG) that describes the characteristics and the required system resources of that board. The EISA Configuration Utility uses the information from the CFG files to create a conflict-free configuration.

Despite the extensive flexibility of the format of the CFG file, there may be times when the Configuration Utility can not support all possible configuration options. Therefore, executable code in the form of an OVL file can be integrated into the Configuration Utility's processing. This allows further customization of the configuration process for any unusual board specific needs, such as determining system equipment, fixed disks, coprocessors, floppy drives, system passwords, and other special devices.

System Configuration Information Files (SCI)

The EISA Configuration Utility has the ability to create configuration information files which can later be used on another computer. These files are named with the SCI extension. In addition, the SYSTEM.SCI file is maintained by the Configuration Utility as a backup for the computer's EISA nonvolatile memory.

When the Configuration Utility is first executed, it determines if the nonvolatile memory information is valid. If the nonvolatile memory information is invalid, the configuration may be restored by loading an SCI file from the configuration disk or by manually selecting CFG files and assigning them to slots.

Starting the utility

This utility runs on any ISA or EISA computer using MS-DOS version 3.2 or later. A minimum of 640 Kbytes of memory and a 1.2 megabyte diskette drive are required. A mouse is also recommended to move the cursor and to select options within the utility.

To start the utility, insert the System Configuration diskette in drive A. Then, turn on your computer. If your computer is already on, hold down [Ctrl] and [Alt], and press [Del] at the same time. A logo screen will be displayed. Press any key to display a welcome screen. Press [Enter] to leave the welcome screen and display the main menu. Refer to Chapter 2, "Command Reference," for instructions on using the menu and a description of the menu selections.

Command Reference

Using the main menu

There are two ways to choose a menu selection from the main menu: using the keyboard and using a mouse. To choose a menu selection using the keyboard, use the up and down arrow keys to position the cursor on the desired menu selection and press [Enter].

To choose a menu selection using a mouse, position the mouse cursor over the desired selection on the menu and click the left button on the mouse.

Using the submenus

Five pull-down menus can be used when you choose the Configure Computer - Advanced Method selection (for menu options see MAIN MENU SELECTIONS section in this guide). These menus are listed in the menu bar at the top of the screen. There are three ways to use the pull-down menus:

Using the keyboard

To make menu selections using the keyboard, follow these

steps:

1. Press the [F10] key to place the cursor on the menu bar.

2. Use the [left arrow] and [right arrow] keys to position the cursor on the desired pull-down menu name.

3. Press the [Enter] key to display the pull-down menu.

4. Use the [up arrow] and [down arrow] keys to position the cursor on the desired pull-down menu item.

5. Press the [Enter] key to execute the pull-down menu item. If the menu item ends in an ellipsis (...), a dialogue box will be displayed to prompt you for further information; otherwise, the menu item will be executed directly.

Using enhanced letters

Each pull-down menu listed on the menu bar and each item on the pull-down menus have a single enhanced letter. To make a selection using enhanced letters, follow these steps:

1. Press and hold down the [Alt] key while pressing the enhanced letter for the desired pull-down menu. The pull-down menu is displayed.

2. Press the enhanced letter that corresponds to the desired menu item to execute the menu item. If the menu item ends in an ellipsis (...), a dialogue box will be displayed to prompt you for further information; otherwise, the menu item will be executed directly.

Using a mouse

To make menu selections using a mouse, follow these steps:

1. Place the mouse cursor over the desired pull-down menu name on the menu bar and click the left button on the mouse. The pull-down menu is displayed.

2. Place the mouse cursor over the desired pull-down menu item and click the left button on the mouse to execute the menu item. If the menu item ends in an ellipsis (...), a dialogue box will be displayed to prompt you for further information; otherwise, the menu item will be executed directly.

Using dialogue boxes

When you select a menu item that ends in an ellipsis (...), a dialogue box will be displayed to prompt you for further information. There are two ways to use a dialogue box:

Using dialogue boxes from the keyboard

To move within a dialogue box using the keyboard, follow these guidelines:

1. Press [Tab] to move to the next field or area.

2. Press [Shift + Tab] to move to the previous field or area.

3. Use the arrow keys to move between items in a list.

4. Press the space bar to turn a check box on or off.

5. Press the [Enter] key to select.

6. Press [Esc] to cancel the dialogue box.

Using dialogue boxes with a mouse

To move within a dialogue box with a mouse, follow these steps:

1. Place the mouse cursor over the desired choice in the dialogue box and click the left button.

2. Place the mouse cursor on a check box and click the left button to turn the check box on and off.

3. Place the mouse cursor on a button and click the left mouse button.

Using screens

When your configuration is displayed in an overview or a detailed view, use the following methods to move the cursor:

Using the keyboard

To move the cursor on the screen using the keyboard, follow these guidelines:

1. Press [Tab] to move to the next option.

2. Press [Shift + Tab] to move to the previous option.

3. Press [Ctrl + Home] to move to the beginning of the information.

4. Press [Page Down] to move down one screen.

5. Press the [down arrow] key to move down one line.

6. Press [Page Up] to move up one screen.

7. Press the [up arrow] key to move up one line.

8. Press [Ctrl + End] to move to the end of the information.

Using a mouse

To move the cursor on the screen using a mouse, follow these guidelines:

1. Place the mouse cursor over the desired area on the screen and click the left button.

2. If a scroll bar is displayed on the right side of the screen, place the mouse cursor over the arrow symbol at the top or bottom of the scroll bar and press the left button.

Getting help

An information line is displayed on the last line of your screen. This line contains a list of the keys you can use.

You can get help about a selection on a menu or any object on a screen by pressing the [F1] key while your cursor is on the selection or object. You can also display help from the help topics index by selecting Help topics from the Help pull-down menu or by pressing [Shift + F1].

For a complete list of key combinations, press [Shift + F1] to display the help topics index. Then use the arrow keys to select "Using the keyboard" and press [Enter].

Main Menu Selections

Learn about configuring your computer

This selection will display an overview of how to use this utility to configure your computer.

Configure computer

When you choose this selection, a menu is displayed with the following selections:

Copy configuration (CFG) files

Configure computer - basic method

Configure computer - advanced method

Return to the main menu

Copy configuration (CFG) files

This selection copies configuration (CFG) files from an OPTION CONFIGURATION diskette or the Configuration File Library to your System Configuration diskette. You need to copy a CFG file for each board or option you plan to install BEFORE you configure your computer.

Configure computer - basic method

This selection will guide you through the procedure for adding boards and options to your computer. When you choose this

selection, a graphical overview of the boards and options installed in your computer is displayed on the screen. When there are no resource conflicts, the configuration information is saved in a system configuration information (SCI) file. If you need to change functions, edit resources or perform advanced tasks, use the advanced method.

Before you return to the main menu, a list of the switch, jumper, and software settings which you need to change for ISA boards can be displayed. Either write them down or print them if you have a parallel printer attached to your computer. If you do not have a CFG file for an option or board you want to install, use the instructions provided with the option to install it.

Configure computer - advanced method

When you choose this selection, a graphical overview of the boards and options installed in your computer is displayed on the screen. A menu bar is displayed at the top of the screen, which you use to access pull-down menus.

This selection allows you to enter configuration information about your computer through the use of pull-down menus (see MENU BAR SELECTIONS for pull down menu option functions). As you enter information, the entries are verified (when Auto Verify is on (default)). If any resource conflicts are found, you are allowed to correct the errors by changing function and resource choices. When there are no resource conflicts, the configuration information is saved in a system configuration information (SCI) file.

Menu Bar Selections

Menu Bar

SelectionPull-Down Menu

SelectionFunction

SystemNew Creates a new configuration and system configuration information (SCI) file for a computer (available in non-target

modeling mode only).

Open Opens an existing system configuration information(SCI) file for editing.

Save As Makes a backup copy of the current configuration choices in a specified system configuration (SCI) file.

Print Prints configuration information about the current option or the entire configuration on a printer.

Verify Verifies that the computer is correctly configured (the configuration is free of system resource conflicts.)

Exit Prompts to either view the settings of the boards and options, save the configuration and exit, or exit without saving any changes. If the configuration has changed and is saved, the computer will reboot instead of exiting to the menu.

EditAdd Adds a selected board or option to the current configuration.

Move Selects the current board and moves it to a selected available slot.

Remove Selects the current board and removes it from the current configuration.

Change Function Selects the current function and allows you to change the function choice. Only available when the detailed view is active.

Change Resource Selects the current function and allows you to change the system resources for the function. Only available when the detailed view is active.

Revert to Saved Sets all choices for the current board or the entire configuration to the last saved choices.

Reset to Defaults Sets all choices for the current board or the

entire configuration to the manufacturer's default choices.

Lock Secures all choices for the current board or the entire configuration to the current selections.

Unlock Unlocks all choices for the current board or the entire configuration.

ViewOverview When selected, displays a general overview of the configuration.

Detailed by Slot When selected, displays a detailed view of the configuration sorted by slot.

Detailed by Type When selected, displays a detailed view of the configuration sorted by type of function.

Switch and Jumper Settings Selects the current board or option and displays information about its switches and jumpers.

Software Parameters Selects the current board or option and displays information about software drivers.

Connections Selects the current board or option and displays information about its external cable connections.

Board Specifications Selects the current board or option and displays information about identification and physical characteristics.

Resources Selects the current board or function and displays system resource summary information.

SettingsAuto Verify When selected, your computer's configuration is checked for resource conflicts each time you changethe configuration. Selecting Auto Verify turns Manual Verify off.

Manual Verify When selected, your computer's configuration is not checked for resource conflicts. You must select Verify from the System pull-down menu to check your computer's configura-

tion. Selecting Manual Verify turns Auto Verify off.

HelpHelp Topics Displays an index of help topics which can be displayed on your screen.

Help Displays help information about the currently selected board, option, or function. This menu item performs the same function as pressing the [F1] key.

How to Use Keys Displays a list of the key sequences which can be used.

How to Use Help Displays information about using help.

Copyright Information Displays copyright information about this utility.

Before you return to the main menu, a list of the switch, jumper, and software settings, which you need to change for ISA boards, can be displayed. Either write them down or print them if you have a parallel printer attached to your computer. If you do not have a CFG file for an option or board you want to install, use the instructions provided with the option to install it.

Return to main menu

This selection will return you to the main menu.

Set date and time

This selection allows you to set your computer's date and time. Once you set the date and time, your computer will keep track of it, even if the power is turned off.

Exit from this utility

This selection exits the utility. The system will reboot. If you do not have an operating system installed on your fixed disk, replace the System Configuration diskette in the diskette drive with your operating system diskette.

Starting the Configuration Utility from a fixed disk

It is recommended that you start this utility from the System Configuration diskette. However, if you have MS-DOS installed on your fixed disk, you can start the utility from your fixed disk. Starting the utility from your fixed disk allows you to customize the utility through the use of command line parameters.

Installing the Utility on a Fixed Disk

Before you can start the utility from your fixed disk, you must copy the utility files to your fixed disk. To copy the files, insert the System Configuration diskette in drive A and enter:

COPY A:*.* [d:][path]

where:

d: is the fixed drive that will contain the utility files

path is the path on the fixed drive that will contain the utility files

Starting the Utility

To run the utility, type the following command followed by [Enter]. Note that items in brackets are optional and only the information within the brackets is entered.

[d:][path]SD [/A] [/B] [/H] [/K] [/M]

where:

d: is the drive which contains the utility files

path is the path to the utility files

/A Expanded mode. Provides an expanded set of menus with additional functionality. This mode may also be activated by pressing CTRL-A instead of [Enter] when the "Welcome" screen is displayed.

/B BIOS video mode. This parameter causes all screens to be displayed using BIOS Int 10h calls. This parameter should be used on computers with non-standard displays. The default mode is to write directly to video memory.

/H High resolution display. If you have an EGA monitor, the utility will be displayed in 43-line mode. If you have a VGA monitor, the utility will be displayed in 50-line mode. When this parameter is not used, 25 lines will be displayed on your screen.

/K Keyboard only mode. If this parameter is used, the computer will not support the use of a mouse device even if one is present. The default is to support a mouse if its driver is loaded.

/M Monochrome display mode. Color not used.

When the utility begins, a logo screen will be displayed. Press any key to display a "Welcome" screen. Press[Enter] to leave the "Welcome" screen and display the main menu. Refer to "Command Reference," for instructions on using the menu and a description of the menu selections.

Starting Configure Computer selection from a fixed disk

You can bypass the main menu and start the Configure Computer selection directly from MS-DOS. This allows you to use command line parameters to customize the operation of the utility. You must have MS-DOS and this utility installed on your fixed disk.

Use the following format to type the command that starts this selection from MS-DOS. Note that items in brackets are optional and only the information within the brackets is entered.

[d:][path]CF [/B] [/E] [/F] [/H] [/K] [/M] [/N] [/T]

where:
d: is the drive which contains the utility files

path is the path to the utility files

/B BIOS video mode. This parameter causes all screens to be displayed using BIOS Int 10h calls. This parameter should be used on computers with non-standard displays. The default mode is to write directly to video memory.

/E Easy configuration operation. Starts the utility in Basic Method.

/F Fast configuration operation. This is the automatic configuration mode. The utility will determine the boards and options and configure the system without user assistance.

/H High resolution display. If you have an EGA monitor, the utility will be displayed in 43-line mode. If you have a VGA monitor, the utility will be displayed in 50-line mode. When this parameter is not used, 25 lines will be displayed on your screen.

/M Monochrome display.

/K Keyboard only mode. If this parameter is used, the computer will not support the use of a mouse device even if one is present. The default is to support a mouse if its driver is loaded in memory.

/N Non-target modeling mode. This parameter runs the utility in non-target modeling mode. When you run this utility, the configuration which is stored in the system configuration information (SCI) file named SYSTEM.SCI is displayed. When the configuration is saved, it is normally saved to the SYSTEM.SCI file. However, with non-target modeling mode, you can create an SCI file for a computer other than the one on which you are running the utility, create multiple SCI files, and create one SCI file for multiple computers.

/T Detailed view by slot. This parameter causes the default view to be detailed by slot instead of overview.

i860/APX Attached Processor Executive

Overview

860/APX is the operating system to support i860 based applications. It requires a host operating system such as UNIX or DOS to run on the 486 processor while the 860/APX Executive runs at the same time on the i860 processor.

i860/APX lets the 486 host operating system perform all system I/O, including reading and writing disk files and displaying data on the video screen. The i860 RISC processor is free to run its programs without having to be concerned with the details of the host operating system. This means that programs developed for the i860 on one operating system (DOS for example) can be run on any of the other APX supported operating systems (SCO UNIX for example), **without modification or recompiling!** This is called "binary portability".

Once i860/APX is loaded and is running on the i860 processor, standard C and Fortran programs can be compiled with i860 compilers and executed using i860 tools. A reference to some of the available i860 tools is at the end of this chapter.

System support requirements to load i860/APX are:

- Hauppauge 4860 MotherBoard with both 486 and 860 processors and 4MBytes minimum for the 486 and 4MBytes minimum for the 860.
- For i860/APX-UNIX: ATT UNIX V version 3.2.2, ATT UNIX V version 4.0, Interactive UNIX V version 3.2.2 or SCO Open Desktop ver. 3.2.2 installed on the hard disk
- For i860/APX-DOS: DOS 3.3, 4.01 or 5.0 installed on a hard disk
- 1.2 MByte floppy

See Chapter 1 for configuring the memory system for use with i860/APX. Please read the sections titled **Installing i860/APX with the EISA BIOS** and **Setting the memory size of APX**.

Installing i860/APX-DOS

To install the i860/APX-DOS software from your Hauppauge APX diskette, insert your APX floppy in drive A; and type:

```
mkdir c:\apx5

xcopy  a:  c:\apx5\*.*  /s
```

i860/APX-DOS is loaded in two parts: a DOS driver which is loaded by CONFIG.SYS, and the i860/APX Executive which can be loaded from the command prompt (c:\).

The DOS driver is called N860.SYS. This driver uses the protected mode of the 486 processor, and so it should be loaded before other protected mode drivers are loaded. Modify your CONFIG.SYS file by adding:

```
device = c:\apx5\N860.SYS
```

The APX system configuration is in the **KER860** subdirectory in a file called **kernel.cfg** The two parameters that should be modified are **ev_memory_size** (the amount of 860 memory that is configured) and **host_memory_size** (the amount of 486 memory that is configured). Once again, see Chapter 1 for configuring the memory system for use with i860/APX. Please read the sections titled **Installing i860/APX with the EISA BIOS** and **Setting the memory size of APX**.

The i860/APX Executive is loaded by moving to the **c:\apx** directory, and then typing:

```
boot860
```

This needs to be done twice due to a bug in i860/APX. After the i860/APX driver is loaded, check to see if APX is functional

by typing:

adm860 -s

The correct status should include the number of hours, minutes and seconds that APX has been loaded. To run a simple test program, type:

run860 hilbert

This demo program does a matrix multiply, and takes about 5 seconds to run. The speed of this complex math program is equivalent to running on the original CRAY 1 supercomputer!

Please note: the i860 program debugger, normally part of in the **run860** command, is currently not functional under i860/APX-DOS.

A seperate directory, called **\usr\pgi**, has been setup for the Portland Group i860 compilers. The i860/APX-DOS diskette does not come with these compilers, but the directory is set up with the correct script files to run the Portland Group compilers should you have them. Also, to support the i860 compilers the i860/APX-DOS distribution diskette comes with **TMP** directories. These can be removed if you are not planning to use the i860 compilers.

Installing 860/APX under UNIX V

Once UNIX V has been loaded on the hard disk and can be booted, 860/APX can be loaded from a floppy disk.

After booting the UNIX operating system on the 486 processor, the user must log in as **root**. Make sure the current directory is the root directory and that the 860/APX load floppy is in the 1.2MByte floppy disk drive.

If the primary floppy disk drive is a 1.2MByte floppy disk drive, the following commands will load i860/APX onto the hard disk:

```
# mkdir /apx5
# mount -r /dev/dsk/f05ht /apx5
# /apx5/installapx
# shutdown -g0
```

The shutdown command will allow the re-booting of UNIX so that the 860/APX driver will be loaded. When the system issues the command "Reboot the system now", reboot by hitting CTRL-ALT-DEL. The loading process takes about 15 minutes.

Note: If drive B: is the 1.2MByte floppy, then substitute the following command in the second step:

```
# mount -r /dev/dsk/f15ht /apx5
```

After the installation is complete, all 860/APX programs, utilities and demo programs are put into a directory called:

```
/usr/apx5
```

There are several subdirectories that hold binary files, demo programs and programs needed during the APX installation process.

At this point UNIX will be rebooted and the 860/APX drivers will be loaded. The following message shows that the 860/APX driver has been loaded into the UNIX operating system:

```
i860 Driver Release 1.0 found
```

After logging in as root, the PATH should be set to the /usr/apx5/bin directory by either adding /usr/apx5/bin to your current path, editing the **.profile** file or by typing:

```
PATH=PATH$:/usr/apx5/bin
```

You can now load 860/APX onto the i860 processor by execut-

ing the following commands:

```
# boot860&

# boot860&
```

- **Note:** boot860 needs tobe run twice on initial powerup. This is due to a bug in 860/APX, which will be fixed in the next release of 860/APX.
- **Note:** the ampersand (&) at the end of boot860 allows UNIX to continue even if the boot860 command fails. If there is a hardware problem, you are still in UNIX and can gracefully shut down UNIX without having to push the reset button.

This loads the Attached Processor Executive on the 4860, and prepares for the execution of i860 programs. If 860/APX does not load at this point, it means the i860 processor is not functioning properly. The reasons for this might be:

- i860 processor was not installed in the Hauppauge 860 MotherBoard, or was not plugged in all the way.
- The SIMM modules for the i860 were not plugged in, or are not functioning properly.

After the 860/APX has been loaded onto the i860, you can check the status of APX by typing:

```
# adm860 -s&
```

To run an i860 program type:

```
# run860 xxx
```

where xxx is the name of a program that has been compiled with an i860 compiler. A demo program called hilbert has been included and can be run by the following command:

```
# run860 /usr/apx5/tests/hilbert
```

To debug this program, the following command can be used. Please refer to the chapter called "i860 Assembler, Linker and Debugger" for a list of debugger commands:

```
# run860 -d /usr/apx5/tests/hilbert
```

To see the source code of this program type:

```
# cat /usr/apx5/tests/hilbert.c
```

There are other programs which have been compiled to run on the i860 processor using a i860C compiler from Metaware. These include hilbert, hil11 and mandel (which requires a VGA adaptor). All of these programs can be found in the directory:

```
/usr/apx5/tests
```

Programs included with i860/APX

boot860: loads the 860/APX Executive into the i860's memory space and then has the i860 Exective run its startup code. Once boot860 has been run, other programs can be loaded into RAM by invoking **run860** for execution by the i860 processor.

adm860: the i860 System Administrator. Shows the status of programs running on the i860, and allows these programs to be shut down.

To see the i860 program status type:

```
adm860 -s&
```

Other options can be found by typing adm860, which will show all options.

run860: loads an i860 program from the disk into the i860's memory space and starts the i860 processor executing it. If the run860 command is followed by **-d**, a debug session is started. If the command line is appended with an ampersand (&), i860 status can be checked by running adm860:

run860 xxx&

adm860 -s&

reset: resets the i860. This program is useful if the i860 program crashes and it is necessary to reset the i860 without reseting the i486 processor.

i860 development tools

Compilers

Metaware: High C i860

- tele: 408-429-6382 fax: 408-429-9273

Green Hills: C-i860, C++, Fortran

- tele: 805-965-6044 fax: 805-965-6343

Portland Group: C, Fortran Compiler

- tele: 503-682-2806 fax: 503-682-2637

Other i860 tools

Hauppauge Computer Works, Inc.: i860ASM, i860Linker

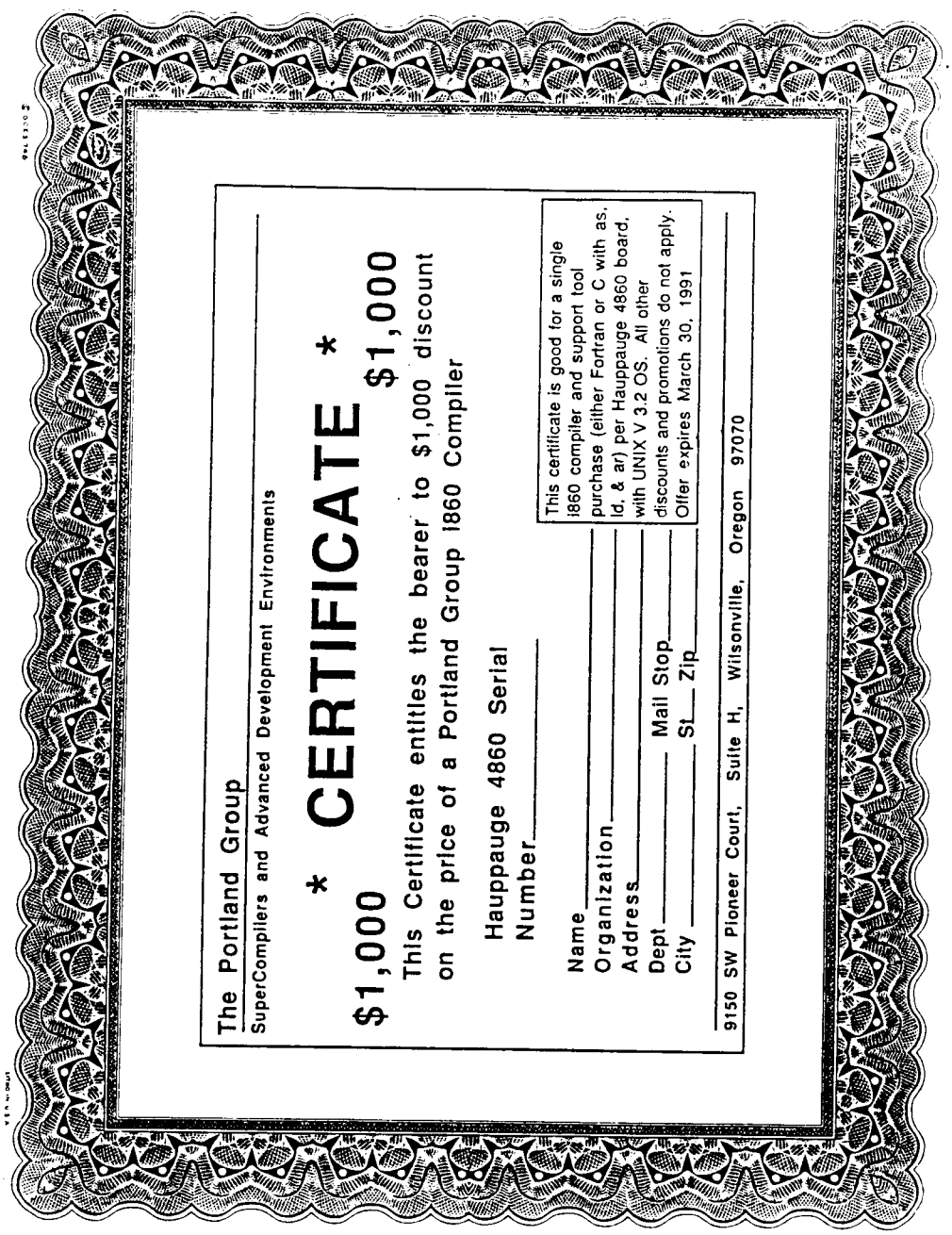
Portland Group: Debugger; GUI for X-windows

Profiler/Monitor

Discount coupons for compilers

On the next page are coupons good for discounts on compilers for the i860. They are provided by the vendors of the compilers, and should be redeemed directly with them.

Portland Group discount coupon



The Portland Group

SuperCompilers and Advanced Development Environments

* CERTIFICATE *

$1,000 $1,000

This Certificate entitles the bearer to $1,000 discount on the price of a Portland Group i860 Compiler

Hauppauge 4860 Serial Number_____

Name_____

Organization_____

Address_____

Dept_____ Mail Stop_____

City_____ St___ Zip_____

This certificate is good for a single i860 compiler and support tool purchase (either Fortran or C with as, ld, & ar) per Hauppauge 4860 board, with UNIX V 3.2 OS. All other discounts and promotions do not apply. Offer expires March 30, 1991

9150 SW Pioneer Court, Suite H, Wilsonville, Oregon 97070

Using VGA display adapters from the i860

The Hauppauge i860VGA Library provides the i860 programmer direct access to VGA graphics capabilities from C programs written under Hauppauge's release of i860/APX-UNIX for the Intel i860 RISC processor. Access to the routines is via the standard C calling sequence and link-editting process. First, a call to the initialization routine, **vga_init()**, is made. Next, calls are made to other VGA library routines to perform the desired functions. Finally, a VGA cleanup routine, **vga_term()**, is called.

Housekeeping: Initialization and Termination

The VGA routines must be initialized before use. This is accomplished by calling the **vga_init()** routine and checking the **int** return value. If **vga_init()** returns a non-zero value, and error occurred while initializing the VGA. It is the user's responsibility to determine the reason for failure. Additional calls to other VGA library routines should not be made if **vga_init()** fails.

The **vga_init()** routine does basic VGA hardware initialization, in addition to setting up certain global values for programming use. VGA graphics mode HEX 13 (0x13) is selected, a default color map is loaded, the screen is cleared, and the VGA display RAM is mapped into the user's address space. VGA mode 0x13 gives the user a 320x200 pixel screen, with the 256 colors. This occupies 64000 bytes of video RAM (320 * 200 * 8 = 64000). Direct access to this video RAM is available; more on that later.

When VGA library functions are no longer needed in a program, a call to **vga_term()** must be made. The **vga_term()** routine essentially undoes the operations performed by **vga_init()**. That is, it returns the display to the default CGA mode 0x03 and removes the mapping of the VGA RAM from the user's address

space. Failure to call **vga_term()** before exitting from a program will leave the display in VGA graphics mode 0x13; you will not be able to get back to a Unix or DOS prompt.

A sample code fragment that initializes the VGA library might look like this:

```
main(){
    int rv;
    /* Initialize the VGA library */
    if((rv = vga_init()) != 0){
        printf("Could not initialize VGA -- error %d\n", rv);
        exit(1);
    }
    /* Insert other VGA calls here */
    /* Terminate (and reset) the VGA library */
    vga_term();
    exit(0);
}
```

VGA Library Functions

Here is a list of all currently implemented VGA library routines, along with a brief description of what each one does:

vga_init() Initializes the VGA routines. Returns zero on success.

vga_term() Terminates use of the VGA routines, and restores display to CGA mode 0x03.

vga_clear_screen() Clear the screen. That is, write zeroes to all 64000 bytes of VGA display RAM.

vga_color(c) : int c Set the color for future operations to c. c must have a value between 0 and 255 inclusive.

vga_rop(r) : int r Set the raster operation for future operations to r. r must have a value between VGA_MINROP and VGA_MAXROP (see vga.h).

vga_move(x, y): int x, y Set the position for the next drawing (or text) operation to the absolute screen coordinates (x, y). x must have a value between 0 and 319 inclusive, and y must have a value between 0 and 199 inclusive.

vga_line(x, y): int x, y Draw a line (using the current color, as set by vga_color()) from the current drawing position to (x, y). x must have a value between 0 and 319 inclusive, and y must have a value between 0 and 199 inclusive.

vga_rect(xc, yc, xl, yl): int xc, yc, xl, yl Draw a rectangle on the screen using the current color. The rectangle has its upper-left corner at (xc, yc). The width of the rectangle is xl, and the height is yl.

vga_loadfont(fn): char *fn Load X-window style font file fn into memory, and return an integer value representing the font-ID to be used for vga_setfont() calls. If there is an error loading the font file, vga_loadfont() will return a negative value. Valid font-ID's are always non-negative integers.

vga_setfont(fid): int fid Set the font for future text operations to the font specified by fid. fid was a valid font-ID returned by vga_loadfont().

vga_unloadfont(fid): int fid Disassociate the font specified by fid, and free the memory the font used. Once a font is unloaded, you can not use the font-ID again. You must explicity reload the font using vga_loadfont(), and use the new font-ID returned.

vga_string(s, n): char *s; int n Write n characters of the string pointed to by s at the current position (as set by vga_move() or vga_line()). The font used is the one specified by the most recent vga_setfont() call. The characters are written using the current color and raster-op. The base-line of the characters writ-

ten is that of the current y position. The x position is updated to point to the end of the text just written. That is, two successive calls to vga_string() will cause text to be output next to each other, NOT on top of each other.

vga_fill_screen(c): int c Fill the screen with color c. This is the same as vga_clear_screen(), except instead of using color 0, color c is used. If you wish to fill the entire screen with a particular color, this method is preferred as opposed to vga_rect(0, 0, 320, 200). vga_fill_screen() is an optimally coded assembler language routine, designed for speed.

Global Variables

There are several constants and global variables available to the user to make program developement simpler. The first thing a programmer should do is #include "**vga.h**" in his program. **vga.h** contains constants such as the values of the default colors, (VGA_BLACK, VGA_GREEN, ...), the raster operations, (ROP_STORE, ROP_OR, ...), the video RAM address (VRAM) and video RAM size (VRAMSIZE). Also included in **vga.h** are structure templates for color map entries and type declarations for global variables (pos_t, color_t). There are only a few global variables available to the programmer. These are listed below:

pos_t _FB_XPOS, _FB_YPOS This is the current (x, y) position. Be careful if you modify these; side-effects are unpredictable.

color_t _FB_COLOR The current color for text/draw operations.

int _FB_ROP The current raster operation as set by vga_rop(). The default raster operation (if you do not explicity set one) is ROP_STORE.

Direct Access to Video RAM

After VGA initialization programmers may access video RAM directly by using the address defined by VRAM in **vga.h**. The value of VRAM is usually (unsignedchar*)(0xf80a0000). You will notice that normal VGA RAM starts at 0xa0000. Under APX, all you have to do is add 0xf8000000 to the VGA RAM addresses

you usually use. You can use VGA RAM like normal memory to perform various specialized tasks (like loading bitmap images, for example). A code fragment that reads a VGA bitmap file into the VGA RAM might look like this:

```
#include        <stdio.h>
#include "vga.h"
main(){
    FILE *fp;
    if((fp = fopen("vga_image.dat", "r")) == NULL){
        perror("Can't open vga_image.dat");
        exit(1);
    }
    if(vga_init()){
        fprintf(stderr, "Can't initialize VGA library\n");
        exit(2);
    }
    if(fread(VRAM, 1, 64000, fp) != 64000){
        fprintf(stderr, "Could not read entire image");
        goto out;
    }
    /* At this point, the entire screen image has been
       read into VRAM */
    /* Wait for a character from the keyboard */
    getc(stdin);
out:
    fclose(fp);
    vga_term();
    exit(0);
}
```

Using the i860 Assembler, Linker and Debugger

Running the i860 Assembler

The i860 Assembler is a cross compiler that runs on the host operating system, generating linkable object code that must be run through the i860 Linker to create a program that can be executed on the i860 processor. Currently only versions of UNIX are supported as the host operating system, though the output from the i860 Linker can be run under DOS (see the paragraph entitled "Running i860 code under DOS").

The i860 Assembler is invoked by the following command:

asm860 [[[options]..]input_file]

where the brackets [and] show options, but are not used in the actual command line. The options must be seperated by at least one space. Some of the options are:

-a Do not automatically import any symbols that are referenced but are otherwise undefined. Issue an error message for this case.

-be Handle all data defined in data sections in "big endian" mode (i.e. with most significant byte as lowest address).

-D sym=val Define sym as a local symbol in the macro processor with value val. The -z option must also be set.

-I inc_file Include (via the macro processor) the file inc_file before the first statement of the input_file. The -z option must also be set. To include multiple input files, use several -I statements.

-i386 Put the magic number for the 386 Architecture in the output object module instead of the magic number for the i860. This is so that i386 tools can be used to process the output from the i860 Assembler.

-I file_spec Enable a source listing. If file_spec is included after the I without any spaces, the source listing is directed to a disk file with the name file_spec. If no file_spec is named, the output will go to the standard output device.

-L Preserve text symbols that begin with .L. High level languages generate labels that begin with .L.

-m Ignore (in the assembly pass) the special directives output (in the macro pass) by the -y option.

-o output_file Set the name of the output module to output_file. If this option is not set, the i860 Asembler creates an output name of input_name.o For example, if the input file name is "hello.860" (containing i860 assembly language), if no ouput_file is specified, the i860 Assembler will create a file called "hello.o".

-R Suppress all .data directives; all code is assembled in the text segment, and the data segment is not used.

-x Enable additional checks of the program to find illegal sequences of instructions.

-y Output in the macro pass special directives that the assembler pass uses for better reporting of the lines in the source file where errors are detected. The -z option must also be used.

-z Use the macro processor to scan the input_file before the assembler stage. The -z option must be used for any macro processing to take place.

Running the i860 Linker

The i860 Linker is used to create a runnable i860 program from the output of the i860 Assembler. The i860 Linker puts together

modules created by the i860 Assembler (it may only be one module for a simple i860 program) and resolves inter-module symbol references to create a i860 program that can be directly executed.

The i860 Linker is executed by the following command:

ld860 [option]...[[-B bss_addr] {[-p]input_file}...]...

The brackets [and] show options but are not used in the actual command line, and can be any of the following:

-B bss_addr Specify the RAM address to be used for common blocks for the bss section of all in_file modules that follow. This sepcification can be repeated to specify different addreses for different addresses for different groups of modules. The bss_address is a hexidecimal integer.

-d integer Specify the address at which the data section is to be loaded. The default starting address is 0x1000.

-D integer Specify the length of the data section as integer bytes. The data section is padded with NULs to the specified length. The integer may not be less than the original length of the data section.

-e symbol Specify symbol as the entry point of the program. The default entry point is ls$start.

-f list_file Read list_file for a list of i860 object modules to be linked. Names in this file can be separated by commas, spaces, tabs or newlines. Multiple -f statements are allowed. When the -f statement is used, the -F and -p options may not be used.

-i386 Use the 386 magix number instead of the i860 magic number in the output file. This will allow i386 tools to be used on the i860 Linker output.

-k Start the text and data sections exactly at the addresses

specified by the -T and -d options (or the default starting address if -T or -d options are not used) without performing the normal modifications to those addresses to make the file pageable.

-M Create a load map. If the -M is followed by a redirection command (out_file), the load map will be put into the output_file. Otherwisw the load map will be sent to the standard output device.

-Mx Create a load map plus a cross reference.

-o output_file Put the output module in a file called Output_file. Otherwise use a file called: a.out

-p Align the data section of the following input module on a page boundary. This is useful for producing page directories and tables.

-r Retain relocation entries in output_file. This enables incremental linking. The output_file can be used as input to another i860 Linker step. When -r is used, the -o option must also be used.

-s Strip all symbols from the output file.

-t Print the name of each input file as it is processed.

-T integer Specify the address at which the text section is to be loaded. The integer is the starting address. The default starting address is 0xF0400000.

-u symbol Initialize the symbol table with symbol. The i860 Linker considers symbol to be undefined; therefore it may be used to trigger the loading of the first member of an archive library.

-v Print the names of the input modules and libraries as they are processed.

Executing an i860 program and using the i860 Debugger

The i860 Debugger, similar in usage to the UNIX adb, will use the output from the i860 Linker, and allow the following functions:

- Setting of breakpoints in both code and data

- Reference to memory and registers via symbolic names

- Examine and set registers and memory sections

- Dissassemble i860 instructions symbolically

- Display i860 pipeline stages and call frames from the stack

- Single step i860 instructions, allowing "stepping over" procedure calls

- Debug high level language programs from source-code level.

To run the i860 Debugger, use the following command line:

run860 [-d][-w][-R][-r[register_type]][-I dir][[-S dir{dir}..][-F[command_file]] object_file [parameters]

where:

object_file is the object file created by the i860 Linker, and should be executable

-d Tells the i860 Debugger to start an interactive debug session, taking requests from the keyboard. If the -d option is not used, run860 simply executes the program.

-w Creates object_file (if necessary), and opens it for reading and writing so that this file can be modified during the debug session.

-R Instructs the i860 Debugger to display all processor registers when a batch session is terminated. This aids in comparing the

results of one debug session to a previous debug session.

-r Displays the values of specified registers when a batch session is terminated. Registers are displayed by types: f = floating point registers (single precision), d = pairs of floating point registers (double precision), i = pairs of floating point registers (64-bit integers), p = piplines, s = simulator registers. If no type is specified, the i860 Debugger displays the integer registers and control registers.

-I directory Specifies the directory that contains files to be read with the commands $ or $<. The default is the current directory. No space is permitted between the -I and the directory.

-S Provides a list of directories to be used during source debugging when searching for source files. The current directory is always searched first.

-F command_file Tells the i860 Debugger to execute requests found in command_file when staring a debug session. This command assumes interactive debugging (the -d option must be used). The command_file usually contains commands to set parameters, to set standard breakpoints, and to initialize registers. If no command_file is specified after the -F, the i860 Debugger assumes a file named object_file.cmd is used.

i860 Debugger Commands

The i860 Debugger commands may use address or line expressions as part of their command line. A line number corresponds to a line from a source program file (either C or Fortran). Note that to preserve line numbers during the Compile/Assemble/Link steps, specify the -g option during compilation, and do not use the -s option during Linking, otherwise the lines numbers will be stripped from the final object module used by the i860 Debugger.

The format of a command line is:

[address_exp][,count_exp] command [modifier] [;] where

the address_exp can be either a memory address or a line number (when doing source code level debugging), count_exp is used to specify the number of times the command is executed, and the modifier is dependant upon the command. In the following list of commands, the count_exp has not been shown, but may be used unless otherwise noted. Several debug commands can be used on the same line by separating them be the semicolon (;).

i860 processor registers

All of the registers within the i860 processor may be viewed with the $r command, or modified with the command. Please refer to the i860 Programmers Reference Manual (Intel Corp.) or the i860 Architecture Reference (McGraw Hill Publishing, author: Neil Margolis) for a listing of the i860 registers.

The format of a line number is:

#line_number[~source_file] where source_file, if not specified, is taken from the directories from the -S option during Debugger invocation.

address_exp:b Set a code breakpoint at address_exp

address_exp:ba Set breakpoint at address_exp after any access

address_exp:brd Set breakpoint at address_exp after read

address_exp:brw Set breakpoint at address_exp after write

address_exp:c Continue program execution at address_exp. Normally used to continue execution after a breakpoint has been reached. If count_exp is given, skip the next count_exp-1 breakpoints.

address_exp:d Delete breakpoint at address_exp. If address_exp is not given, delete all breakpoints.

address_exp:r [parms] Run object_file. Other command line

parameters (parms) may be given after the :r command. The address_exp, if present, gives a starting location. If not present, execution starts at the standard entry point. If a count_exp is given, skip the next count_exp-1 breakpoints. All program variables are reinitialized, but the breakpoints, registers and i860 Debugger variables are not changed.

count_exp:s Single steps count_exp times.

count_exp:S Single steps count_exp times, but skips over procedure calls.

address_expname Assign the value of address_exp to name. Name can be an i860 register or a debug register.

address_exp /i [format] Disassmble contents of memory starting at address_exp. This command can only be run after execution has been started by the :r, :R, :s, :S or :I commands.

Source code debug statements

address_exp,count ?i [format] Disassemble count lines of source code from object_file starting at address_exp.

line_number:a Display the instructions generated by the compiler at line_number.

line_number:i [parms] Single step count_exp source lines starting at line_number.

line_number:I [parms] Single step count_exp source lines but skip over procedure calls.

line_number:w Display the source code starting at statement line_number. Display count_exp lines.

:n Display the next set of source lines using the count_exp from the previous :w command.

:p Display the previous set of source lines using the

count_exp from the previous :w command.

Debugger restrictions

For instructions that execute in dual-instruction mode, breakpoints can be set only on the core instructions.

A breakpoint cannot be set on a delayed branch instruction.

When delayed branch instructions are used in the dual instruction mode, a breakpoint cannot be set on any of the three instructions that follow the delayed branch.

4860 Programmers Reference

This chapter is intended to aid systems programmers in writing software for, or porting software to, the Hauppauge 4860 family of MotherBoards. The Hauppauge 4860 provides PC compatible platforms with two industry standard I/O busses, the PC/AT bus (ISA) and the EISA bus.

This draft version of the system programmers reference manual focuses on describing 4860 specific features. It does not provide information on programming the industry standard MotherBoard peripherals such as timer/counters, DMA ports, etc..

I/O Port Summary (ISA/EISA)

Address	Item	Access
0F8H	Motherboard Configuration Register	Write Only
0F9H	Unused	
0FAH	Unused	
0FBH	486 Attention Interrupt	Read/Write
0FCH	860 Attention Interrupt	Write Only
0FDH	860 Configuration Register	Write Only
0FEH	860 Interrupt Controller	Read/Write
0FFH	860 Interrupt Controller Mask	Write Only

I/O Port Summary (EISA ONLY)

Address	Item	Access
C00H	Configuration RAM page register	Write Only
800H-8FFH	Configuration RAM	Read/Write
C80H-C83H	EISA ID Registers	Read Only

486 Processor Memory Map

00000000H - 03FFFFFF	Mappable as onboard DRAM, Expansion RAM, I/O bus or BIOS ROM
04000000H - BFFFFFFF	Expansion RAM, or I/O Bus
C0000000H - C1FFFFFF	Weitek 4167
C2000000H - C200FFFF	i860 I/O space
FC000000H - FFFFFFFF	Mappable as onboard DRAM, Expansion RAM, I/O Bus or BIOS ROM

Note: The state of the keyboard controller output A20 Gate will affect access to some portions of the 486's memory space.

860 Processor Memory Map

00000000H - 03FFFFFF	Mappable as onboard DRAM, Expansion Ram, I/O Bus or BIOS ROM
04000000H - BFFFFFFF	Expansion Ram, or I/O Bus
C0000000H - C1FFFFFF	Weitek 4167
C2000000H - C200FFFF	860 I/O Space
C2010000H - FBFFFFFF	Expansion Ram, or I/O Bus

FC000000H - FFFFFFFF	Mappable as onboard DRAM, Expansion Ram, I/O Bus or BIOS ROM

Note: The state of the keyboard controller output A20 Gate will **NOT** affect the 860s memory map.

4860 Frame Buffer Memory Map

C3000000H - C303FFFF	Frame buffer memory
C2800000H - C2800018	BT431 Cursor controller
C2400000H - C2400018	BT457 DAC/Palette
C2000000H	D1 = Frame Interrupt MASK (Active LOW)
C2000008H	D0 = VSYNC (Active LOW)
C2000010H	D1 = Frame Interrupt (Active LOW)

4860 MotherBoard Configuration Register

The Motherboard configuration register is an 8 bit write only register. It is located at I/O port 0F8H and contains the following information:

Bit 0	0 = Disable Mapping RAM
	1 = Enable Mapping RAM
Bit 1	0 = Disable writes to Mapping RAM
	1 = Enable writes to Mapping RAM
Bit 2	0 = Select 486 cells when writing
	1 = Select 860 cells when writing
Bit 3	0 = Normal Parity
	1 = Force Parity Error

Bits 4 - 5	Reserved
Bit 6	0 = Enable Onboard Serial/Parallel Ports
	1 = Disable Onboard Serial/Parallel Ports
Bit 7	0 = 486 External Cache Enabled
	1 = 486 External Cache Disabled

In order to allow programs to set and clear individual bits in the configuration register, a copy of the current configuration register image is stored in CMOS RAM.

Mapping RAM

The Memory Mapping RAM determines what resource each processor accesses at a given memory address:

- The Mapping RAM allows each 32K block of the first 64 Megabytes to be enabled as RAM or disabled, allowing access to off board RAM (64 bit expansion memory or I/O Bus RAM).

- In addition each block may also be marked as cacheable or noncacheable, and as read/write or read only (write protected).

The same capabilities are provided for the last 64 megabytes of memory as well.

Each processor (the 486 and the 860) has an independent memory map allowing each block of memory to be accessible to the 486, the 860, or both.

The Mapping RAM is 8K bytes, with each byte controlling access to 32K bytes of one processor's address space (4 Gigabytes of memory is mappable). Each byte in the Mapping RAM is formatted as follows:

Bit 0	0 = Block is Read / Write
	1 = Block is Read Only (write protected)

Bit 1 0 = Block is cacheable

1 = Block may not be cached

Bit 2 0 = Block is fast RAM

1 = Block is slow RAM

Bit 3 reserved (must be set to a 1)

Bit 4 0 = Block has RAM enabled

1 = Block has RAM disabled

Bit 5 0 = Front side of SIMM Module Selected

1 = Back side of SIMM Module Selected

Bits 6 - 7 Bank number of SIMM module pair to select.

for example, if Bits 6 and 7 are both 0, this selects SIMM Bank 0, sockets U20 and U24.

NOTE: if Bit 4 is 1 (Block does not use on board RAM), Bits 5, 6 and 7 have different definitions:

Bit 5 RESERVED

Bit 6 0 = Access EISA address = > 16 Megabytes

1 = Access EISA address < 16 Megabytes

Bit 7 0 = EISA bus

1 = On-board EPROM select

Initial programming of the Mapping RAM is accomplished by:

1) Disable Interrupts

2) Set the Enable Writes bit and the Select 486 bit in the **4860 MotherBoard Configuration Register**

3) For each of the 4096 memory blocks, write that blocks' 486 control word to any memory address within the block it controls (any write to memory space while the Mapping RAM Writes are enabled will result in a write to the corresponding mapping RAM

byte)

4) Set the Select 860 bit in the **4860 MotherBoard Configuration Register**

5) For each of the 4096 memory blocks, write that blocks' 860 control word to any memory address within the block it controls

6) Clear the Enable Writes bit and set the Enable Mapping Ram bit in the **4860 MotherBoard Configuration Register**

7) Enable Interrupts

Subsequent changes to the Mapping RAM are accomplished as follows:

1) Disable Interrupts

2) Set the Enable Writes bit and set or clear the Select 486/860 bit in the configuration register

3) Write the block's control word to any memory address within the block it controls

4) Clear the Enable Writes bit in the configuration register

5) Enable Interrupts

Note: Any write to memory space while the Mapping RAM writes are enabled will result in a write to the mapping RAM. This includes write generated by DMA or another processor (hint: turn off DMA and interrupts when programming the Mapping RAM).

860 Processor Configuration Register

The 860 configuration register is an 8 bit write only register. It is located at I/O port 0FDH and contains the following information:

Bits 0 - 1 Arbiter Mode

Bit 1	Bit 0	Mode
0	0	Fair
0	1	Priority 486
1	0	Priority 860
1	1	Priority 64-Bit Slot

Bit 2 486 Interrupt Select

0 = Write to 486 Interrupt Register sends external interrupt 13.

1 = Write to 486 Interrupt Register sends NMI.

Bit 3:

0 = I/O Command delays inserted

1 = No I/O command delays

When enabled, 3 I/O Bus clocks (BCLKs) are inserted for 16-bit I/O devices, and 11 BCLKs are inserted for 8-bit I/O devices.

Bit 4:

0 = EISA I/O READ delayed by one BCLK

1 = No EISA I/O READ delays

Bit 5-6 Reserved

Bit 7:

0 = 860 running

1 = 860 held in reset

In order to allow programs to set and clear individual bits in the configuration register, a copy of the current configuration register image is stored in CMOS RAM.

860 Processor Interrupt Controller

When the 860 receives an interrupt it can determine the source or sources by reading the 860 pending interrupt register at I/O port 0FEH. Each bit in this register indicates a pending interrupt as follows:

Bit 0 IO system interrupt (level sensitive from

8259)

Bit 1	Timer interrupt (edge sensitive)
Bit 2	860 Attention interrupt (edge sensitive)
Bit 3	Parity interrupt (edge sensitive)
Bit 4	64-bit Expansion Slot interrupt
Bit 5	Unused
Bits 6 - 7	Reserved

Writing to I/O port 0FEH allows bits in the pending interrupt register to be selectively cleared. Writing a one to a bit in port 0FEH clears the corresponding bit in the pending interrupt register.

The 860 interrupt Mask register is a write-only register at I/O port 0FFH. This register contains a mask bit for each 860 interrupt. When an interrupt mask bit is set, no 860 interrupt is generated when the corresponding bit in the pending interruptregister is set.

Master/NON-Master EISA I/O Slots

Slots J1, J3-J6 and J8 are capable of supporting Master-mode EISA boards. Slots J2 and J7 do not support Master-Mode EISA boards.

EISA Slot Addresses

EISA slots are encoded to work in certain address ranges when used with adapter boards using AEN selection. The following address ranges apply to the slots on the 4860EISA MotherBoard:

Slot	Address Range
J1	10xx, 14xx, 18xx, 1Cxx
J2	20xx, 24xx, 28xx, 2Cxx
J3	30xx, 34xx, 38xx, 3Cxx
J4	40xx, 44xx, 48xx, 4Cxx

J5 50xx, 54xx, 58xx, 5Cxx

J6 60xx, 64xx, 68xx, 6Cxx

J7 70xx, 74xx, 78xx, 7Cxx

J8 80xx, 84xx, 88xx, 8Cxx

Keyboard Controller Input Port

Bit 0 Reserved

Bit 1 Reserved

Bit 2 0 = Turbo switch inactive

1 = Turbo switch active

Bit 3 0 = Second Level Cache present

1 = Second Level Cache not present

Bit 4 0 = Expansion RAM card present

1 = Expansion RAM card not present

Bit 5 0 = Manufacturing Test jumper installed

1 = Manufacturing Test jumper not installed

Bit 6 0 = Color monitor selected

1 = Monochrome monitor selected

Bit 7 0 = Keyboard Locked

1 = Keyboard Unlocked

486 Attention Interrupt

Writing to I/O port 0FBH generates either an 8259 interrupt 13 (AT Math Coprocessor error) or an NMI to the 486. Which interrupt is generated is selectable using the 860 Configuration Register.

When the 486 receives the interrupt it can determine the source by reading I/O port 0FBH and examining bit 0. Bit 0 equal to 1 indicates that the 486 interrupt register generated the interrupt. Reading port 0FBH causes bit 0 to be cleared following the read.

860 Attention Interrupt

Writing to I/O port 0FCH generates an 860 Attention Interrupt.

486 BIOS Special Functions

- Master Processor Selection
- Memory Detection & Setup
- Weitek support
- 860 support
- Setup
  - Ram Speed
  - Bus Speed
  - Shadow Ram
  - Master Processor
  - 486/860 Ram Allocation
  - Internal & External Cache Control

860 BIOS Functions

- Initialization
- 486 Detection
- Master Processor Selection
- POST
- Setup
  - Ram Speed
  - Bus Speed
  - Shadow Ram
  - Master Processor
  - Internal Cache Control
  - 486/860 Ram Allocation
- Bootstrap

Heartbeat demo

```
UNIX386 80860 ASSEMBLER, Beta Version 2.0, 11-Apr-1989
Tue Dec 26 16:31:58 1989      Page 1
// This program, when run on a Hauppauge 4860 board with both i486 and i860
// processors, will have the i860 write alternating astericks in the upper right hand
// corner of a Hercules compatible display, while the i486 continues to run DOS.  It is
// loaded from DOS with the Hauppauge utility program LOAD860.

                                1       screen_base = 0x0B0000  // Hercules screen
                                2       screen_offs = 160 - 8   // Place to put *
                                3
                                4       delay_count = 1000000   // Delay between *'s
                                5
                                6       iobase= 0xC2000000       // 860 I/O Space
                                7
                                8        .atmp r31
                                9        .text
                                10
                                11      // initialize control reg
                                12      // leave all ints disable
                                13
38200000 00000000               14      st.c    r0,psr
38800000 00000004               15      st.c    r0,fsr
 30000000 00000008              16      ld.c    fir,r0
                                17
                                18      // clear pipeline
48000430 0000000c               19      pfadd.ss                f0,f0,f0
48000430 00000010               20      pfadd.ss                f0,f0,f0
48000430 00000014               21      pfadd.ss                f0,f0,f0
48000420 00000018               22      pfmul.ss f0,f0,f0
 48000420 0000001c              23      pfmul.ss f0,f0,f0
 48000420 00000020              24      pfmul.ss f0,f0,f0
                                25
edefc200e40f0000 00000024       26      mov      iobase, r15
 94100001 0000002c              27      mov     0x01,r16
 0de08080 00000030              28      st.b    r16, 0x80(r15)  //write 01 to port 80h
                                29
 6c000022 00000034              30      call    flush_cache     // make sure we get a miss
 a0000000 00000038              31      nop
                                32
 94100002 0000003c              33      mov     0x02,r16
 0de08080 00000040              34      st.b    r16, 0x80(r15)  //write 02 to port 80h
                                35
 a0020000 00000044              36      mov      r0,r2
```

```
ee52000be4120000 00000048   37      mov     screen_base, r18//set up screen
                            38
 94100003 00000050          39      mov     0x03,r16
 0de08080 00000054          40      st.b    r16, 0x80(r15)     //write 03 to port 80h
                            41
                            42  loop:
                            43
 6c00000f 00000058          44      call    delay
 a0000000 0000005c          45      nop
                            46
 9410072a 00000060          47      mov     0x072A,r16
 0de08080 00000064          48      st.b    r16, 0x80(r15)
 1e408098 00000068          49      st.s    r16, screen_offs(r18)      // *
 94100720 0000006c          50      mov     0x0720,r16
 1e40809a 00000070          51      st.s    r16, screen_offs+2(r18)    //space
                            52
 6c000008 00000074          53      call    delay
 a0000000 00000078          54      nop
                            55
 94100720 0000007c          56      mov     0x0720,r16
 0de08080 00000080          57      st.b    r16, 0x80(r15)
 1e408098 00000084          58      st.s    r16, screen_offs(r18)      //space
 9410072a 00000088          59      mov     0x072A,r16
 1e40809a 0000008c          60      st.s    r16, screen_offs+2(r18)    // *
                            61
 6bfffff1 00000090          62      br      loop
 a0000000 00000094          63      nop
                            64
                            65
                            66  delay::
 9405ffff 00000098          67      mov     -1, r5
ecc6000fe4064240 0000009c   68      mov     delay_count, r6
 b4c02801 000000a4          69      bla     r5, r6, delay_loop
 a0000000 000000a8          70      nop
                            71
                            72  delay_loop:
 b4df2fff 000000ac          73      bla     r5, r6, delay_loop
 a0000000 000000b0          74      nop
                            75
 40000800 000000b4          76      bri     r1
 a0000000 000000b8          77      nop
                            78
                            79
80  //The following cache flush procedure is from the i860TM Programmer's Reference
81  //manual.       Please reference this Intel manual for additional information.
                            82
 a0000000 000000bc          83      .align  .quad
                            84
                            85  flush_cache::
```

```
                                    86
                                    87      FLUSH_P= 0x7f000000-32
                                    88
                                    89      //rw=r24, rx=r25, ry=r26, rz=r27
                                    90
a0220000 000000c0                   91      mov     r1,r2
305b0000 000000c4                   92      ld.c    dirbase,r27
e77b0800 000000c8                   93      or      0x800,r27,r27
9419ffff 000000cc                   94      adds    -1,r0,r25
6c000008 000000d0                   95      call    D_FLUSH
3840d800 000000d4                   96      st.c    r27,dirbase
                                    97
e77b0900 000000d8                   98      or      0x900,r27,r27
6c000005 000000dc                   99      call    D_FLUSH
3840d800 000000e0                   100     st.c    r27,dirbase
                                    101
f77b0900 000000e4                   102     xor     0x900,r27,r27
a0410000 000000e8                   103     mov     r2,r1
40000800 000000ec                   104     bri     r1
3840d800 000000f0                   105     st.c    r27,dirbase
                                    106
                                    107     D_FLUSH::
e418ffe0 000000f4                   108     or      l%FLUSH_P,r0,r24
ef187eff 000000f8                   109     orh     h%FLUSH_P,r24,r24
e41a007f 000000fc                   110     or      127,r0,r26
b740c803 00000100                   111     bla     r25,r26,D_FLUSH_LOOP
17000021 00000104                   112     ld.l    32(r24),r0
                                    113
a0000000a0000000 00000108           114     .align  .quad
                                    115
                                    116     D_FLUSH_LOOP::
08000000 00000110                   117     ixfr    r0, f0
b75fcffe 00000114                   118     bla     r25,r26,D_FLUSH_LOOP
37000021 00000118                   119     flush   32(r24)++
08000000 0000011c                   120     ixfr    r0, f0
                                    121
40000800 00000120                   122     bri     r1
1700fe01 00000124                   123     ld.l    -512(r24),r0
                                    124
      00000128                      125     .end
```

```
UNIX386 80860 ASSEMBLER, Beta Version 2.0, 11-Apr-1989
Tue Dec 26 16:31:57 1989      Page 1

// Program:  Reset
// Sets the initial vector to jump to on the 860 processor.

                                    1
                                    2       test_addr = 0xfffff000        // rom entry address
                                    3
                                    4        .text
                                    5       .align      .quad
                                    6
a0000000 00000000                   7   nop     // two nop's for A0 errata
a0000000 00000004                   8   nop
                                    9
9401f000 00000008                  10   mov     test_addr, r1              // jump to test program
40000800 0000000c                  11   bri     r1
a0000000 00000010                  12   nop
                                   13
00000014                           14   .end
```

Frame Buffer Demo

UNIX386 80860 ASSEMBLER, Beta Version 2.0, 11-Apr-1989
Fri Apr 27 14:28:32 1990 Page 1

```
// This program displays a test pattern on the Hauppauge 4860 Frame Buffer.  The i860
// changes pixels in the frame buffer by writing to memory locations in the frame
// buffers memory space, which starts at location VRAM_ADDRESS.
//
//  This test program will work only with version X.01 of the 64-bit frame buffer.

 1
 2
 3      iobase = 0xC2000000       // 860 I/O Spac
 4
 5      // display resolution
 6
 7      DISPLAY_WIDTH             = 1280
 8      DISPLAY_LINES             = 1024
 9
10      HORIZ_STRIDE              = 2048
11
12      VRAM_ADDRESS              = 0x03000000
13      VRAM_SIZE                 = 4 * 1024 * 1024
14
15      VRAM2                     = 0x03200000
16
17      //Bt457 - Brooktree Pallette DAC address
18
19      PALETTE_DAC_ADDRESS   = 0x02400000
20
21      RED_ADDRESS_PORT      = 0x00
22      RED_PALETTE_RAM_PORT = 0x08
23      RED_REGISTER_PORT     = 0x10
24      RED_OVERLAY_PORT      = 0x18
25
26      GREEN_ADDRESS_PORT    = 0x20
27      GREEN_PALETTE_RAM_PORT        = 0x28
28      GREEN_REGISTER_PORT   = 0x30
29      GREEN_OVERLAY_PORT    = 0x38
30
31      BLUE_ADDRESS_PORT     = 0x40
32      BLUE_PALETTE_RAM_PORT         = 0x48
33      BLUE_REGISTER_PORT    = 0x50
34      BLUE_OVERLAY_PORT     = 0x58
35
```

```
36      DAC_OVERLAY_0           = 0x0
37      DAC_OVERLAY_1           = 0x1
38      DAC_OVERLAY_2           = 0x2
39      DAC_OVERLAY_3           = 0x3
40      DAC_READ_MASK_REG       = 0x4
41      DAC_BLINK_MASK_REG      = 0x5
42      DAC_COMMAND_REG         = 0x6
43      DAC_CONTROL_REG         = 0x7
44
45      //          Bt431 - Brooktree Cursor chips
46
47      CURSOR = 0x02800000
48
49      CUR_ADDRESS_0_PORT      = 0x00
50      CUR_ADDRESS_1_PORT      = 0x08
51      CUR_RAM_PORT            = 0x10
52      CUR_CONTROL_REG_PORT            = 0x18
53
54
55
56      PORTS                   = 0x02000000
57
58      //          Bt439 - Brooktree clock chip
59
60      Bt439_reset_port        = 0x08
61
62      clk_on                  = 0x00000000
63      clk_off                 = 0x00000002
64
65      //          Frame Buffer Status Ports
66      //
67      //Read      0x00        Bit 0-Buffer Select
68      //Read      0x00        Bit 1-Frame Int. Mask
69      //
70      //Write     0x00        Bit 0-
71      //Write     0x00        Bit 1-
72      //
73      //Read      0x08        Bit 0-Vertical Sync
74      //Read      0x08        Bit 1-Clock Reset
75      //
76      //Write     0x08        Bit 0-
77      //Write     0x08        Bit 1-Clock Reset
78      //
79      //Read      0x10        Bit 0-Mouse Interrupt
80      //Read      0x10        Bit 1-Frame Interrupt
81      //
82      //Write     0x10        Bit 0     -
83      //Write     0x10        Bit 1     -
84      //
```

```
85    //Read   0x18          Bit 0     -
86    //Read   0x18          Bit 1     -
87    //
88    //Write  0x18          Bit 0-Buffer Select
89    //Write  0x18          Bit 1-Frame Interrupt Mask
90
91    FB_CLK_RST             = 0x02
92    FB_VSYNC               = 0x01
93
94    FB_Status_Port         = 0x08
95
96    FB_CLK_RST             = 0x02
97    FB_VSYNC               = 0x01
98
99
100   //some colors
101
102   RED      = 0b1111110000000000
103   GREEN    = 0b0000001111110000
104   BLUE     = 0b0000000000001111
105
106
107   .text
108
109   //       Register Usage:
110   //r1     Return Address
111   //r2     Temporary
112   //r3     Unused
113   //r4     Base for accessing I/O Ports
114   //r5     Base for Frame Buffer Ports
115   //r6     Base for accessing Palette DACs
116   //r7     Base for accessing Cursor
117   //r8     Unused
118   //r9     Unused
119   //r10    Clock OFF Command
120   //r11    Unused
121   //r12    Temporary
122   //r13    Unused
123   //r14    Unused
124   //r15    Unused
125   //r16    Temporary
126   //r17    Temporary
127   //r18    Temporary
128   //r19    Video RAM address pointer
129   //r20    Unused
130   //r21    Unused
131   //r22    Used to store pixel value for RED
132   //r23    Used to store pixel value for GREEN
133   //r24    Used to store pixel value for BLUE
```

```
                                134   //r25     Temporary
                                135   //r26     Temporary
                                136   //r27     Temporary
                                137   //r28     Temporary
                                138   //r29     Temporary
                                139   //r30     Temporary
                                140   //r31     Temporary
                                141   //
                                142   //
                                143
                                144
                                145   ld$start::
                                146
                                147   //        initialize control reg
                                148   //        leave all ints disable
                                149
38200000 00000000               150   st.c      r0,psr
38800000 00000004               151   st.c      r0,fsr
30000000 00000008               152   ld.c      fir,r0
                                153
                                154   //        clear pipeline
                                155
48000430 0000000c               156   pfadd.ss  f0,f0,f0
48000430 00000010               157   pfadd.ss  f0,f0,f0
48000430 00000014               158   pfadd.ss  f0,f0,f0
48000420 00000018               159   pfmul.ss  f0,f0,f0
48000420 0000001c               160   pfmul.ss  f0,f0,f0
48000420 00000020               161   pfmul.ss  f0,f0,f0
                                162
6c0000fa 00000024               163   call flush_cache  // make sure we get a miss
a0000000 00000028               164   nop
                                165
eca50200e4050000 0000002c       166   mov       PORTS,r5
                                167
ecc60240e4060000 00000034       168   mov       PALETTE_DAC_ADDRESS,r6
                                169
ece70280e4070000 0000003c       170   mov       CURSOR,r7
                                171
                                172   //        reset Bt439
                                173
940a0002 00000044               174   mov       clk_off,r10
1ca05008 00000048               175   st.s      r10,Bt439_reset_port(r5)
1ca00008 0000004c               176   st.s      r0,Bt439_reset_port(r5)
                                177
6c000056 00000050               178   call      wait_for_vsync
a0000000 00000054               179   nop
                                180
                                181   //at this point vsync just became active
                                182
```

```
1ca05008 00000058   183   st.s    r10,Bt439_reset_port(r5)
1ca00008 0000005c   184   st.s    r0,Bt439_reset_port(r5)
                    185
6c00005c 00000060   186   call    init_palette_dacs
a0000000 00000064   187   nop
                    188
                    189   //write some stuff to VRAM
                    190
6c0000de 00000068   191   call    clear_screen
a0000000 0000006c   192   nop
                    193
94110800 00000070   194   mov     HORIZ_STRIDE,r17
a6310001 00000074   195   shl     1,r17,r17                  // pitch
ee730300e4130000 00000078 196 mov   VRAM_ADDRESS,r19
eed60000e416fc00 00000080 197 mov   RED,r22
941703f0 00000088   198   mov     GREEN,r23
9418000f 0000008c   199   mov     BLUE,r24
                    200
a2de0000 00000090   201   mov     r22,r30                    // r30 = white
e3deb800 00000094   202   or      r23,r30,r30
e3dec000 00000098   203   or      r24,r30,r30
                    204
94120800 0000009c   205   mov     HORIZ_STRIDE,r18
ee730300e4130000 000000a0 206 mov   VRAM_ADDRESS,r19
                    207   hline::
1e60f000 000000a8   208   st.s    r30,0(r19)
96730002 000000ac   209   adds    2,r19,r19
9652ffff 000000b0   210   adds    -1,r18,r18
525f07fc 000000b4   211   btne    r0,r18,hline
                    212
94120800 000000b8   213   mov     HORIZ_STRIDE,r18
ee730320e4130000 000000bc 214 mov   VRAM2,r19
                    215   hline1::
1e60b800 000000c4   216   st.s    r23,0(r19)
96730002 000000c8   217   adds    2,r19,r19
9652ffff 000000cc   218   adds    -1,r18,r18
525f07fc 000000d0   219   btne    r0,r18,hline1
                    220
94120384 000000d4   221   mov     900,r18
ee730300e4130000 000000d8 222 mov   VRAM_ADDRESS,r19
a01f0000 000000e0   223   mov     r0,r31
                    224   line::
1e60b010 000000e4   225   st.s    r22,0x10(r19)
1e60b012 000000e8   226   st.s    r22,0x12(r19)
1e60b014 000000ec   227   st.s    r22,0x14(r19)
1e60b016 000000f0   228   st.s    r22,0x16(r19)
1e60b018 000000f4   229   st.s    r22,0x18(r19)
                    230
1e60b820 000000f8   231   st.s    r23,0x20(r19)
```

```
1e60b822 000000fc        232    st.s     r23,0x22(r19)
1e60b824 00000100        233    st.s     r23,0x24(r19)
1e60b826 00000104        234    st.s     r23,0x26(r19)
1e60b828 00000108        235    st.s     r23,0x28(r19)
                         236
1e60c030 0000010c        237    st.s     r24,0x30(r19)
1e60c032 00000110        238    st.s     r24,0x32(r19)
1e60c034 00000114        239    st.s     r24,0x34(r19)
1e60c036 00000118        240    st.s     r24,0x36(r19)
1e60c038 0000011c        241    st.s     r24,0x38(r19)
                         242
1e60f060 00000120        243    st.s     r30,0x60(r19)    //white line
                         244    //st.s   r22,2552(r19)    //RED line
                         245    //st.s   r23,2554(r19)    //GREEN line
                         246    //st.s   r24,2556(r19)    //BLUE line
                         247    //st.s   r30,2558(r19)    //white line
                         248
1e60f100 00000124        249    st.s     r30, 0x100(r19)  //fat white line
1e60f102 00000128        250    st.s     r30,0x102(r19)
1e60f104 0000012c        251    st.s     r30,0x104(r19)
1e60f106 00000130        252    st.s     r30,0x106(r19)
1e60f108 00000134        253    st.s     r30,0x108(r19)
1e60f10a 00000138        254    st.s     r30,0x10a(r19)
1e60f10c 0000013c        255    st.s     r30,0x10c(r19)
1e60f10e 00000140        256    st.s     r30,0x10e(r19)
1e60f110 00000144        257    st.s     r30,0x110(r19)
1e60f112 00000148        258    st.s     r30,0x112(r19)
1e60f114 0000014c        259    st.s     r30,0x114(r19)
                         260
1e600116 00000150        261    st.s     r0,0x116(r19)    //black line
                         262
1e60f118 00000154        263    st.s     r30,0x118(r19)   //fat white line
1e60f11a 00000158        264    st.s     r30,0x11a(r19)
1e60f11c 0000015c        265    st.s     r30,0x11c(r19)
1e60f11e 00000160        266    st.s     r30,0x11e(r19)
1e60f120 00000164        267    st.s     r30,0x120(r19)
1e60f122 00000168        268    st.s     r30,0x122(r19)
1e60f124 0000016c        269    st.s     r30,0x124(r19)
1e60f126 00000170        270    st.s     r30,0x126(r19)
1e60f128 00000174        271    st.s     r30,0x128(r19)
1e60f12a 00000178        272    st.s     r30,0x12a(r19)
1e60f12c 0000017c        273    st.s     r30,0x12c(r19)
                         274
a26c0000 00000180        275    mov      r19,r12
87ff0002 00000184        276    addu     2,r31,r31
83ec6000 00000188        277    addu     r12,r31,r12
1d80f000 0000018c        278    st.s     r30,0(r12)       //45 deg. white line
                         279
82339800 00000190        280    addu     r19,r17,r19      //next line
```

```
                        281
8652ffff 00000194       282     addu    -1,r18,r18
525f07d2 00000198       283     btne    r0,r18,line
                        284
6c000047 0000019c       285     call    init_cursor
a0000000 000001a0       286     nop
                        287
                        288
                        289     ok::
6bffffff 000001a4       290     br      ok
a0000000 000001a8       291     nop
                        292
                        293
                        294     //wait_for_vsync
                        295     //loop until vsync has just become active
                        296     //      and then return
                        297
                        298
                        299     wait_for_vsync::
                        300
                        301     //loop until vsync goes inactive
                        302
14bc0008 000001ac       303     ld.s    FB_Status_Port(r5), r28
c79c0001 000001b0       304     and     FB_VSYNC,r28, r28
73fffffd 000001b4       305     bc      wait_for_vsync
a0000000 000001b8       306     nop
                        307
                        308     //loop until vsync goes active
                        309
                        310     no_vsync::
14bc0008 000001bc       311     ld.s    FB_Status_Port(r5),r28
c79c0001 000001c0       312     and     FB_VSYNC,r28,r28
7bfffffd 000001c4       313     bnc     no_vsync
a0000000 000001c8       314     nop
                        315
                        316     //at this point, vsync just became active
                        317
40000800 000001cc       318     bri     r1
a0000000 000001d0       319     nop
                        320
                        321
                        322
                        323     init_palette_dacs::
a0220000 000001d4       324     mov     r1, r2
                        325
941c0004 000001d8       326     mov     DAC_READ_MASK_REG,    r28  //RED
0cc0e000 000001dc       327     st.b    r28, RED_ADDRESS_PORT(r6)
0cc0e020 000001e0       328     st.b    r28, GREEN_ADDRESS_PORT(r6)
0cc0e040 000001e4       329     st.b    r28, BLUE_ADDRESS_PORT(r6)
```

```
941c00ff 000001e8          330   mov     0xff, r28
0cc0e010 000001ec          331   st.b    r28, RED_REGISTER_PORT(r6)
0cc0e030 000001f0          332   st.b    r28, GREEN_REGISTER_PORT(r6)
0cc0e050 000001f4          333   st.b    r28,BLUE_REGISTER_PORT(r6)
                           334
941c0005 000001f8          335   mov     DAC_BLINK_MASK_REG, r28
0cc0e000 000001fc          336   st.b    r28, RED_ADDRESS_PORT(r6)
0cc0e020 00000200          337   st.b    r28, GREEN_ADDRESS_PORT(r6)
0cc0e040 00000204          338   st.b    r28, BLUE_ADDRESS_PORT(r6)
0cc00010 00000208          339   st.b    r0, RED_REGISTER_PORT(r6)
0cc00030 0000020c          340   st.b    r0, GREEN_REGISTER_PORT(r6)
0cc00050 00000210          341   st.b    r0, BLUE_REGISTER_PORT(r6)
                           342
941c0006 00000214          343   mov     DAC_COMMAND_REG,    r28
0cc0e000 00000218          344   st.b    r28, RED_ADDRESS_PORT(r6)
0cc0e020 0000021c          345   st.b    r28, GREEN_ADDRESS_PORT(r6)
0cc0e040 00000220          346   st.b    r28, BLUE_ADDRESS_PORT(r6)
941c0040 00000224          347   mov     0x40, r28
0cc0e010 00000228          348   st.b    r28, RED_REGISTER_PORT(r6)
0cc0e030 0000022c          349   st.b    r28, GREEN_REGISTER_PORT(r6)
0cc0e050 00000230          350   st.b    r28, BLUE_REGISTER_PORT(r6)
                           351
941c0007 00000234          352   mov     DAC_CONTROL_REG,    r28
0cc0e000 00000238          353   st.b    r28, RED_ADDRESS_PORT(r6)
0cc0e020 0000023c          354   st.b    r28, GREEN_ADDRESS_PORT(r6)
0cc0e040 00000240          355   st.b    r28, BLUE_ADDRESS_PORT(r6)
0cc00010 00000244          356   st.b    r0, RED_REGISTER_PORT(r6)
0cc00030 00000248          357   st.b    r0, GREEN_REGISTER_PORT(r6)
0cc00050 0000024c          358   st.b    r0, BLUE_REGISTER_PORT(r6)
                           359
6c00000d 00000250          360   call    load_default_palette
a0000000 00000254          361   nop
                           362
                           363   //init overlay colours
0cc00000 00000258          364   st.b    r0, RED_ADDRESS_PORT(r6)
0cc00020 0000025c          365   st.b    r0, GREEN_ADDRESS_PORT(r6)
0cc00040 00000260          366   st.b    r0, BLUE_ADDRESS_PORT(r6)
                           367
ef9c0000e41cfc00 00000264  368   mov     RED, r28
0cc00010 0000026c          369   st.b    r0, RED_REGISTER_PORT(r6) //overlay 00
0cc00010 00000270          370   st.b    r0,RED_REGISTER_PORT(r6) //overlay 01
0cc00010 00000274          371   st.b    r0,RED_REGISTER_PORT(r6) //overlay 10
0cc0e010 00000278          372   st.b    r28,RED_REGISTER_PORT(r6) //overlay 11
                           373
a0410000 0000027c          374   mov     r2, r1
40000800 00000280          375   bri     r1
a0000000 00000284          376   nop
                           377
                           378
```

```
                        379
                        380     load_default_palette::
0cc00000 00000288       382     st.b    r0,RED_ADDRESS_PORT(r6)
0cc00020 0000028c       383     st.b    r0,GREEN_ADDRESS_PORT(r6)
0cc00040 00000290       384     st.b    r0,BLUE_ADDRESS_PORT(r6)
                        385
a01c0000 00000294       386     mov     r0,r28
941d0100 00000298       387     mov     256,r29
                        388     next_colour::
                        389     //st.b  r28,RED_ADDRESS_PORT(r6)
                        390     //st.b  r28, GREEN_ADDRESS_PORT(r6)
                        391     //st.b  r28, BLUE_ADDRESS_PORT(r6)
                        392             //store colour to palettes
0cc0e008 0000029c       393     st.b    r28,RED_PALETTE_RAM_PORT(r6)
0cc0e028 000002a0       394     st.b    r28,GREEN_PALETTE_RAM_PORT(r6)
0cc0e048 000002a4       395     st.b    r28,BLUE_PALETTE_RAM_PORT(r6)
                        396
879c0001 000002a8       397     addu    1,r28,r28
539feffb 000002ac       398     btne    r29, r28, next_colour
a0000000 000002b0       399     nop
                        400
40000800 000002b4       401     bri     r1
a0000000 000002b8       402     nop
                        403
                        404
                        405     init_cursor::
                                // set addreg 0 = 0
1ce00000 000002bc       406     st.s    r0,CUR_ADDRESS_0_PORT(r7)
                                // set addreg 1 = 0
1ce00008 000002c0       407     st.s    r0,CUR_ADDRESS_1_PORT(r7)
                                // two command regs
94106464 000002c4       408     mov     0b0110010001100100,     r16
1ce08018 000002c8       409     st.s    r16,CUR_CONTROL_REG_PORT(r7)
94100000 000002cc       410     mov     0x0000,r16  // two cursor x low values
1ce08018 000002d0       411     st.s    r16,CUR_CONTROL_REG_PORT(r7)
94100303 000002d4       412     mov     0x0303,r16  // two cursor x high values
1ce08018 000002d8       413     st.s    r16, CUR_CONTROL_REG_PORT(r7)
94100000 000002dc       414     mov     0x0000,r16  // two cursor y low values
1ce08018 000002e0       415     st.s    r16, CUR_CONTROL_REG_PORT(r7)
94100202 000002e4       416     mov     0x0202,r16  // two cursor y high values
1ce08018 000002e8       417     st.s    r16, CUR_CONTROL_REG_PORT(r7)
                        418
94100000 000002ec       419     mov     0x0000,r16  // two window x low regs
1ce08018 000002f0       420     st.s    r16, CUR_CONTROL_REG_PORT(r7)
94100000 000002f4       421     mov     0x0000,r16  // two window x high regs
1ce08018 000002f8       422     st.s    r16, CUR_CONTROL_REG_PORT(r7)
94100000 000002fc       423     mov     0x0000,r16  // two window y low regs
1ce08018 00000300       424     st.s    r16, CUR_CONTROL_REG_PORT(r7)
94100000 00000304       425     mov     0x0000,r16      // two window y high regs
```

```
1ce08018 00000308      426   st.s    r16, CUR_CONTROL_REG_PORT(r7)
                       427
ee100000e410ffff 0000030c 428   mov    0xffff,r16          // two window width low regs
1ce08018 00000314      429   st.s    r16,CUR_CONTROL_REG_PORT(r7)
94101f1f 00000318      430   mov    0x1f1f,r16          // two window width high regs
1ce08018 0000031c      431   st.s    r16,CUR_CONTROL_REG_PORT(r7)
ee100000e410ffff 00000320 432   mov    0xffff,r16          // two window height low regs
1ce08018 00000328      433   st.s    r16,CUR_CONTROL_REG_PORT(r7)
94101f1f 0000032c      434   mov    0x1f1f,r16          // two window height high regs
1ce08018 00000330      435   st.s    r16,CUR_CONTROL_REG_PORT(r7)
                       436
                       437
                       438   //init cursor pattern
94120200 00000334      439   mov    512,r18
ee100000e410f0f0 00000338 440   mov    0xf0f0,r16
94110f0f 00000340      441   mov    0x0f0f,r17
                       442   nextcurpat::
1ce08010 00000344      443   st.s    r16,CUR_RAM_PORT(r7)
1ce08810 00000348      444   st.s    r17,CUR_RAM_PORT(r7)
9652ffff 0000034c      445   adds   -1,r18,r18
525f07fc 00000350      446   btne   r0,r18,nextcurpat
                       447
40000800 00000354      448   bri    r1
a0000000 00000358      449   nop
                       450
                       451
                       452   fill_screen::
94190100 0000035c      453   mov    DISPLAY_LINES / 4,r25
ef5a0001e41a0001 00000360 454   mov    0x10001,r26
a01b0000 00000368      455   mov    r0,r27
efde0300e41e0000 0000036c 456   mov    VRAM_ADDRESS,r30
941cffff 00000374      457   adds   -1,r0,r28
                       458   next_line::
941d009f 00000378      459   mov    DISPLAY_WIDTH / 8 - 1,    r29
b7a0e001 0000037c      460   bla    r28,r29,fill_line
a3df0000 00000380      461   mov    r30,r31
                       462
                       463   fill_line::
1fe0d801 00000384      464   st.l    r27,0(r31)
1fe0d805 00000388      465   st.l    r27,4(r31)
1fe0d809 0000038c      466   st.l    r27,8(r31)
1fe0d80d 00000390      467   st.l    r27,12(r31)
1fe1d801 00000394      468   st.l    r27,HORIZ_STRIDE+0(r31)
1fe1d805 00000398      469   st.l    r27,HORIZ_STRIDE+4(r31)
1fe1d809 0000039c      470   st.l    r27,HORIZ_STRIDE+8(r31)
1fe1d80d 000003a0      471   st.l    r27,HORIZ_STRIDE+12(r31)
1fe2d801 000003a4      472   st.l    r27,2*HORIZ_STRIDE+0(r31)
1fe2d805 000003a8      473   st.l    r27,2*HORIZ_STRIDE+4(r31)
1fe2d809 000003ac      474   st.l    r27,2*HORIZ_STRIDE+8(r31)
```

```
1fe2d80d 000003b0    475    st.l     r27,2*HORIZ_STRIDE+12(r31)
1fe3d801 000003b4    476    st.l     r27,3*HORIZ_STRIDE+0(r31)
1fe3d805 000003b8    477    st.l     r27,3*HORIZ_STRIDE+4(r31)
1fe3d809 000003bc    478    st.l     r27,3*HORIZ_STRIDE+8(r31)
1fe3d80d 000003c0    479    st.l     r27,3*HORIZ_STRIDE+12(r31)
87ff0010 000003c4    480    addu     16,r31,r31
b7bfe7ee 000003c8    481    bla      r28,r29,fill_line
835bd800 000003cc    482    addu     r27,r26,r27
                     483
87de2000 000003d0    484    addu     HORIZ_STRIDE * 4,r30,r30
8739ffff 000003d4    485    addu     -1,r25,r25
533f07e7 000003d8    486    btne     r0,r25,next_line
                     487
40000800 000003dc    488    bri      r1
a0000000 000003e0    489    nop
                     490
                     491
                     492
                     493
                     494    clear_screen::
941cffff 000003e4    495    adds     -1,r0,r28
efbd0007e41dffff 000003e8  496    mov      VRAM_SIZE / 8 - 1,r29
efde02ffe41efff8 000003f0  497    mov      VRAM_ADDRESS-8,r30
b7a0e001 000003f8    498    bla      r28,r29,clear
a0000000 000003fc    499    nop
                     500
                     501    clear::
b7bfe7ff 00000400    502    bla      r28,r29,clear
2fc00009 00000404    503    fst.d    f0,8(r30)++
                     504
40000800 00000408    505    bri      r1
a0000000 0000040c    506    nop
                     507
                     508
                     509
510 //The following cache flush procedure is from the 860TM programmer's reference
511 //manual. Please reference the manual for additional information.
                     512
                     513    flush_cache::
a0220000 00000410    514    mov      r1,r2
                     515
                     516    FLUSH_P=0x00008000-32
                     517
                     518    //rw=r24,rx=r25,ry=r26,rz=r27
                     519
305b0000 00000414    520    ld.c     dirbase,r27
e77b0800 00000418    521    or       0x800,r27,r27
9419ffff 0000041c    522    adds     -1,r0,r25
6c000008 00000420    523    call     D_FLUSH
```

```
3840d800 00000424     524   st.c     r27,dirbase
e77b0900 00000428     525   or       0x900,r27,r27
6c000005 0000042c     526   call     D_FLUSH
3840d800 00000430     527   st.c     r27,dirbase
f77b0900 00000434     528   xor      0x900,r27,r27
                      529
a0410000 00000438     530   mov      r2,r1
40000800 0000043c     531   bri       r1
3840d800 00000440     532   st.c     r27,dirbase
                      533
                      534   D_FLUSH::
e4187fe0 00000444     535   or       l%FLUSH_P,     r0,r24
ef180000 00000448     536   orh      h%FLUSH_P,     r24,r24
e41a007f 0000044c     537   or       127,r0,r26
171f0021 00000450     538   ld.l     32(r24),r31
a7ff0000 00000454     539   shl      0,r31,r31
b740c801 00000458     540   bla      r25,r26,D_FLUSH_LOOP
a0000000 0000045c     541   nop
                      542
                      543   .align   32
                      544
                      545   D_FLUSH_LOOP::
08000000 00000460     546   ixfr     r0,f0
b75fcffe 00000464     547   bla      r25,r26,D_FLUSH_LOOP
37000021 00000468     548   flush    32(r24) + +
08000000 0000046c     549   ixfr     r0,f0
                      550
40000800 00000470     551   bri      r1
1700fe01 00000474     552   ld.l     -512(r24),r0
                      553
a0000000 00000478     554   nop
a0000000 0000047c     555   nop
a0000000 00000480     556   nop
a0000000 00000484     557   nop
                      558
 00000488             559   .end
```