ALACRON

RT860 Software Reference Manual

30002-00110

Notice

Alacron, Inc.
71 Spitbrook Road, Suite 204
Nashua, NH 03060 USA

telephone: (603)891-2750
fax: (603)891-2745
electronic mail:
sales@alacron.com
support@alacron.com

Document Name: RT860 *Software Reference Manual*
Alacron Part Number: 30002-00110
Document Number: 02-00110.DOC

Revision History:

1.0	01-Aug-95	1.4	31-Jul-96
1.1	22-Aug-95	1.4.1	13-Dec-96
1.2	15-Sep-95	1.4.2	19-Dec-96
1.2.1	30-Sep-95	1.4.3	31-Dec-96
1.3	19-Jul-96		

Table of Contents

Table of Contents

Preface

Purpose of This Manual

The *RT860 Software Reference Manual* describes a list of functions that Alacron provides to software developers for the purpose of application development. Both host and processor board functions are included. This manual is a comprehensive reference for the Slave Mode and processor board libraries for all Alacron processor board products.

New users may have a difficult time understanding the information presented in this text without first obtaining a general understanding of the Alacron processor board and it's software architecture. Please refer to the *RT860 Software User's Guide* for a conceptual overview of the RT860 runtime software and the development environment.

Audience

This reference manual is intended for technical personnel responsible for developing application software that uses an Alacron processor board. This manual assumes familiarity with the C and FORTRAN programming languages which are used to develop the applications that execute on the Alacron processor board.

Manual Organization

The *RT860 Software Reference Manual* is divided into the following sections:

Chapter 1, "RT860 Release Notes" provides software release notes for RT860. Release notes are separated according to processor board.

Chapter 2, "Utilities " lists software utilities that are offered with the RT860 Runtime Software.

Chapter 3, "Host Slave Mode Library Reference " provides a reference of all functions in the Slave Mode library.

Chapter 4, "Processor Board Library Reference" lists the functions that are callable by programs running on the processor board.

Chapter 5, "Multiple Processor Extensions" contains a reference of the functions callable on the processor board that support multiple processors.

Chapter 6, "PCI Processor Board DMA Library" contains a reference of the DMA support functions that are available for PCI processor boards.

Appendix A, "System call support" discusses system call support on processor boards and provides a table of supported system calls.

Appendix B, “Getting Help,” contains suggestions on how to obtain technical support from Alacron.

Notation Conventions

Alacron documentation uses *italics* to introduce or define new terms. For example:

> The DIO daughtercard is intended to be used in conjunction with additional hardware, a *mezzanine card*, which handles the interface to some external device or bus.

Boldface is used to emphasize words within the text. For example:

> The user **must** take precaution against damage from static electricity when handling the processor board.

`Computer font` is used to represent text that would appear on your display screen. For example, the user may change directory into the default FT200 directory tree by typing the following:

```
cd \usr\ft200
```

`[ Square brackets ]` are used to indicate that any of a set of characters may be used . For example, the functions alsetiresult, alsetfresult and alsetdresult are collectively abbreviated by the expression:

```
alset[ifd]result
```

Related Information

This manual assumes that you are already familiar with the information in the user’s manual for your PC or workstation. In addition, the following documents provide additional reference information:

- *AL860-AT and FT200-AT Hardware User’s Guide*
- *AL860-PCI and FT200-PCI Hardware User’s Guide*
- *FT200-V Hardware Reference Manual*
- *RT860 Software User’s Guide*
- Intel Corporation’s *i860Ô Microprocessor Family Programmer’s Reference Manual*

1

RT860 Release Notes

This chapter provides software release notes for the RT860 runtime software. Release notes are grouped according to processor board. Please contact Alacron Technical Support if you have any questions related to the information shown here.

AL860-PCI and FT200-PCI

Version 1.9.2

Enhancements - Bug Fixes

- The user diagnostics have been improved to be more robust, and test more functionality.
- The TSR now performs only 32-bit writes to the board's DRAM. This is to work around a problem in the PLX PCI interface chip. Thus writes of partial long words result in a 32-bit read, followed by the modification of the appropriate bytes, followed by a 32-bit write. Reads are not affected. Applications should not align data such that the on board DRAM is corrupted by this cycle. (Shared data should be 32-bit aligned, and a multiple of 32-bits.)
- A potential DMA problem involving host interference on the PCI bus is overcome using a work-around provided in the EEPROM.MAP file. All boards in the field should re-run FTINIT to update the serial EEPROM device.

Notes - Warnings

- This installation requires version 6.22 or later of MS-DOS if EMM386 is being used.
- The AL860 environment variable needs to be set to the appropriate host base directory: C:\USR\ALACRON\MSDOS for Microsoft C support, and C:\USR\ALACRON\WATCOM for WATCOM support. The PATH variable should include %AL860%\BIN (for WATCOM you should also append C:\USR\ALACRON\MSDOS\BIN to get utilities for which WATCOM executables are not provided).

FT200-AT Software Release Notes

- It is recommended that the MS-DOS 6.22 MEMMAKER utility be run after making the AUTOEXEC.BAT changes to move the TSR into upper memory.

FT200-AT Software Release Notes

-

Version 1.9.1

Enhancements - Bug Fixes

- The 1.9.1 release contains some cosmetic modifications, and the only known problem in 1.9 was verified as having been caused by using an obsolete validation program.

Version 1.9

Enhancements - Bug Fixes

None

Notes - Warnings

- The same RT860 media supports the AL860-PCI and FT200-PCI.
- RT860 was built using Portland Group C version 2.3 and MSVC 1.5.

FT200-AT Release Notes

The following section lists enhancements, bug fixes, notes and warnings for the FT200-AT and FT200-V releases. Please read this section before attempting to use the FT200 card. The releases are listed here in reverse chronological order. In other words, the most recent releases are listed first.

Version 1.12.1

Enhancements Bug Fixes

- This version now includes a Window 3.11 capable library (w3slv860.lib).

Notes - Warnings

- Only Microsoft C8/DOS (slv860.lib), Microsoft C8/Win3 (w3slv860.lib) and Watcom version 10.x (slv860wc.lib) are supported in this release.

The following FT200-AT Runtime internal part numbers have been updated to version 1.12.1. You may match these part numbers to the one contained in the ALINFO directory that is installed by the Runtime software:

Part Number	**Operating System**	**Release Type**	**Media**
40000-00222	MSDOS	Object	**3.5” floppy**
40001-00199	MSDOS	Source	**3.5” floppy**

Version 1.12

Enhancements Bug Fixes

- This version represents a conversion to the standard ISA/PCI unified source tree. This version fixes a problem with almapload executing an incorrect board reset/unreset pair - potentially causing problems in program execution.

Notes - Warnings

- There is no support in this release for a Microsoft Windows capable DLL.
- Only Microsoft C8 and Watcom version 10.x is supported in this release.

The following FT200-AT Runtime internal part numbers have been updated to version 1.12. You may match these part numbers to the one contained in the ALINFO directory that is installed by the Runtime software:

Part Number	**Operating System**	**Release Type**	**Media**
40000-00222	MSDOS	Object	**3.5” floppy**
40001-00199	MSDOS	Source	**3.5” floppy**

Version 1.7

Enhancements - Bug Fixes

- A bug was fixed in the way that stacks are managed within the mpkernel. Before this release, certain calls to user functions on the i860 board were able to overflow the user stack and run on the Kernel stack.
- An instance was found in which the Kernel moved the stack by 12 bytes instead of 16. This was fixed.
- A fix was made to the install program so that the Runtime software could be installed on a Novell network drive.

Notes - Warnings

The following FT200-AT Runtime internal part numbers have been updated to version 1.7. You may match these part numbers to the one contained in the ALINFO directory that is installed by the Runtime software:

Part Number	**Operating System**	**Release Type**	**Media**
40000-00222	MSDOS	Object	**3.5” floppy**
40000-00223	OS/2	Object	**3.5” floppy**
40000-00224	Solaris x86	Object	**3.5” floppy**
40001-00199	MSDOS	Source	**3.5” floppy**
40001-00200	OS/2	Source	**3.5” floppy**
40001-00201	Solaris x86	Source	**3.5” floppy**

FT200-AT Software Release Notes

Version 1.6

Enhancements - Bug Fixes

- The Runtime software now correctly sizes and works with 272 Megabytes of memory.
- User Diagnostics (FT2DIAG): Changes were made to correct time-out problems with tests on large memories. A runtime stack overflow error was corrected. Sources for FT2DIAG are now included in the source release. Other cosmetic changes were also made.
- The i860XP trap handler and floating point exception handler source files are now included in the source distribution.
- A problem with the system calls time() and times() was fixed. Previously, these functions would not always return results that were monotonically increasing.
- Several make files for Solaris (sun5x) in the examples area have been fixed.
- The file include/alhost.h in the Alacron software runtime directory tree defined the C language keyword const to the null string. This resulted in compile errors under Solaris 2.4 with function prototype mismatches. In this release, this #define has been removed.
- The make files were fixed so that the make clean command would perform cleans correctly and would recurse sub-directories correctly.
- The doc files in the example subdirectories have been updated to reflect the latest make and run models.

Notes - Warnings

- The FT200-AT Runtime Software Version 1.6 was built with the tools listed below. If you have different versions of tools and are experiencing difficulty please contact Alacron customer support for assistance.

MSDOS Host C compiler	Microsoft C version 8.0 (MSVC 1.5) Watcom C version 10.0
MSDOS Host Assembler	Microsoft MASM version 5.1
i860 C compiler	Portland Group C version 2.3
Unix/386 Reference	Interactive Systems 3.2 Release 2
Solaris 2/x86 version 5.4	GCC version 2.5.7
OS/2 version 2.10	CSET2++, version 2.01

- When using Microsoft Windows:
    1. The make file include file tools\include\incl.c7w or tools\include\incl.c8w must be modified to reflect the correct Microsoft C compiler directory area being used.
    2. The MS-DOS PATH environment variable must include both the bin and lib directories.
    3. The FT200AT.INI file must be correctly setup in the Windows directory. See the section on the W3Slv860Setup() function for more information.
    4. To compile and link the Windows example code in src\examples\w3lib, please use the make files make.c7w or make.c8w. You may need to modify these files to have them reference the correctly default directories.
- Functions alsetba(), alsetwa(), algetba(), algetwa(), and algetla are documented as returning integer values indicating SUCCESS or FAILURE. These functions are incorrectly implemented as void functions which do not return a value. Please treat these functions as void functions, or alternatively, refrain from error checking their return values.
- The functions read860() and write860() are documented as returning the number of bytes read or written or -1 to indicate an error condition. These functions actually return -1 (FAILURE) on an error and zero (SUCCESS) if there are no errors.
- Functions almapload() and alload() cause the application to exit() if they detect an invalid COFF file.

Version 1.6.B.3

Enhancements - Bug Fixes

None

Notes - Warnings

- A bug has been found which prevents this release from being used with FT200-AT boards that have 272 MB. Boards with less memory that 272 MB work OK.
- If RTC interrupts are enabled, the values returned by time() and times() system calls are computed from a count of RTC interrupts and reading of the 32 bit µsec counter. If the 32 bit counter is read just after it rolls over, but before the interrupt has been services, an incorrect time value is computed. The error in the time value is no greater than 1/freq (where freq is the RTC frequency). The symptom of the problem is that a call to the function to get the time returns a value less than on a previous call.
- The **mp_thread_start()** function does not error check the processor parameter passed. It will not reject illegal values. The function will default to using the *other* processor.
- The Slave Mode code has instances in which error codes are incorrectly passed back to the user program due to the use of the exit() system call.
- Functions read860 and write860 return 0 for success and -1 for failure. The number of bytes read is not returned.
- The **aldev** function does not properly error check the *dev* argument. Currently, function **aldev** may be called on a non-opened device.

FT200-AT Software Release Notes

Version 1.6.B.1

Enhancements - Bug Fixes

- Full support has been added for the FT200-AT running on an ISA OS/2 system. For the OS/2 installation procedure, see the *AL860-AT and FT200-AT Hardware User's Manual* (part number 30002-00100).
- MSDOS Slave Mode libraries have been provided for use with the Watcom C Compiler. This facilitates development of 32 bit host code under MSDOS. The following libraries are provided:

slv860wc.lib	Slave Mode library
dos860wc.lib	Driver Interface Library
rt860wc.lib	RT860 library for user extensibility

- In FT200 software version 1.6, the Watcom libraries do not make use of hardware interrupts from the FT200 board (a simple polling mechanism is used). Make files are provided in the slave mode examples (src/examples/ex*) that use the Watcom libraries (make.wc).
- The function malloc() (in file mphost/w3setup.c) now has proper string length computation.
- The i860lib functions mp_thread_start() and mp_thread_query() have been substantially modified to fix problems resulting from improper stack utilization. The use of inter-processor interrupts has been eliminated.

WARNING: i860 applications utilizing thread functions MUST be recompiled for use with Version 1.6

- The floating point signal exception handler (SIGFPE) has been modified to operate correctly.
- Use of the function prototypes in mphost.h now have API declarations to facilitate use with Windows application and other development environments. API will be defined as required depending on the host operating system and C compiler. To avoid syntax errors when compiling, the appropriate C preprocessor symbols (described in this release notice) must be used.
- If a call to Slave Mode functions **alcall()** or **mpcall()** terminates for abnormal reasons (floating point trap, page fault, etc.), the host may issue further calls to the FT200. In the past the system would hang with no recourse other than

perhaps an **almapload()**. After an abnormal termination, the Slave Mode program may call the functions **algetstatus(), mpgetstatus(), alprtexitstatus(), mpprtexitstatus()** to obtain exit status information.

- The RTC_ATTACHINT command of function of **rtcioctl()** may be called to attach multiple interrupt handler functions. The command RTC_DETACHINT is added to detach a particular function.
- The functions umalloc(), upmalloc(), and ufree() have been implemented using the standard C library function malloc(). Address aliasing is used to convert cached addresses to uncached.
- Windows applications writing to stdout and stderr, sometimes printed residual characters from previous writes. This problem is now resolved.
- The following new functions have been implemented:

    1. algetstatus ()
    2. mpgetstatus ()
    3. alprtexitstatus ()
    4. mpprtexitstatus ()
    5. alinvalidate_caches ()

Descriptions of these functions may be found in the reference section of this manual.

- Numerous source code changes have been made for enhanced code portability.
- Support is now provided for FT200-AT boards equipped with the maximum of 272 Mbytes.
- To implement 272 Mbyte memory sizes, the write protected 4 Mbyte page aliases windows were removed. With 1.6, the following virtual to physical mappings are made:

Virtual address	**Physical address**	**Cache policy**
B0000000	FF000000	write back cached
C1000000	FF000000	write through cached
D2000000	F0000000	uncached

The following socket functions are now available to i860 applications running under Unix hosts:

gethostbyname ()
inet_addr ()
sendto ()
send ()
accept ()
bind ()
connect ()
getpeername ()
getsockname ()
listen ()
recv ()
recvfrom ()
select ()
setsockopt ()
getsockopt ()
shutdown ()
socket ()
gethostname ()

To utilize these functions, the i860 application must link to the file **`lib/i860lib.a`**.

Host Slave Mode applications must include host libraries supporting sockets. In the case of SYSV386, the following should be appended to the host application link command:

```
-linet
```

For Solaris 2/X86 no additional link directives are required.

Notes - Warnings

- The FT200-AT Run-Time Software Version 1.6.B.1 (updated August 16, 1994) was built with the tools listed below. If you have different versions of tools and are experiencing difficulty please contact Alacron customer support for assistance.

MSDOS Host C compiler	Microsoft C Version 8.0 (MSVC 1.5) Watcom C Version 10.0
MSDOS Host Assembler	Microsoft MASM Version 5.1
i860 C compiler	Portland Group C Version 2.3
Unix/386 Reference	Interactive Systems 3.2 Release 2
Solaris 2/x86	Version 2.1
OS/2	CSET2++, version 2.0

- A new directory structure has been implemented in order to integrate the slave mode libraries of various operating systems under one directory tree. The default installation directory for the FT200 software remains `/usr/ft200` (or `\USR\FT200`). The bin directory underneath `/usr/ft200` will now only contain i860 executable files. Host executable files have been moved into separate directories for each operating system. The following table shows the host executable directories for currently supported hosts:

Host Operating System	Directory of Host Executable Files
Unix System V/386	`/usr/ft200/sysv386/bin`
Solaris 2/X86	`/usr/ft200/sun5x/bin`
OS/2	`\USR\FT200\OS2\BIN`
MSDOS	`\USR\FT200\MSDOS\BIN`

Note that the **AL860** environment variable must now be set to the appropriate host operating bin directory shown in the table above. This directory must also be placed in the shell **PATH** so that host executable files may be referenced without regard to the user's current working directory.

- The user diagnostics are now invoked using the command *ft2diag* instead of *userdiag*.
- Sources are included for those user's that have obtained a source license. Each source directory tree contains make files which build the Slave Mode libraries, i860 libraries, and i860 Kernel. The following table summarizes the names of make files provided in this release:

make.sysv386	Unix 386 make file
make.sun5x	Solaris 2/x86 make file
make.c8	Microsoft C Version 8
make.c8w	Microsoft C Version 8, Windows
make.wc	Watcom C
make.i860	i860 make files for use with Unix make
nmake.860	i860 make files for use with Microsoft NMAKE

Note, all MSDOS make files (including Watcom) are written for use with the Microsoft NMAKE utility.

- Since the Slave Mode libraries now support several environments, the Slave Mode library include files now contain conditional code for each environment or host operating system. The user is required to define preprocessor symbols on the command line during Slave Mode program builds. The following table summarizes the defines required for each environment or operating system:

MSDOS using Microsoft C Version 8	-DMSDOS -DMSC
MSDOS using Watcom C	-DMSDOS -DWATCOM
MSDOS using Microsoft C, Windows	-DMSDOS -DMSC -DW3
SYSV386	-DUNIX -DSYSV386
Solaris 2/x86	-DUNIX -DSUN5X
OS/2	-DOS2

- Host slave mode applications now need only include the file **allib.h** to define all data types, configuration parameters, and library function prototypes.
- Use of function prototypes in mphost.h now have API declarations to facilitate use with Windows applications and other development environments. API will be defined as required depending on the host operating system and C compiler. To avoid syntax errors when compiling, the C preprocessor symbols described above must be used.
- All i860 applications need only include the file **i860lib.h** to define all data and function prototypes required when using the library i860lib.a.
- A fatal error now occurs if the **AL860** environment variable is not defined.

- The function **W3Slv860Setup()** is provided as part of the Windows Slave Mode DLL to initialize the library. With version release 1.6, the calling sequence and semantics have been changed as follows:

```
extern API W3Slv860Setup (HWND hWnd)
```

The DLL now establishes it's required environment variables by reading the file **FT200AT.INI**, which should be created in the windows main directory (the default is C:\WINDOWS). The format of the file is as follows:

```
        [environment]
        AL860=C:\USR\FT200\MSDOS
        AL860XPCFG=(0,110,12)
```

The environment variables **AL860** and **AL860XPCFG** have retained the same semantics as in the past. If the W3Slv860Setup function ***hWnd*** argument is non-null, then standard error and standard output are formatted and displayed on the window given by handle ***hWnd***. If ***hWnd*** is null, then standard error and standard output are directed to a windows debug screen using the windows SDK function **OutputDebugString()**.

It is now optional to call **W3Slv860Setup()** in a windows application utilizing the slave mode DLL.

- The language support tools (as860, ld860, ar860, nm860, etc...) are no longer provided in the ALACRON Run-Time software directory. As in past releases, the user should use the versions of these tools that are distributed with the Portland Group C and FORTRAN compilers.
- The SYSV386 device driver is modified to make reference to 'ft200' rather than 'xp860'. This is reflected in the device special files (/dev/ft200*n*) and installation procedure. There are no functional changes to the driver.
- The Solaris 2/x86 driver is modified to make reference to 'ft200' rather than 'xp860'. This is reflected in the device special files (/dev/ft200*n*) and installation procedure. There are no functional changes to the driver. For driver installation instructions please refer to the manual *AL860-AT and FT200-AT Hardware User's Manual*, (Part Number: 30002-00100).

FT200-AT Software Release Notes

Version 1.5

Build Tools Release Level

MSDOS Host C compiler	Microsoft C Version 7.0 Watcom C Version 9.0
MSDOS Host Assembler	Microsoft MASM Version 5.1
i860 C compiler	Portland Group C Version 2.2
Unix/386 Reference	Interactive Systems 3.2 Release 2

Warnings:

- On releases of FT200-AT Run-Time 1.5 or earlier, a user interrupt handler can call certain functions which cause the i860 trap handler to fail. A hardware interrupt (from a daughtercard, the host, the real-time clock, or another processor) causes the i860 to save state and switch context to the i860 trap handler. If enabled and registered correctly, the trap handler will invoke a user specified interrupt handler. If the user handler calls a function which reenters the trap handler, then the trap handler will not correctly return to the application's main thread of control. Examples of functions that could reenter the trap handler follow: printf(), mp_printf(), flush(), inthost() or user extensibility functions. The user handler is restricted from calling functions which require host support as is the case for many C run-time library functions.

Changes with release 1.5:

- fixed bug in mphost/w3setup.c, call to malloc() now has proper string length computation.
- modified signal(SIGFPE) for proper operation
- Host C compiler defines

  When using the host include files (allib.h, mphost.h, etc.) it is necessary that preprocessor symbols be defined to specify the host O/S and C compiler. The following table summarizes the required defines.

MSDOS using Microsoft C Version 7	-DMSC -DMSDOS
MSDOS using Watcom C	-DWATCOM -DMSDOS

MSDOS using Microsoft C, Windows	-DMSC -DMSDOS -DW3
Unix386	-DUNIX

- mphost.h - function prototypes have FAR CDECL declarations. Use of the function prototypes in mphost.h now have FAR and CDECL declarations to facilitate use with Windows application development. To avoid syntax errors when compiling, the C preprocessor symbols described above must be used.
- AL860 environment variable must be defined. A fatal error now occurs if the AL860 environment variable is not defined
- Language support tools (as860, ld860, ar860, nm860, ...) are not provided. The language support tools are no longer provided with the ALACRON Runtime software. Use those distributed with the Portland Group C and FORTRAN compilers version 2.2.
- Watcom Support: Slave mode libraries are now provided for use with the Watcom C compiler. This facilitates development of 32 bit host code under MSDOS. The following libraries are provided:

slv860wc.lib	Slave Mode library
dos860wc.lib	Driver Interface Library

In FT200 software version 1.5, the Watcom libraries do not make use of hardware interrupts from the FT200 board (a simple polling mechanism is used).

Make files are provided in the slave mode examples (src/examples/ex*) that use the Watcom libraries (make.wc).

Make file naming

The following table summarizes the names of make files provided in this release:

makefile	Unix 386 make file
make.c7	Microsoft C Version 7
make.c7w	Microsoft C Version 7, Windows
make.wc	Watcom C
*.nmk	i860 or generic makefile for MSDOS

- In release 1.5, all MSDOS make files (including Watcom) written for use with Microsoft NMAKE.
- Multiple calls to RTC_ATTACHINT may be made. The RTC_ATTACHINT function of **rtcioctl()** may be called to attach multiple interrupt handler functions. The command RTC_DETATCHINT is added to detach a particular function.

- New umalloc, upmalloc, ufree: The functions umalloc, upmalloc, and ufree have been implemented using the standard C library function malloc, and address aliasing to uncached addresses.
- Added i860lib.h: Function prototypes and parameter definitions for all i860 functions in i860lib.a (including rtc, at, ...) are declared in the new include file **i860lib.h**.
- Windows DLL writes to stdout, stderr fixed: A bug was fixed where i860 writes to stdout and stderr that are redirected to a window were picking up data from previous calls.

FT200-AT Software Release Notes

Version 1.4

Changes with release 1.4:

- Added all DOS based makefiles & windows/ICB.
- Added mpstatus( ) host library function
- Corrected bugs in mphost/wait.c dealing with multi-board/processor waits.
- Corrected proc's missing printf in mphost/mapload.c
- Corrected printf/errexit parameters to longs
- Converted macro-fied functions, back into functions for compatibility purposes.
- Took out UNPROTO from the MSDOS releases.
- Added NMAKE files, removed makit.bat and dmakefil files.
- Revised user extensibility files.
- Fixed (possible) race condition in interrupt servicing routine.
- Cleaned up (DOS) ioctl860 routine, with respect to interrupt disabling.
- Moved PIC clearing into cdosp860\dos860.c\int860() for DOS release.
- Fixed large block writes for DOS.
- Added M860_INTWAITA ioctl, for determining which processor of the board interrupted.
- Fixed potential memory leak problem in the Kernel.
- Fixed I/O buffer problem in syscall.c for DOS.
- Provided more support for PGI debugger.
- Added mas860 to release.
- Fixed extioctl function prototype and code.
- Fixed IDLE/RUN bug in mpkernel/proc.c - found by mpstatus().
- Fixed some code which produced MSC warnings.

Errata:

- Preloading and Kernel initialization are not yet implemented.
  - Kernels are loaded each time application is executed, no rt860 -i required.
- Certain mis-aligned data traps are not correctly handled by the Intel XP trap handler. Thus, all mis-aligned data will be flagged as such, regardless of the rt860 flag '-a'.

- The i860 assembler provided does not contain the XP specific instructions, including the pfld.q instruction.

    - This has been fixed by PGI in release 2.0+

- TRAP handler has problems with quads - traps when doing stuff with pfld.q may cause incorrect data to be loaded into the pipeline.
- (Not an errata, just a message) Do not alias already cached data to uncached addresses, as once a physical address is recognized in the cache, it will override any uncached accesses to that physical address.
- DOS only: The PGI FORTRAN compiler has been known to cause the i860XP to "hang" under high optimization levels on certain programs. This is currently being worked on, and no work-around is suggested, other then to compile at a lower optimization level, or use the cross compiler.

FT200-V Release Notes

Version 1.9.2

Enhancements - Bug Fixes

- Added bd$attach_vmebus_intr call:

```
void bd$attach_vmebus_intr( int (*handler)( int vector ) );
```

This routine will setup a VMEbus interrupt handler - providing interrupt handling for any enabled IRQ level. The handler is invoked when any enabled IRQ is asserted, the vector is acquired via a standard VMEbus interrupt acknowledge cycle, and the handler function is invoked. The function should return a non-zero value if it is an expected vector to be handled, else 0 if the vector is not expected.

Restrictions / Notes:

1. The handler function must observe all other rules concerning interrupt-time notification calls (one should not perform system calls in the callback routine).
2. The following vector ranges cannot be generated on any enabled IRQ, as these are reserved for on-board vectors: 0xA0-0xBF, 0xD0-0xDF.
3. It is the applications responsibility to ensure that any possible VME IRQ handlers are disabled from generating interrupt acknowledge cycles for the requested interrupt levels. [In particular: the Slot 1 (or Monarch) controller must have each IRQ disabled that is required by the application.]
4. To enable a specific IRQ level use: **int bd$enable_vme_irq(int level)** - where level is the request IRQ level to enable. The interrupt level that the board is setup to generate interrupts on must not be enabled.
5. To generate an interrupt on the VMEbus (other than the IRQ/vector specified for the board), use: **void bd$setup_vme_int(int level, int vector)** call to setup the vector for the level required, and: **void bd$gen_vme_Int(void)** to actually generate the interrupt.
6. At the present time, there is no way to convey the IRQ along with the vector to the handler - do not use the same vector for different IRQ levels.

- Added the bd$vme_dma_asynch call:

```
void bd$vme_dma_asynch( unsigned long phys_dram_addr, unsigned
long virt_vme_addr, int in, int nbytes, void (*callback)(
unsigned short status ) )
```

This routine will cause a data transfer to be started using the on-board block transfer mechanism. The phys_dram_addr provides the on-board buffers physical address (this buffer is usually allocated via the umalloc call, the return value of which can be passed to vtop to obtain the physical address). The virt_vme_addr is the virtual address of the remote VME resource, this address is usually obtained via the mapvtop call. [Note: All of the remote resource must be mapped in.] The in parameter indicates whether the transfer is from the board to the remote VMEbus resource (in == 0), or if the transfer is from the remote VMEbus resource to the on-board buffer (in != 0). nbytes indicates how many bytes of data is to be transferred. The callback function is invoked when the transfer has completed. The callback function receives a 16 bit parameter, with bits 0-7 containing the VIC Bus Error Status Register value, and bits 8-15 containing the VIC DMA Status Register value. Normal completion of the transfer is indicated when bits 8-15 are clear.

Restrictions:

1. All of the remote VMEbus resource must be mapped into the 860's virtual address space.
2. nbytes must be greater than 0
3. Interrupts must be enabled in order for completion of the transfer to be detected.
4. The callback function must observe all other rules concerning interrupt-time notification calls (one should not perform system calls in the callback routine...)
5. This routine must not be invoked if there is an outstanding transfer being performed.

Version 1.9.1

Enhancements - Bug Fixes

- The time100 host routine was added to the Solaris distribution
- The times system call was fixed.

Notes - Warnings

- This release of RT860 was built using the following products and revisions:

Solaris 2/Sparc	Version 2.4
Portland Group Compiler	2.3-1

- The following part numbers have been modified:

40000-00200	1.9.1	RT860 FT200-V runtime, Solaris 2/Sparc - Object
40001-00195	1.9.1	RT860 FT200-V runtime, Solaris 2/Sparc - Source

Version 1.9

Version 1.9 of FT200-V software provides support for VxWorks operating systems, as well other minor changes.

Part Number Information

FT200-V Software Release Notes

40001-00276	1.9	RT860 FT200-V runtime, VxWorks - Source

Software Tools/Revisions

VxWorks	Version 5.3/Tornado
Portland Group Compiler	2.3-1

Enhancements - Bug Fixes

- fixed problems in mpdump utility pertaining to location of global structure, and MMU tables

VxWorks Notes

The VxWorks release of RT860 software is made in source form only, without binaries. The user must recompile the RT860 libraries and utilities prior to being able to use the FT200-V board and software.

The reader is referred to the VxWorks software installation section of the *Processor Board Installation Manual (Getting Started)* manual (Alacron part number 30002-00112) for details on installing, configuring and compiling RT860 for VxWorks.

- This release is untested on more than one FT200-V board
- the **talkft** utility is not provided. The user must configure the board using the front panel serial port.
- filesystem I/O using NFS is faster than ftp/rsh alternatives

  Certain RT860 functions perform disk I/O, and run faster if the VxWorks environment makes use of NFS file system access.
- host environment is not automatically exported to VxWorks target

  Since RT860 software requires environment variables, the application must manual export required environment variables by invoking the VxWorks function ***putenv()***.
- library calls to printf and exit

  The RT860 software may, under error conditions, direct printf output to stdout and stderr and call the exit library function. This may have the appearance of hanging the VxWorks shell task, or otherwise terminating the application task
- Limited i860 System Call Support

  The following i860 system calls are supported in the VxWorks version of RT860.

system call	comments

access	implemented using stat - only returns access rights based on “other” file permissions
chdir	vxWorks requires a full pathname
close	
creat	vxWorks may ignore mode argument
exit	invokes vxWorks exit(), which hangs windsh
fstat	
mkdir	creates permissions and owner that may be unexpected
open	
read	
rename	not presently usable (PGCC library turns rename() into link/unlink
rmdir	
seek	
stat	
time	vxWorks will need time of day properly initialized
times	returns relative time in 1/100th seconds, all execution time is assigned to tms_utime
ulimit	implemented for ulimit(3) only (returns max break)
uname	
unlink	
write	

- Manually loading the kernel

  Normally the **rt860** program is used to load the FT200-V kernel, which must be done prior to loading and running application code. It may not be desirable for a VxWorks application to keep a copy of **rt860** around to run. In this instance the function **do_load_kernel()** may be invoked following calls to **alopen()** and **aldev()** and before calls to **almapload()**.

- ft2diag terminates with exit()

  When running **ft2diag**, the program terminates following any series of tests with a call to exit(), which appears as a hang. This is expected behavior.

- memory leaks

  Repeated loading and running of RT860 application modules will result in a memory leak originating from space allocated from calls to almapload().

Version 1.8.1

Enhancements - Bug Fixes

- A Solaris device driver problem involving an incorrect implementation of the ALWAITANY ioctl was fixed..

Notes - Warnings

- This release of RT860 was built using the following products and revisions:

Solaris 2/Sparc	Version 2.4
Portland Group Compiler	2.3-1

- The following part numbers have been modified:

40000-00200	1.8.1	RT860 FT200-V runtime, Solaris 2/Sparc - Object
40001-00195	1.8.1	RT860 FT200-V runtime, Solaris 2/Sparc - Source

Version 1.8

Enhancements - Bug Fixes

- The mpdump utility was added.

Notes - Warnings

- This release of RT860 was built using the following products and revisions:

Solaris 2/Sparc	Version 2.4
Portland Group Compiler	2.3-1

- The following part numbers have been modified:

40000-00200	1.8	RT860 FT200-V runtime, Solaris 2/Sparc - Object
40001-00195	1.8	RT860 FT200-V runtime, Solaris 2/Sparc - Source
40002-00064	2.1.1	PGI Debugger for FT200-V RT860 1.8, Solaris 2/Sparc - Object

FT200-V Software Release Notes

Version 1.6

The following tools were used to build this release:

Host OS/Environment	Host C Compiler	i860 Compiler
LynxOS	GNU	Portland Group C Version 2.3 - Native
SunOS 4.1.3	cc	Portland Group C Version 2.3 - Cross

Enhancements - Bug Fixes

- The Runtime software now supports 272 Megabytes of memory.
- The User Diagnostic program has been renamed **FT2DIAG**. Substantial modifications have been made to the new user diagnostics. The FT2DIAG sources are now included in the source release.
- The i860XP trap handler and floating point exception handler source files are now included in the source distribution.
- The functions mp_thread_start and mp_thread_query have been substantially modified to fix problems resulting from improper stack utilization, and to eliminate the need for interprocessor interrupts to kick off threads. As a result of this modification, i860 applications which utilize the FT200 thread functions must be recompiled and relinked using release 1.6.
- The umalloc(), upmalloc() and ufree() functions have been reimplemented using the underlying Portland Group malloc() and free() C library functions as well as the address aliasing feature of the i860XP. Applications which use these functions must be recompiled and relinked.
- During the implementation for support of 272 MB FT200-V boards, the write protected aliased areas were removed. The available options to mp_4meg_addr() are: MP_KADDR_WBC, MP_KADDR_WTC and MP_KADDR_UNC.
- The following socket system calls and library functions are now implemented: gethostbyname, inet_addr, sendto, send, accept, bind, connect, getpeername, getsockname, listen, recv, recvfrom, select, setsockopt, getsockopt, shutdown, socket, gethostname.
- The seek860 function now checks addresses passed to ensure that invalid VMEbus cycles are not performed.
- The i860 function ismapped() now correctly handles 4 MB pages. It previously reported that all 4 MB pages were not mapped.
- The RTC_ATTACHINT command to rtcioctl may be called to *attach* multiple interrupt handler functions. The command RTC_DETACH has been added to *detach* a specified function. Note that all attached functions are invoked

on a Real-Time Clock interrupt if and only if the RTC_ENABLE command has been issued.

- LynxOS only: Previous versions of the LynxOS driver did not contain the ioctl function support for I860_DEASSERT_RESET.
- LynxOS Only: Utilities bison and flex, supplied with LynxOS, did not create the mklib program correctly. This program is part of the mpmklib functionality. The supplied mpmklib and mklib program now work correctly as the yacc and lex source files are compiled on a working host. The source distribution now contains the yacc and lex output files.
- LynxOS only: The I860_GETINTS function previously cleared interrupts upon return. This has been corrected and certain alwaitany() and alintstat() problems have been fixed as a result.

Notes - Warnings

- LynxOS Specific Notes:

  1. Host programs should use the preprocessor definition -DLYNX when building.
  2. The default directory for the Runtime software is `/usr/ft200`. The AL860 environment variable should be set to `/usr/ft200/lynx`. The PATH environment variable should be modified to reference `/usr/ft200/lynx/bin`. Host applications should link with library slv860.a. If socket support is desired, the host program should be linked with the flag `-lbsd`. All programs that run on the 860 should like with `lib\i860lib.a.`
  3. Make files have the name *make.lynx*.

- SunOS/Solaris Specific Notes:

  4. Installation root directory is `/usr/ft200` by default.
  5. The AL860 environment variable should be set to `/usr/ft200/sunos`.
  6. The PATH environment variable should reference `/usr/ft200/sunos/bin`.
  7. Host Slave Mode applications should link with the library slv860.a.
  8. i860 programs should link with library i860lib.a.
  9. Make files for sunos targets are named make.sunos.
  10. Slave Mode programs should be compiled with the -Dsunos flag.

- Make files for i860 targets are named make.i860.
- Host Slave Mode programs should use the include file allib.h.
- i860 application programs should include the file i860lib.h.

- Functions alsetba(), alsetwa(), algetba(), algetwa(), and algetla() are documented as returning integer values indicating SUCCESS or FAILURE. These functions are incorrectly implemented as void functions which do not return a value. Please treat these functions as void functions, or alternatively, refrain from error checking their return values.
- The functions read860() and write860() are documented as returning the number of bytes read or written or -1 to indicate an error condition. These functions actually return -1 (FAILURE) on an error and zero (SUCCESS) if there are no errors.
- Functions almapload() and alload() cause the application to exit() if they detect an invalid COFF file.
- The following functions were converted to ANSI variable argument function prototypes: atioctl, extioctl, rtcioctl and usertrap.
- The processor argument passed to mp_thread_start is now checked to validate that the processor is valid. Previously, all mp_thread_start requests defaulted to invoking a thread on the other processor.
- LynxOS only: The setpgrp() system call fails.

FT200-V Software Release Notes

Version 1.2C

Date: 11-Jan-94

Host and Target Build Environments:

SunOS Host Environment	SunOS 4.1.3
Solaris 2 Host Environment	Solaris 2.1
i860 C Compiler and Tools	PGI 2.2

Warnings:

- On releases of FT200-AT Run-Time 1.2C or earlier, a user interrupt handler can call certain functions which cause the i860 trap handler to fail. A hardware interrupt (from a daughtercard, the host, the real-time clock, or another processor) causes the i860 to save state and switch context to the i860 trap handler. If enabled and registered correctly, the trap handler will invoke a user specified interrupt handler. If the user handler calls a function which reenters the trap handler, then the trap handler will not correctly return to the application's main thread of control. Examples of functions that could reenter the trap handler follow: printf(), mp_printf(), flush(), inthost() or user extensibility functions. The user handler is restricted from calling functions which require host support as is the case for many C run-time library functions.
- The Slave Mode code has instances in which error codes are incorrectly passed back to the user program due to the use of the exit() system call.
- Functions read860 and write860 return 0 for success and -1 for failure. The number of bytes read is not returned.
- The aldev function may be called on a non-opened device without returning a FAILURE. The *dev* function is not error checked.

Notes for version 1.2C:

- Support for Solaris 2.1/Sparc has been added. This release supports the loadable VME device driver. Support for the Performance Technologies and Bit3 SBus to VME converter cards will be present in release 1.3.
- mp_thread_start and mp_thread_query have been re-implemented to fix a bug with multiple fast mp_thread_start/mp_thread_query requests.
- The umalloc, upmalloc and ufree calls have been re-implemented to fix bugs dealing with cache coherency.
- The version numbers printed during the quick test will be displayed as 1.3 instead of 1.2C.
- The documentation states to run mpex7 with the '-T' flag, this is incorrect: instead run the test as 'rt860 mpex7'.

mp_4meg_addr() supports only the following values for *prop.*

MP_KADDR_WBC	Write-back cached
MP_KADDR_WTC	Write-through cache
MP_KADDR_UNC	Uncached

The read only address spaces are no longer supported.

- The Software Reference manual has been updated to include information on available virtual address.

FT200-V Software Release Notes

Version 1.2b (formerly 1.2.2) - Special SP61

Date: 15 December 1993

Host and Target Build Environments

SunOS Host Environment	SunOS 4.1.3
i860 C Compiler and Tools	PGI 2.2

Notes for version 1.2b:

- Support has been added for the Bit3 model 467 card.
- Preloading problems introduced in 1.2.1 have been fixed.
- This release supports PGDBG 2.1A only.

Version 1.2a (formerly 1.2.1) - Special SP52

Date: 27 October 1993

Host and Target Build Environments

SunOS Host Environment	SunOS 4.1.3
i860 C Compiler and Tools	PGI 2.2

Notes for Version 1.2a:

- Modifications made to support PGDBG. These modifications do not impact the SunOS or Lynx device drivers. Therefore, only host and i860 applications need to be rebuilt - the host operating systems do not need to be reconfigured or rebuilt.
- The Lynx binaries included in this release are from version 1.2.
- The user validation diagnostics have been modified to handle embedded VME based systems.

FT200-V Software Release Notes

Version 1.2

Date: 28 September 1993

Host and Target Build Environments

SunOS Host Environment	SunOS 4.1.3
LynxOS	LynxOS + GCC
i860 C Compiler and Tools	PGI 2.2

Notes for Version 1.2:

- The User Validation Diagnostics have been added. Refer to the section in the FT200-V Hardware Installation and Reference Manual. The sources for the user validation diagnostics are not included as part of the release.
- Multiple board support has been added.
- Support has been added for an installed VME device driver for the SunOS and Solaris environments.

FT200-V Software Release Notes

Version 1.1

Host and Target Build Environments

SunOS Host Environment	SunOS 4.1.1
LynxOS	LynxOS + GCC
i860 C Compiler and Tools	PGI 2.2

Notes for Version 1.1:

- The distribution contains all of the files required to build any of the components of the release. All the sources are included, except for the following four object files found in /usr/ft200/src/mpkernel: trap.o, n10fpeh.o, n10utila.o, and n10utils.o. These files are required to rebuild /usr/ft200/lib/mpkernel. To build any of the components, change to the appropriate source directory and execute a make command, specifying as the target one of the following: **sunos-all** or **lynx-all**, substituting the appropriate host, the make target **i860-all** is used to create FT200 targets **-** such as the mpkernel, and i860lib.a. The make command will recurse through all of the appropriate sub-directories for each command, thus a **make sunos-all** at the root of the distribution will perform a complete make of all of the SunOS related targets. There are also corresponding target cleaning requests: **sunos-clean**, **lynx-clean** and **i860-clean**. These will remove all intermediary targets in the tree. *Do not do a* **make i860-clean** *in any directory that would make clean in /usr/ft200/src/mpkernel (these include /usr/ft200, /usr/ft200/src and /usr/ft200/src/mpkernel) as that will remove the aforementioned object-only distributed files. If you wish to clean out the /usr/ft200/src/mpkernel directory please make sure that you save the objects and restore them after doing the clean.*
- Release 1.1 of FT200-V RT860 does not contain the User Validation Diagnostics.
- The current FT200-V boards include hardwired VME Extended addresses in the PROM. Consult the release notes included with your FT200-V Hardware Installation and Reference manual for more information.

AL860-AT Release Notes

Version 1.6.3

Enhancements - Bug Fixes

- modified src/rt860/sysif.c for MSDOS Debugger
- The ^c interrupt handling was fixed in sysif.c to allow asynchronous interruption of execution while running with the debugger.
- modified slave mode kernel to correctly flush floating point pipe-line following reset
- Certain floating point pipe-line registers were not initialized at chip reset time in the slave mode kernel, which resulted in floating point traps occurring the first time an application was run (this problem only seems to have occurred following a power on condition). **SLKERNEL** was modified to perform the required initialization.
- correctly implemented floating point trap handling under Slave Mode.
- Floating point traps occurring on the i860 result in the current alcall() terminating with float point trap error condition. Previously, the i860 program would endlessly loop on the floating point trap, emitting a message of the form “opcode is xx”.
- Following an alcall(), the algetstatus() and alprtexitstatus() functions may be called to ascertain the reason that the alcall() terminated.

  Sample code:

  ```
  alcall (A_main, 0);

  if (algetstatus() != EXIT_NORMAL)

  alprtexitstatus ();
  ```
- released software has **ALINFO** directory

  The **ALINFO** directory is present in the software release tree and contains Alacron part number information.

Notes - Warnings

- Release 1.6.3 includes MSDOS, OS/2, Unix386 and Solaris2/x86 hosts.
- kernel sources include Trap Handler and IEEE Exception handler code
- Previously, source releases did not include the source for the i860 trap handler and IEEE floating point exception handler.

AL860-AT Software Release Notes

The following part numbers are affected:

40000-00191	AL860-AT	AL860-AT Runtime, Object, MSDOS
40001-00191	AL860-AT	AL860-AT Runtime, Source, MSDOS
40000-00192	AL860-AT	AL860-AT Runtime, Object, OS/2
40001-00192	AL860-AT	AL860-AT Runtime, Source, OS/2
40000-00193	AL860-AT	AL860-AT Runtime, Object, Unix
40001-00193	AL860-AT	AL860-AT Runtime, Source, Unix

AL860-AT Software Release Notes

Version 1.6.2

Enhancements - Bug Fixes

A bug was corrected in the RT860 kernel that would cause the system to fail if a hardware interrupt occurred during a system call (such as write from printf, etc.)

Notes - Warnings

None

AL860-AT Software Release Notes

Version 1.6.1

Enhancements - Bug Fixes

- In release 1.6, the file include/allib.h was missing FAR designators that caused compile problems with Windows applications that were not large model. This affected applications using the MSDOS/Windows DLL. Changes have been made to allib.h to correct the problem.
- The 860 library ufree() function now frees memory correctly. Function ufree() is used to free *uncached* memory that is allocated using the umalloc() function. In release 1.6, repeated calls to umalloc() and ufree() would ultimately fail after the depletion of the entire memory heap.
- The 860 library functions umalloc(), upmalloc() and ufree() have been modified to reduce likelihood of cache coherency problems arising with intermixing calls to malloc() and free().

Notes - Warnings

- When using Watcom C under MSDOS, you must place the Watcom BIN directory ahead of the Portland Group BIN directory in your PATH. The file DOS4GW.EXE should be referenced from the Watcom directory tree not the Portland Group directory tree.

AL860-AT Software Release Notes

Version 1.6

Source and Object Releases

Version 1.6 is provided in two forms; the OBJECT form contains only library and executable binaries, the SOURCE form contains all binaries as well as sources required to rebuild binaries with the following exceptions:

- no source for USERDIAG is provided
- no source for the i860 trap handler is provided
- no source for i860 floating point exception handler is provided

Sample source code is provided for both OBJECT and SOURCE releases.

Compiler Revision Information

O/S	Compiler
MSDOS	MSC Version 8 (Visual C++ Version 1.5)
MSDOS	WATCOM Version 10.0
SYSV386	native cc
Solaris 2/x86	Suntools Version
OS/2	IBM CSET++ Version 2.0

note: MSDOS make files are provided for Microsoft C version 7, however all supplied binaries are compiled using version 8.

File Listing

MSDOS binary files

MSDOS\LIB\DOS860WC.LIB	AL860-AT Driver library for WATCOM
MSDOS\LIB\COP860WC.LIB	RT860 library for WATCOM
MSDOS\LIB\SLV860WC.LIB	Slave mode library for WATCOM
MSDOS\LIB\RT860WC.LIB	RT860 (user extensibility) library for WATCOM
MSDOS\LIB\CDOSP860.LIB	AL860-AT Driver library for MSC
MSDOS\LIB\W3SLV860.LIB	Slave mode library for Windows DLL (MSC)
MSDOS\LIB\COP860.LIB	RT860 library for MSC
MSDOS\LIB\SLV860.LIB	Slave mode library for MSC
MSDOS\LIB\RT860.LIB	RT860 (user extensibility) library for MSC
MSDOS\BIN\PRTMAP.EXE	PRTMAP program executable
MSDOS\BIN\RT860.EXE	RT860 program executable
MSDOS\BIN\W3SLV860.DLL	Windows DLL
MSDOS\BIN\USERDIAG.EXE	user diagnostic executable

SYSV386 binary files

sysv386/driver/*	Device Driver Files
sysv386/bin/cpp	ansi C preprocessor executable
sysv386/bin/prtmap	prtmap program executable
sysv386/bin/rt860	rt860 program executable
sysv386/bin/userdiag	user diagnostic program executable
sysv386/lib/cop860.a	rt860 library
sysv386/lib/slv860.a	slave mode library
sysv386/lib/rt860.a	rt860 (user extensibility) library

Solaris 2/x86 binary files

sun5x/driver*	Device driver files
sun5x/bin/prtmap	prtmap program executable
sun5x/bin/rt860	rt860 program executable
sun5x/bin/userdiag	user diagnostic program executable
sun5x/lib/cop860.a	rt860 library
sun5x/lib/slv860.a	slave mode library
sun5x/lib/rt860.a	rt860 library

OS/2 binary files

OS2\BIN\PRTMAP.EXE	PRTMAP program executable
OS2\BIN\RT860.EXE	RT860 program executable
OS2\BIN\USERDIAG.EXE	user diagnostic program executable
OS2\LIB\COP860.DLL	rt860 DLL
OS2\LIB\COP860.LIB	rt860 library
OS2\LIB\SLV860.DLL	slave mode DLL
OS2\LIB\SLV860.LIB	slave mode library
OS2\LIB\RT860.LIB	rt860 (user extensibility library)
OS2\SYS\*	OS/2 device driver files

MSDOS\OS\2 Example Files

SRC\EX*	Slave Mode examples
SRC\W3LIB	Windows DLL example
SRC\USEREXT	User extensibility example

Unix Example Files

src/ex*	Slave Mode examples
src/userext	User extensibiltity example

Makefile Convention

Make files are provided for rebuilding various host operating system and i860 targets. Recursive make files are provided where invoking make in a directory will cause make to be invoked in all sub-directories. The following conversions apply:

- Make utility - For all MSDOS compilers, NMAKE.EXE (from Microsoft C) is used. For Watcom makes, NMAKE is used (it may be required that NMAKE from Version 6 or 7 Microsoft C be used)
- For Unix Hosts (SYSV386 and Solaris), the native make program is used.
- For OS/2, NMAKE provided with the IBM CSET++ compiler is used.

- For building i860 targets under MSDOS and OS/2, NMAKE is used. For building i860 targets under Unix, the native make program is used.

Make Files

The following summarizes the make files for various host and i860 targets:

Makefile	Description
make.c7	make host targets using Microsoft C version 7
make.c8	make host targets using Microsoft C version 8
make.c7w	make host Windows 3.x targets using Microsoft C version 7
make.c8w	make host Windows 3.x targets using Microsoft C version 8
make.wc	make host targets using Watcom Version 10.0
nmake.860	make i860 targets under MSDOS
make.sysv386	make host targets under SYSV386
make.sun5x	make host targets under Solaris 2/x86
make.i860	make i860 targets under SYSV386, or Solaris 2/x86

Make Targets

The following targets (arguments to make) are available when invoking make (or NMAKE):

Target	Description
all	build targets (not including examples)
test	build example code targets
run-test	execute example code programs
clean	clean targets (not including examples)
clean-test	clean example code targets

AL860-AT Software Release Notes

Changes with release 1.6:

- source code changes for portability
- Windows 3 DLL initialization modified

  The initialization sequence for the Windows 3 DLL has been significantly modified. The ***W3Slv860Setup()*** function now accepts a single argument consisting of a window handle (HWND *hWndMain*). The semantics of this window handle are as follows:

  if *hWndMain* is non-null, then i860 writes to standard out and error are directed to *hWndMain* as before. If *hWndMain* is null, i860 writes to standard out and error are directed to the Windows 3 Debug Screen, using the SDK function ***OutputDebugString()***. If ***W3Slv860Setup()*** is not called, the default case is output directed to the debug screen.

  Formerly, an argument was presented to ***W3Slv860Setup()*** to pass environment information to the DLL. This is replaced with a standard **.INI** file. The file **AL860AT.INI** may be placed in the windows home directory which contains the definitions of environment variables required by the DLL. The following example shows a sample AL860AT.INI.

  ```
  [environment]
  AL860=C:\USR\AL860\MSDOS
  AL860CFG=(0,110,15)(1,118,12)
  ```

  The syntax of defining AL860 and AL860CFG are identical to that of the standard environments set using the MSDOS *set* command.

- Solaris2/x86 driver modified

  The Solaris Device Driver now allows multiple host processes to perform alopen() calls on the same board.

- Sockets support implemented. (SYSV386, Solaris only)

  The following socket functions are available to i860 applications running under Unix hosts:

  gethostbyname ()
  inet_addr ()
  sendto ()
  send ()
  accept ()
  bind ()
  connect ()
  getpeername ()
  getsockname ()
  listen ()

recv ()
recvfrom ()
select ()
setsockopt ()
getsockopt ()
shutdown ()
socket ()
gethostname ()

To utilize these functions, the i860 application must link to the file ***lib/i860lib.a***.

Host Slave Mode applications must include host libraries supporting sockets. In the case of SYSV386, the following should be appended to the host application link command:

```
            -linet
```

For Solaris2/x86 no additional link directives are required.

- OS/2 support

  Version 1.6 has full support for OS/2. Refer to the section "OS/2 Software Installation" for information on RT860 software installation under OS/2.

- added RTC_DETACHINT to rtcioctl()

  Previous releases implemented RTC_DETATCHINT (which is not spelled correctly). The constant RTC_DETACHINT has been added. Previous references in the manual to RTC_DETACH were incorrect and have been fixed.

- added alinvalidate_caches ()

  The host function alinvalidate_caches() has been added to force the reload of host PTE caches. This function may be used in cases where the I860 application is performing MMU mapping to ensure that VtoP() calls on the host side operate correctly.

- slave mode trap enable bits now set correctly

  The default trap enable bit are set to enable flush zero, enable floating point traps, and to enable mis-aligned data access trap handling. ***This breaks applications that relied on floating point and data access traps being disabled.*** The user application should explicitly set the trap enables if the default is unacceptable.

- Fix MSC version 7 time() problem

  Changes made to compensate for a bug introduced in Microsoft C version 7 where the return from time() library function did not adhere to the industry standard.

- Fix bug in rtkernel standard system call

  Parameter 6 was passed incorrectly to host during system calls. This problem showed up during addition of socket library support; it would only be a problem for user extensibility applications passing 6 or more arguments.

- Fix bug in rtkernel time reference.

  A bug was fixed in rtkernel where the kernel initiated system call to time() could result in program data corruption.

- Added System V IPC to i860lib.a

  The System V IPC (messaging and semaphores) library entry points were added to i860lib.a. Formerly the Portland Group compiler libraries were relied upon for providing these function entry points.

- OS/2 User Extensibility in Slave Mode

  OS/2 slave mode extensibility does not work due to the nature of the OS/2 DLL model. We expect to be able to have the slave mode library call out to a user provided function usersys(). The DLL can only invoke it's linked in version.

  Work arounds would include patching usersys.c in src/slv860, or building a non-DLL slv860.lib. Neither of these is provided with this release.

AL860-AT Software Release Notes

Version 1.5

Build Tools Release Level

MSDOS Host C compiler	**Microsoft C Version 7, Watcom C Version 9.0**
MSDOS Host Assembler	Microsoft MASM Version 5.1
i860 C Compiler	Portland Group C Version 2.2
Unix/386 Reference	Interactive Systems 3.2 Release 2
Solaris Reference	Sunsoft Solaris 2.1

Changes with release 1.5:

- **load860** program no longer supplied.
- Implemented new directory structure

  In order to support an increasing number of host operating system environments, the basic directory hierarchy for ALACRON software has been changed. The following table summarizes the new directory structure.

Directory	**Contents**
src	all source code sub-directories
include	all include files
lib	all i860 libraries and Kernel files
msdos/bin	DOS executable programs and Windows DLL
msdos/lib	DOS libraries (Microsoft C, Watcom)
sysv386/bin	Unix SYSV386 executable programs
sysv386/lib	Unix SYSV386 libraries
sysv386/driver	Unix SYSV386 Device Driver
sun5x/bin	Solaris executable programs
sun5x/lib	Solaris libraries
sun5x/driver	Solaris Device Driver
tools	Common tools and include files

  The **AL860** environment variable must be set to the sub-directory for the appropriate host operating system. For example, if MSDOS is the host O/S, **AL860** would be set to **C:\USR\AL860\MSDOS**. If Solaris were to be used, then **AL860** would be set to **/usr/al860/sun5x**.

- Host C compiler Defines

  When using the host include files (allib.h, etc.) it is necessary that the preprocessor symbols be defined to specify the host O/S and C compiler. The following table summarizes the required defines.

MSDOS using Microsoft C Version 7	-DMSC -DMSDOS
MSDOS using Watcom C	-DWATCOM -DMSDOS
MSDOS using Microsoft C, Windows	-DMSC -DMSDOS -DW3
Unix SYSV386	-DUNIX -DSYSV386
Solaris 2	-DUNIX -DSUN5X

- allib.h - function prototypes have FAR CDECL declarations

  Function prototypes in allib.h now have FAR and CDECL declarations to facilitate use with Windows applications development. To avoid syntax errors when compiling the C preprocessor symbols described above (item 3) must be used.

- Language support tools (as860, ld860, ar860, nm860, ...) are not provided.

  The language support tools are no longer provided with the ALACRON Runtime software. Use those distributed with the Portland Group C and FORTRAN compilers version 2.2 and later.

- Watcom Support

  Slave mode and user extensibility libraries are now provided for use with the Watcom C compiler. This facilitates development of 32 bit host code under MSDOS. The following libraries are provided:

slv860wc.lib	Slave Mode Library
dos860wc.lib	Driver Interface Library
cop860wc.lib	User extensibility library
rt860wc.lib	User extensibility library

  In AL860 software version 1.5, the Watcom libraries do not make use of the hardware interrupts from the AL860-AT board (a simple polling mechanism is used)

- Makefile naming

  The following table summarizes the names of make files provided in this release.

make.sysv386	Unix SYSV386 makefiles, host software
make.sun5x	Solaris 2 makefiles, host software
make.c7	Microsoft C Version 7, host software
make.c7w	Microsoft C Version 7, Windows, host software
make.wc	Watcom C, host software

make.i860	Unix/Solaris makefiles, i860 software
nmake.860	NMAKE makefiles, i860 software

Where in the past, a single make file was provided for both host and i860 code (in for example the slave mode examples), separate make files are provided.

Make files for Watcom are compatible with Microsoft NMAKE.

- Multiple calls to RTC_ATTACHINT may be made

  The RTC_ATTACHINT function of **rtcioctl()** may be called to attach multiple interrupt handler functions. The command RTC_DETATCHINT is added to detach a particular handler.

- New umalloc, upmalloc, ufree

  The functions umalloc, upmalloc, and ufree have been implemented using the standard C library function malloc, and address aliasing to uncached addresses.

- Added i860lib.h

  Function prototypes and parameter definitions for all i860 functions in i860lib.a (including rtc, at, ...) are declared in the new include file **i860lib.h.**

- Windows DLL writes to stdout, stderr are modified

  Output is buffered until a string containing a newline is received.

- Added error message functions control functions

  The following functions were added to control error and warning messages emanating from the Slave Mode library, and i860 Kernel.

alseterrorlevel	set error message level
algeterrorlevel	get current error message level
errmsg	generate error message

The argument to **alseterrorlevel** is an integer taken from one of the following (defined by including allib.h)

ERRMSG_NONE	no messages generated
ERRMSG_ERROR	error messages only
ERRMSG_WARN	warnings and error messages
ERRMSG_TRACE	all messages

By default, the error message level is set to ERRMSG_ERROR.

- Implement slave mode completion status functions

  Functions have been added to allow host code to determine the reason for an **alcall()** function completing. The function **algetstutus()** returns an integer that corresponds to one of the following reasons:

0	Normal Termination
1	Instruction Trap
2	Interrupt Trap
3	Instruction Access Trap
4	Data Access Trap
5	Floating Point Trap
6	Kernel Problem
7	Integer Overflow
8	Illegal Instruction
9	Missing Unlock
10	Unknown Trap
11	Page Fault
12	Uncaught Signal
13	Alarm Clock
14	Bad Signal

  The function **alprtexitstatus()** will print on standard error a message indicating the reason for completion.

- Windows 3.1 Slave Mode DLL added
- Slave SYS_ACCESS function operation corrected
- Added setresult() to Slave Mode for User Extensibility
- aldev() returns correct value
- Added ELF file loading support
- Numerous cosmetic source code changes

  All C code contains ANSI compliant declarations. The an alternative C pre-preprocessor (**unproto**) is provided for Unix SYSV386 where the standard C compiler is not ANSI.

- w3setup - malloc called with correct string length

User Diagnostic Software Release Notes

Version 2.5

Enhancements - Bug Fixes

The following changes were made to allow the user diagnostics to run on certain non-standard memory configuration sizes such as 2 MB and 4 MB boards:

- A time-out value in the 860 memory test was set to the memory size divided by 4 MB. This produced a time-out of zero for 2 MB boards. This has been fixed.
- An error message in the RTC test was longer than 80 characters. This was shortened for cosmetic reasons.
- The makefile for 860 programs was changed so that program data was at a valid address for boards with small memories.
- Code using the .abs directive was recoded to use .align 32.
- When the configuration is determined in the initial table, the diagnostics run a program to determine the clock speed of the 860 processor, i.e. 40 or 50 MHz. The RTC test uses this information. If an unknown speed is detected, then an error is produced.

Notes - Warnings

None

User Diagnostic Software Release Notes

Version 2.4

Enhancements - Bug Fixes

- The 40 MHz tests were removed from the RTC test
- Fixed a problem in which certain error messages were not reported as FAILED.
- Changes were made to allow the diagnostic to run on certain non-standard memory sizes such as 2 MB.

Notes - Warnings

None

User Diagnostic Software Release Notes

Version 2.3

Enhancements - Bug Fixes

- Displays summary total of the tests that were run and the number of failures. If a failure occurred, a test matrix and summary are printed. The summary and test matrix will appear after the completion of each test suite by any board.

Notes - Warnings

None

2

Utilities

Alacron RT860 runtime software is shipped with a number of utilities. This section provides a description of each of these utilities, their command line arguments and functionality. Any functionalities that are not common to all processor boards are noted explicitly.

Utility **rt860** is the stand-alone mode control program, used to load and execute programs on the processor board. The **prtmap** program is a utility that displays the current processor board's memory management configuration.

RT860 (uppercase) is the generic term for the Alacron processor board software environment. It includes host and processor board include files, libraries, utilities and source code. This manual is a reference for RT860. rt860 (lowercase) is the stand-alone mode utility covered in this section.

Usage:

```
rt860 [options] [file] [program arguments]
```

Description:

The **rt860** utility executes the program specified by the *file* argument on an Alacron processor board. The function **main()** in *file* is invoked with any arguments specified as *program arguments.* Utility **rt860** is an alternative to using Slave Mode which requires a program running on the host computer to make calls to a host library which controls the Alacron processor board. The **rt860** utility allows a user on the host computer to start a program on the Alacron processor board without having to write software that runs on the host.

If *file* is not found in the current directory, the directories specified by the environment variable **PATH** are searched.

When a program is loaded, the stack frame is initialized with the *program arguments* that follow *file* and the environment string (argc, argv, argp). For MSDOS Hosts, general command line expansion is supported where the form *@file* takes arguments from *file*, one per line. The **rt860** utility will read in command line arguments from *file*. Each line in the file can contain one parameter (and it's optional argument).

On Intel 860 processor boards, *file* must be an Application Binary Interface (ABI) compliant COFF executable file. The ABI supports a comprehensive set of operating system services (file I/O, etc.). The i860 program is run with its Memory Management Unit (MMU) enabled. This allows ABI program virtual address resolution, and memory protection.

When invoked, **rt860** loads a small Kernel onto the processor board and initializes the i860 runtime environment. The i860 Kernel supports a core set of system calls enhancing the portability of applications running on the processor board. The Kernel also provides MMU support and i860 trap handling including data and instruction traps, memory access violations and IEEE floating point exception handling.

Utility **rt860** supports a fixed set of system calls made by programs executing on the processor board. If you wish to provide additional system call support for your application, please refer to the section on user extensibility.

On the FT200-AT, FT200-PCI and the AL860-PCI, invoking **rt860** without options will cause *file* to be loaded. If *file* is absent, **rt860** attempts to load file **a.out**. If **a.out** is not found in the current directory, an appropriate error message will be displayed.

Utilities

Options:

Runtime characteristics of programs invoked on the Alacron processor board may be controlled using one or more *options* arguments listed below:

-a	Disable data alignment trap handling. If data alignment trap handling is enabled, programs may access mis-aligned data without disrupting program execution. The overhead of handling data access traps could degrade system performance. If data alignment trap handling is disabled, programs terminate upon receipt of data access traps. The compiler aligns data according to type. By default, data alignment trap handling is enabled. It is recommended that data access traps be resolved by modifying source code.
-B	Instructs **rt860** to configure the data, stacks, bss and heap program regions as Write-Back. This is the default state.
-c *freq*	The processor boards have a real time clock which is enabled by this option. The real time clock delivers an interrupt to processor zero at a rate of *freq* Hz. Clock interrupts are counted by the processor board Kernel and subsequently used to support time system call requests. By default, the real time clock interrupt is disabled and time system call requests are handled by performing a remote procedure call to the host computer.
-D	Display version numbers of the Alacron runtime software. *Not supported on the AL860-AT.*
-d *dev*	Specifies the processor board unit number. The default is device 0.
-f *code*	Sets floating point trap code enable bits. The *code* is a hex value between 0 and 7. Each of the three bits specified enables or disables a type of processor trap. When the bit is set, the trap is enabled. Bit 0 (LSB) enables floating point traps. Bit 1 enables traps on inexact results, and Bit 2 (MSB) enables flush to zero. When flush to zero is disabled, underflow traps are executed. The default value for *code* is 1 which means that floating point traps are enabled and traps on inexact results and flush to zero are disabled (underflow traps enabled).
-h	Display help messages. *Not supported on the AL860-AT.*
-i	Initializes the processor board. The command `rt860 -i` must be executed before program *file* is run. Certain processor boards do not require this initialization. Specifically, the AL860-AT and FT200-V do require processor board Kernel initialization, but all other boards do not. During the initialization, memory is sized, the processor board Kernel is loaded, and MMU paging tables are initialized. Previously loaded programs are discarded. *It is not necessary to use this option on the FT200-AT, FT200-PCI or AL860-PCI boards.*
-s *size*	Allocates *size* bytes of memory on the processor board for runtime stack growth. Argument *size* is specified in base 10, and is rounded up to the nearest multiple of page size (4096 bytes). Memory that is not used for code, data, bss or stack is placed on the heap. Heap memory is available

for dynamic memory allocation using system calls sbrk() and malloc(). The default stack allocation is 65536 bytes. No provision is made for dynamic handling of data access faults on the stack or heap.

-T	Instructs **rt860** to configure the data, stacks, bss and heap program regions as Write-Through/Snooped caching. The default is write-back cached. *This option is not supported on AL860 series processor boards.*
-U	Instructs **rt860** to configure the data, stacks, bss and heap program regions as Uncached. The default is write-back cached. *This function is not supported on the AL860-AT processor board.*
-v	Enables verbose output during program execution. All system calls initiated by the application running on the processor board produce diagnostic output.
-V	Enable higher level of verbose output during **rt860** execution. *Not supported on the AL860-AT.*
-Z	Instructs **rt860** not to map virtual address zero. This is useful for helping track programs that dereference NULL pointers. *Not supported on the AL860-AT.*

Example:

```
C:> rt860 -i                        # necessary on AL860-AT
                                    # and FT200-V only

C:> rt860 -D -T execfile 1 2 3 4       # Execute program
```

See also:

prtmap
User Extensibility

prtmap

Usage:

```
prmap [options]
```

Description:

The **prtmap** program may be used to display the Memory Management Unit (MMU) mappings that are in effect for a given processor board. Utility **prtmap** is used after a program that has been run on the processor board completes**.**

On FT200 series boards, there are four page tables. Separate page tables are created for the Kernel and program spaces of each of the two processors. The **-A**, **-B**, **-C**, and **-D** flags are used to designate which of the four page tables to dump. Also, on the FT200 only, table entry lines displayed with "pages" are 4 KB pages, lines displayed with "PAGES" are 4 MB pages.

Options:

Options Supported on all Processor Boards	
-?	Display help message
-d *dev*	Use the processor board unit number specified by *dev*
-u *dev*	Use the processor board unit number specified by *dev*
-a	Display all information available. This option displays each directory and second level page table entry
-b *addr*	Specify the address of the first level page table.

Options Supported on FT200 Processor Boards	
-A	Display MMU configuration for processor 0, Kernel mapping.
-B	Display MMU configuration for processor 1, Kernel mapping.
-C	Display MMU configuration for processor 0, task mapping.
-D	Display MMU configuration for processor 1, task mapping.

Output:

The output of the **prtmap** utility is a table of mappings. Pages with common mode settings are grouped together. Each line represents a range of addresses. Both virtual (V) and physical (P) addresses are listed. MMU mode settings are shown in the right column. The mode flags are as follows:

Flags Supported on All Processor Boards	
D	Page is dirty. In other words, it has been written.
U	Page is user accessible.
W	Page is writeable.
A	Page has been accessed. In other words it has been read or written.

Flags Supported by FT200 Series Boards Only	
M	Page is a 4 Megabyte page. The default page size is 4 Kilobytes.
t	Page uses the write-through (snooped) cache mode.
P	Page is present in processor board memory.
c	Page is cache disabled.

Utilities

Usage:

```
mpdump [-d <device>] [-a] [-g] [-p <processor-id>] [-P <processor-id>]
```

Description:

The **mpdump** utility will provide system information obtained from the specified board's kernel global area. The mpdump utility is used during development to assist in determining application run states and termination conditions.

One or more of the following types of information may be displayed for any specified board in the system:

- Global kernel information (state; on-board real time clock information; DRAM information including physical; kernel and task information)
- Per processor kernel information (state; counter information including interrupts, system calls, and clock ticks; page directory addresses; application supplied interrupt handlers; and kernel message or panic information)
- Per processor task information (processor context; function entry address; function parameters; and function return value)

The mpdump program provides RT860 runtime software kernel state information. The following table describes the optional parameter list:

Parameter	Description
-d <device>	Indicates which Alacron board to retrieve information from, the default is device 0.
-a	Shorthand means of showing all information (-g -P0 -P1).
-g	Display global information only
-p <processor-id>	Display terse processor state - does not include context dump.
-P <processor-id>	Display verbose processor state: includes context information.

The following table describes the values displayed with the -g flag:

Field	Description
g_initted	Set to zero when the kernel is not initialized, one when board resources are initialized.
g_nprocs	Number of processors which successfully initialized.
g_ticks	Number of on-board real time clock interrupts fielded.
g_freq	Current real time clock frequency requested (in interrupts per second). If zero the on-board clock is disabled.
g_rfreq	The (floating point) approximation of the actual number of interrupts per second. This number may be slightly different due to the actual rates available to the on-board device.
g_phys	The physical base address of the detected on-board DRAM.
g_size	The size of on-board DRAM.
g_skpbrk	End of kernel data+bss+heap prior to task execution, page aligned physical address.
g_kpbrk	End of kernel data+bss+heap at present, page aligned physical address.
g_tpbrk	End of tasks data+bss+heap, page aligned physical address.
g_tvbrk	End of tasks available data+bss+heap, virtual address.
g_tvhigh	Largest end of tasks data+bss+heap, virtual address.
g_tvend	Current end of tasks data+bss+heap, virtual address. [A task may move the g_tvend field up and down, but when new heap area is allocated - via a brk or sbrk call, usually invoked by malloc - it is only cleared out when moved above g_tvhigh.]
g_tvedata	End of tasks data+bss, virtual address.

Not supported on the AL860-AT processor board.

Fields displayed with the -p flag:

Field	Description
p_procid	Processor identifier
p_initted	Set to zero when processor is not present, or not initialized. Set to one when processor is present and initialized.
p_state	Processor state, see the Processor State Table below
p_nints	Number of interrupts this processor has fielded.
p_syscalls	Number of system calls fielded.
p_ticks	Total number of clock ticks this processor has handled.
p_uticks	Number of clock ticks handled when not in the system call code.
p_sticks	Number of clock ticks handled when in the system call code.
p_epsr	EPSR value, including the following information: big/little-endian status, XR/XP processor identification, chip stepping information.
p_context	Last processor context address. This value is only valid when the processor has exited or in a system call. The context data will be displayed only if the -P flag is specified.
p_ipgdir	Kernel page directory address.
p_apgdir	Application page directory address.
p_exitreturn	return value for exit conditions.
p_cause	Source of exit exception.
p_panic	Kernel message string (including panic messages).
p_atfctn	Application interrupt handle for host initiated interrupts.
p_atenable	Enable flag for invocation of p_atfctn when a host interrupt is fielded on this processor.
p_atcount	Number of interrupts from host.
p_ext0fctn	Address of application supplied interrupt handler for EXT0.
p_ext1fctn	Address of application supplied interrupt handler for EXT1.
p_mp_catch	Address of application supplied interrupt handler for interprocessor interrupts.
p_rtcenable	Enable flag for invocation of chained real-time clock interrupts.
p_rtcchainfctn	List of application functions to be invoked when a real-time clock interrupt is fielded. Only invoked if the p_rtcenable flag is non-zero.
p_entry	Last function entry address
p_src	Indicates whether the host or the another processor invoked the function.
p_stacktop	Stack address when function entered.
p_params	list of parameters passed into function.
p_rval_l	32 bit integer return value from function. [Only one of p_rval_l, p_rval_f, and p_rvalp_d is valid for any function.]
p_rval_f	32 bit floating point return value from function.
p_rvalp_d	64 bit double precision floating point return value for function.

Utilities

Processor State Table:

Value	Name	Description
0	MPS_RESET	Board has been reset, kernel has not initialized
1	MPS_INITTED	Board resources have been initialized
2	MPS_IDLE	Kernel is waiting for function invocation request
3	MPS_RUN	Application is being executed
4	MPS_DBG	Application has hit a debug break point
5	MPS_SYSCALL	Kernel is servicing an application system call request
6	MPS_PANIC	Kernel has panicked
7	MPS_TERMINATED	An exit condition was reached
8	MPS_STOPPED	Application is in a stopped state

Context Fields:

Field	Description
uintreg[0-31]	Integer register value r0-r31
ufltreg[0-31]	Floating point register values f0-f31
udblreg[0-15]	Double precision floating point register values f0-f30
ufir	Faulting instruction register value at time of trap
upsr	Processor status register value at time of trap
uepsr	Extended processor status register value at time of trap
ufsr	Floating point status register value at time of trap
udirbase	Dirbase register value at time of trap
udb	Value of DB register at time of trap

3

Host Slave Mode Library Reference

The Slave Mode Library provides support to a host application that wishes to interface with an Alacron processor board. This chapter provides reference information for each of the library functions. The functions are grouped according to functionality and then listed alphabetically. Please refer to the index if you are having difficulty locating a particular function.

Include Files

The standard include file for the Host Slave Mode library is allib.h. This file may be found in the include directory underneath the Alacron RT860 software directory tree.

A standard Host Slave Mode program no longer needs to include the file al860.h.

Libraries

A Host Slave Mode application uses different libraries depending on the host environment:

Environment	Libraries
MSDOS (AL860-AT, FT200-AT)	CDOSP860.LIB and SLV860.LIB
MSDOS (AL860-PCI, FT200-PCI)	SLV860.LIB
MSWindows 3.1	W3SLV860.LIB and W3SLV860.DLL
OS/2	SLV860.LIB
Unix	slv860.a

Host Slave Mode Library Reference

Constants

The following table contains a listing of the definitions provided in the include file allib.h. These constants are used throughout this chapter.

Constant	**Description**
SUCCESS	Indicates a successful completion of a call
FAILURE	Indicates a failure of a call, the global variable err860 should contain more information
MAX_I860DEVS	Maximum number of devices supported.

Type Definitions

The following table contains common typedef's provided in allib.h:

Name	**Purpose**
ADDR	Used to convey on-board addresses to various functions.

Quick Reference

The following table provides a list of all of the functions in the Slave Mode library:

Function	Summary
aladdr	provides the address of a processor board program global symbol
alcall	starts a function on the main processor
alclose	closes the processor board device
alclrint	clear pending interrupts from the main processor
aldev	select a processor board device
algetb	read a byte from processor board memory
algetba	read a byte array from processor board memory
algeterrorlevel	return the current error level
algetdresult	get the double return value from a function
algetfresult	get the float return value from a function
algetiresult	get the integer return value from a function
algetw	read a 16-bit word from processor board memory
algetwa	read a 16-bit word array from processor board memory
algetl	read a 32-bit longword from processor board memory
algetla	read a 32-bit longword array from processor board memory
algetstatus	get the termination status of a processor board function
alinterrupt	interrupt the main processor
alintstat	handle system call requests from main processor and return
alinvalidate_caches	force reload of symbol table from processor board memory
alload	load processor board program with MMU disabled
AllocPage	allocate physical memory

Host Slave Mode Library Reference

Function	Summary
almapload	loads a processor board executable file and the Kernel
alopen	open processor board device
alprtexitstatus	print the status returned by algetstatus
alputs	Windows 3.1: write a string to a window
alreset	reset processor board
alsetb	write a byte to processor board memory
alsetba	write a byte array to processor board memory
alseterrorlevel	set the current error level
alsetw	write a 16-bit word to processor board memory
alsetwa	write a 16-bit word array to processor board memory
alsetl	write a 32-bit longword to processor board memory
alsetla	write a 32-bit longword array to processor board memory
alsize	return size of processor board memory
alwait	wait for main processor to complete and handle system calls
alwaitany	wait for main processor on any board to complete and handle system calls
ClrMMU	initializes and clears the directory page table
errmsg	write an error message if error exceeds current error level
MapVtoP	creates a virtual to physical mapping
mpcall	starts a function on the specified processor
mpclrint	clear pending interrupts from the specified processor
mpgetdresult	get the double return value from a specified processor
mpgetfresult	get the float return value from a specified processor
mpgetiresult	get the integer return value from a specified processor
mpgetstatus	get the termination status of a specified processor
mpinterrupt	interrupt the specified processor
mpintstat	handle system call requests from specified processor and return
mpprtexitstatus	print the status returned by mpgetstatus
mp_vtop	perform virtual to physical translation on specified processor
mpwait	wait for any processor to complete and handle system calls
mpwaitany	wait for any processor on any board to complete and handle system calls
VtoP	perform virtual to physical translation on main processor
W3Slv860Setup	Windows 3.1: specify a window as standard output and standard error

C Usage:

```
#include <allib.h>
ADDR   aladdr (char *name)
```

Arguments

name	Name of a program global symbol

Description:

The **aladdr** function returns the virtual address of the variable given by the string input argument *name*. The symbol table information in the currently loaded i860 executable file (COFF file format) is examined to resolve the address.

Function **aladdr** may only be called after a call to **almapload** or **alload**. Only the symbols loaded by the last call to **almapload/alload** may be referenced by **aladdr.**

The returned value is a virtual address and should be converted to a physical address (see **VtoP**) before use in functions such as **alget[bwl][a]**, and **alset[bwl][a]**.

Note, that there is significant overhead in executing the **aladdr** call. If the address of a symbol is to be used more than once, it is recommended that **aladdr** be called once and the value saved in a program variable.

Return Values:

Function **aladdr** returns a 32 bit unsigned address, or zero if the variable address could not be found.

Support:

AL860 and FT200 series processor boards.

Example:

```
ADDR virtadd;

virtadd = aladdr(“_shared_buffer”);
if (virtadd == 0) {
      printf (“FATAL: aladdr failed\n”);
      }
```

See also:

VtoP

C Usage:

```
#include <allib.h>
void   alcall (ADDR func, int numargs, ...)
void   mpcall (int proc, ADDR func, int numargs, ...);
```

Arguments

proc	The processor number
func	The virtual address of the processor board function
numargs	The number of arguments to function *func*. Valid values for *numargs* range from 0 to 12.
...	Optional variable number of additional arguments to function *func*. Arguments are all 32-bits in size and have a type of long. These are supplied if *numargs* is greater than 1.

Description:

The **alcall** and **mpcall** functions start the execution of function *func* on the processor board. Function **alcall** invokes *func* on processor zero while **mpcall** allows users of multiple processor boards to select the processor *proc* which will begin execution of *func.*

The value of *func* is a virtual address in the processor board program. The virtual address of any global symbol, including function names, is returned from a successful call to function **aladdr**.

Functions **alcall** and **mpcall** return immediately after function initiation. Function completion is determined by using functions **alwait**, **mpwait** and other similar functions.

All arguments are 32 bit integers and are abstract values in that they may be a combination of integer or address values. Currently a maximum of 12 arguments may be passed to **alcall** and **mpcall**. Single and double precision floating point arguments may be passed to the processor board, however, since they are passed as integer values, the processor board program must reconstruct the single and double precision floating point values as required.

The called function may return an integer, single or double precision floating point value, which may be retrieved by calling **alget[ifd]result** or **mpget[ifd]result.**

If a function called by **alcall** or **mpcall** terminates abnormally, the host may query the exit status of the processor board function by calling function **algetstatus** or **mpgetstatus**. Functions **alcall** or **mpcall** may start another processor board function even when a previously started function has failed. Clearly, this is only true if the processor board Kernel is still intact.

Do not attempt to call a function on a processor that does not exist - the host application will exit with an error message.

Return Values:

None

Support:

alcall	AL860 and FT200 series processor boards
mpcall	FT200 series boards only

Example:

```
/* Slave Mode Program */
 ADDR V_func;

V_func = aladdr("_func");
alcall(V_func, 3, 10L, 20L, 30L);
alwait();

/* 860 program */
void func(long a, long b, long c)
{
return();
}
```

See also:

aladdr
alwait
alwaitany
alintstat
alget[ifd]result
mpwait
mpwaitany
mpintstat
mpget[ifd]result

alclose

C Usage:

```
#include <allib.h>
int    alclose (int dev)
```

Arguments

dev	Processor board unit number

Description:

The **alclose** function closes processor board device specified by the *dev* argument. This function should be called before the user application exits.

Return Values:

None.

Support:

AL860 and FT200 series processor boards.

Example:

```
int dev=0;

alclose(dev);
```

See also:

alopen

C Usage:

```
#include <allib.h>
int    alclrint (int dev)
int    mpclrint (int dev, int proc);
```

Arguments

proc	The processor number
dev	The processor board unit number

Description:

The **alclrint** and **mpclrint** functions clear pending interrupts from a program running on the processor board. ***These functions are not normally used by application level code***.

Return Values:

SUCCESS	Function succeeded
FAILURE	Function failed

Support:

alclrint	AL860 and FT200 series processor boards
mpclrint	FT200 series processor boards only

Example:

```
mpclrint(0, 0);
```

See also:

alwait
alwaitany
alintstat
mpwait
mpwaitany
mpintstat

C Usage:

```
#include <allib.h>
int    aldev (int dev)
```

Arguments

dev	Processor board unit number

Description:

The **aldev** function selects the processor board device specified by the *dev* argument. This function should be called after a device is opened, using **alopen,** in order to select an active device for subsequent calls.

Return Values:

SUCCESS	Function succeeded
FAILURE	Function failed, device not opened

Support:

AL860 and FT200 series processor boards.

Example:

```
int dev;

for (dev=0; dev<8; dev++) {
       alopen(dev);
       aldev(dev);
       almapload(“prog”, 0x10000);
       }
```

See also:

alopen

Host Slave Mode Library Reference

algetb	**algetb**
algetw	**algetw**
algetl	**algetl**
algetba	**algetba**
algetwa	**algetwa**
algetla	**algetla**
alsetb	**alsetb**
alsetw	**alsetw**
alsetl	**alsetl**
alsetba	**alsetba**
alsetwa	**alsetwa**
alsetla	**alsetla**

C Usage:

```
#include <allib.h>
int    algetb (ADDR physadd)
int    algetw(ADDR physadd)
long   algetl (ADDR physadd)
int    algetba (ADDR physadd, char *buf, int n)
int    algetwa(ADDR physadd, short *buf, int n)
long   algetla (ADDR physadd, long *buf, int n)
int    alsetb (ADDR physadd, int value)
int    alsetw(ADDR physadd, int value)
long   alsetl (ADDR physadd, long value)
int    alsetba (ADDR physadd, char *buf, int n)
int    alsetwa(ADDR physadd, short *buf, int n)
long   alsetla (ADDR physadd, long *buf, int n)
```

Arguments

physadd	A physical address in the processor board program. Use function **aladdr** to get a program virtual address and **VtoP** to translate a virtual address to its corresponding physical address.
buf	A pointer to a host buffer
n	The number of bytes, words, longwords to be transferred
value	The byte, word or long value to write to processor board address *physadd*.

Description:

These functions transfer data between a Slave Mode program running on the host and a buffer in processor board memory. The **alget** functions transfer data from the processor board buffer to a host buffer while the **alset** functions transfer data from host memory to processor board memory. The memory transfer is performed on the processor board that was selected by the **aldev** function. These functions allow the host to read and write processor board memory. They do not support writes of host memory by the processor board.

The functions that end in the suffix *a,* transfer arrays of bytes, 16-bit words or 32-bit longwords between the processor board buffer specified by argument *physadd* and the host buffer specified by argument *buf.* The size of both buffers must be at least *n* bytes, words or longwords as appropriate.

Functions that do not end in the suffix *a* transfer a single byte, word or longword between processor board memory and host memory. In the case of the **alget** functions, the value read from processor board physical address *physadd* is the function return value. The **alset** functions write the byte, word or longword in argument *value* to physical address *physadd*.

Buffers that are shared by the host and i860 processors should be uncached. Functions alsetb, alsetw and alsetl may perform a read-modify-write in processor board memory. Data could be overwritten if user software allows the host and processor board application to modify the same buffer in processor board memory simultaneously.

Return Values:

On AL860-AT processor boards, algetba, algetwa, algetla, alsetba, alsetwa and alsetla are implemented as void functions. They have no return value. All other processor boards return SUCCESS or FAILURE as shown below.

algetb	byte at address *physadd*
algetw	16-bit word at address *physadd*
algetl	32-bit longword at address *physadd*
algetba	SUCCESS or FAILURE
algetwa	SUCCESS or FAILURE
algetla	SUCCESS or FAILURE
alsetb	none
alsetw	none
alsetl	none
alsetba	SUCCESS or FAILURE
alsetwa	SUCCESS or FAILURE
alsetla	SUCCESS or FAILURE

Support:

AL860 and FT200 series processor boards

Example:

```
/* Slave Mode Program */
char hostbuf[SIZE];             /* input-output data */
ADDR V_func;
ADDR V_procadd, P_procadd;

V_func = aladdr(“_func”);
V_procadd = aladdr(“_global_array”);
P_procadd = VtoP(V_procadd);

alsetba(P_procadd, hostbuf, SIZE);     /* copy input to board */
alcall(V_func, 1, (long) V_procadd);   /* process data */
alwait();

algetba(P_procadd, hostbuf, SIZE);     /* retreive output data */
```

See also:

aladdr
VtoP

algeterrorlevel / alseterrorlevel / errmsg

C Usage:

```
#include <allib.h>
void   alseterrorlevel (int level)
int    algeterrorlevel (void)
void   errmsg (int level, char *fmt, ...);
```

Arguments

level	One of the error levels in the table below
fmt	A variable length string

Description:

The functions **alseterrorlevel, algeterrorlevel and errmsg** allow the user a measure of control over the generation of messages from within the Slave Mode Library, the User Extensibility library code or the processor board Kernel code.

Function **alseterrorlevel** causes the current error level to be set to one of the following. These symbols are defined in the include file allib.h.

Error Levels	
ERRMSG_NONE	No error messages generated
ERRMSG_ERROR	Only fatal error messages are generated
ERRMSG_WARNING	Warnings and errors are generated
ERRMSG_TRACE	Enable trace messages

If the current error level is set to ERRMSG_NONE, then no messages of any sort are generated (even in the presence of errors). If error level is set to ERRMSG_ERROR, then only fatal errors messages are generated. If error level is set to ERRMSG_WARNING, warnings and errors are generated. ERRMSG_TRACE enables trace messages (no Alacron software generates trace messages). The default error level is ERRMSG_ERROR.

The function **algeterrorlevel** returns the current error level. The function **errmsg** is provided to allow user application code to generate messages that are subject to the current error level. The *level* argument to **errmsg** specifies ERRMSG_ERROR, ERRMSG_WARNING, or ERRMSG_TRACE. If the current error level is set high enough, the message will be generated.

Return Values:

For **algeterrorlevel,** the current error level. No return value for the other functions.

Support:

AL860-AT processor boards only.

Example:

```
int level;

alseterrorlevel(ERRMSG_TRACE);
level = algeterrorlevel(void);
if (level != ERRMSG_TRACE) {
      errmsg(ERRMSG_ERROR, “error setting error message level”);
      }
```

See also:

none

C Usage:

```
#include <allib.h>
int    algetstatus (void)
void   alprtexitstatus (void)
int    mpgetstatus (int proc)
void   mpprtexitstatus (int proc)
```

Arguments

proc	The processor number.

Description:

The functions **algetstatus** and **mpgetstatus** enable a Slave Mode program to retrieve the completion status of a processor board function that has terminated. These functions return a completion code which indicates specific types of normal and abnormal termination.

Function**s alprtexitstatus** and **mpprtexitstatus** print text describing the completion code to the standard error output. Completion codes are as follows:

0	Normal Termination
1	Instruction Trap
2	Interrupt Trap
3	Instruction Access Trap
4	Data Access Trap
5	Floating Point Trap
6	Kernel Problem
7	Integer Overflow
8	Illegal Instruction
9	Missing Unlock
10	Unknown Trap
11	Page Fault

12	Uncaught Signal
13	Alarm Clock
14	Bad Signal

Functions **algetstatus** and **alprtexitstatus** retrieve the completion status of processor zero on the processor board. Applications with a multiple processor board may use **mpgetstatus** and **mpprtexitstatus** to select the specific processor whose exit status will be queried.

Return Values:

Functions **algetstatus** and **mpgetstatus** return a completion code. Functions **alprtexitstatus** and **mpprtexitstatus** have no return value.

Support:

algetstatus	AL860 and FT200 series processor boards
alprtexitstatus	AL860 and FT200 series processor boards
mpgetstatus	FT200 series boards only
alprtexitstatus	FT200 series boards only

Example:

```
int code, proc;

proc = 0;
mpcall (proc, aladdr ("_main"), 0);
mpwait ();
code = mpgetstatus (proc);
if (code != 0)
{
      mpprtexitstatus (proc);
      exit (code);
}
```

See also:

alcall
alwait
mpcall
mpwait

algetiresult algetfresult algetdresult

C Usage:

```
#include <allib.h>
long         algetiresult (void)
double       algetdresult (void);
float        algetfresult (void)
long         mpgetiresult (int proc)
double       mpgetdresult (int proc);
float        mpgetfresult (int proc)
```

Arguments

proc	The processor number

Description:

These functions provide a mechanism for retrieving the value returned from an i860 function that was invoked using the **alcall** or **mpcall** functions. The i860 program may exit with a statement of the form:

```
return(n)  ;
```

where *n* is a long integer, float or double. In these cases, the function return value *n* may be retrieved by a Slave Mode program using **alget[ifd]result** or **mpget[ifd]result**.

The functions **mpgetiresult, mpgetfresult** and **mpgetdresult** are multiprocessor versions of the functions **algetiresult, algetfresult** and **algetdresult.** The *mp* versions of the functions accept an argument *proc* which specifies the processor from which the Slave Mode program requests a return value. The *al* versions of the functions assume that the processor number is zero. On single processor boards such as the AL860 series, it only makes sense to use the *al* versions of the functions and consequently, the *mp* functions are ***not supported on AL860-AT boards***. Semantically, the *al* functions are equivalent to the *mp* functions with a *proc* argument of zero. For example, the function call **mpgetdresult(0***)* is equivalent to the function call **algetdresult.**

If the host application passes an invalid processor argument to these functions the application will exit with an error message displayed.

Return Values:

Functions **algetiresult** and **mpgetiresult** return a 32 bit integer value. Functions **algetfresult** and **mpgetfresult** return a 32 bit floating point result (single precision), and **algetdresult** and **mpgetdresult** return a 64 bit floating point result (double precision). For correct operation, the user must call the appropriate **algetresult** or **mpgetresult** function depending on exactly how the processor board function was declared.

Support:

alget[ifd]result	AL860 and FT200 processor boards
mpget[ifd]result	FT200 series processor boards only

Example:

```
/* Slave Mode Program */
int ires;
ADDR V_func;

V_func = aladdr("_func");
alcall(V_func, 1, 0x10);
alwait();
ires = algetiresult();

/* 860 program */
long func(long i)
{
return(i);
}
```

See also:

alcall
alwait
mpcall
mpwait

C Usage:

```
#include <allib.h>
void   alinterrupt (int dev)
void   mpinterrupt (int dev, int proc)
```

Arguments

dev	Processor board unit number
proc	The processor number.

Description:

The **alinterrupt** and **mpinterrupt** functions generate an interrupt to a processor on the processor board specified by argument *dev.* Function **alinterrupt** interrupts processor zero while **mpinterrupt** provides multiple processor support by allowing the user to select a processor to interrupt. The processor board application may install an interrupt handler by calling **atioctl**.

Return Values:

none

Support:

alinterrupt	AL860 and FT200 series processor boards
mpinterrupt	FT200 series boards only

Example:

```
mpinterrupt(0, 0);
```

See also:

atioctl

Host Slave Mode Library Reference

alintstat	**alintstat**
alwait	**alwait**
alwaitany	**alwaitany**
mpintstat	**mpintstat**
mpwait	**mpwait**
mpwaitany	**mpwaitany**

C Usage:

```
#include <allib.h>
int    alintstat (void)
void   alwait (void)
int    alwaitany (void)
void   mpintstat (long devprocmask[MAX_I860_DEVS])
long   mpwait (void)
void   mpwaitany (long devprocmask[MAX_I860_DEVS])
```

Arguments

devprocmask	Argument *devprocmask* is an output argument which is a pointer to a long integer array of size MAX_I860_DEVS. Each element of the array is a bitmask for a different processor board device. The bitmask for device unit number *N* is devprocmask[N]. Each bitmask bit represents a different processor on board N. The bit for processor *M* is 2eM or *two to the Mth power*. Bits in the bitmask are set when the last function, started by **alcall** or **mpcall** on the processor in question, executes a return or exit system call.

Description:

Processor board system calls, such as printf and disk I/O, are implemented via a remote procedure call to the host. System calls in the processor board application pend until the host makes a function call to service them. This group of functions allows the host to handle processor board system service requests. Additionally, these functions allow the Slave Mode program to determine if a function started with **alcall (**or **mpcall**) has completed.

Functions **alintstat, alwait and alwaitany** are intended to be used on single processor boards while functions **mpintstat, mpwait and mpwaitany** are intended for multiple processor boards. The single processor and multiple processor versions of these routines differ in their return arguments not in their functionality.

Functions **alintstat** and **mpintstat** return immediately to the calling program after handling current outstanding requests, if there are any. Functions **alwait, mpwait, alwaitany** and **mpwaitany** also handle system call requests, but do not return to the calling program until at least one function completion request is serviced. Host Slave Mode programs that use **alintstat** or **mpintstat** should call them whenever it is suspected that the processor board has a system call request pending. Often this means calling the functions repeatedly throughout the duration of the Slave Mode program.

Function **alintstat** polls for system call requests from any processor on any device. System calls are handled, if there are any, and **alintstat** returns to the calling program.

Function **alwaitany** also handles system call requests from any processor on any device. However, unlike **alintstat**, **alwaitany** waits for at least one completion request to appear before returning.

Functions **alintstat** and **alwaitany** return a 32-bit device bitmask. Each bit in the bitmask represents a unique processor board device. Processor board *N* is represented by the bit 2eN or *two to the Nth power.* A bit in the device bitmask is set when the host detected that any processor, on the device in question, completed a function that was started with **alcall** or **mpcall**.

Functions **mpintstat** and **mpwaitany** perform like **alintstat** and **alwaitany**, but return an updated *devprocmask* array as described above.

Functions **alwait** and **mpwait** pend until any processor of the board currently selected by **aldev** completes function execution. Function **alwait** has no return value. Function **mpwait** returns a processor bitmask indicating which processors sent the completion request. The bit for processor *M* is 2eM or *two to the Mth power*. Bits in the bitmask are set when the last function, started by **alcall** or **mpcall** on the processor in question, executes a return or exit system call.

Return Values:

alintstat	bitmask of devices which completed a function call
alwait	none
alwaitany	bitmask of devices which completed a function call
mpintstat	*devprocmask* array
mpwait	bitmask of processors which completed a function call
mpwaitany	*devprocmask* array

Support:

alintstat	AL860 and FT200 series processor boards
alwait	AL860 and FT200 series processor boards
alwaitany	AL860 and FT200 series processor boards
mpintstat	FT200 series boards only
mpwait	FT200 series boards only

mpwaitany	FT200 series boards only

Example:

```
mpcall (0, aladdr ("_main"), 0);
mpwait ();
```

See also:

alcall
mpcall

alinvalidate_caches

C Usage:

```
#include <allib.h>
void   alinvalidate_caches (void)
```

Arguments

none

Description:

The host caches the memory management unit (MMU) page table entries (PTE) in order to efficiently perform processor board program virtual to physical address translation. The actual table of PTE's is resident in processor board memory and changes when the either **mapvtop** or **MapVtoP** is called to change MMU mappings dynamically. Changes performed in the real table are not updated in the MMU table copy that is cached in host memory. Function alinvalidate_caches forces the host to reload the current MMU tables from processor board memory.

It would be appropriate to use this function if the host needs to call VtoP after the processor board program uses **mapvtop**.

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
alcall (0, aladdr ("_main"), 0);
alwait ();
alinvalidate_caches();
```

See also:

VtoP
mapvtop
MapVtoP

alload

C Usage:

```
#include <allib.h>
void   alload(char *filename)
```

Arguments

filename	Processor board executable file residing on host disk file system.

Description:

Function **alload** loads the processor board executable file given by *filename* into physical memory of the currently selected processor board. Addresses of all segments in the executable file are taken as absolute and loaded at the physical addresses specified.

Application programs should generally call **almapload** instead of **alload**.

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
alopen (0);
aldev (0);
alload("filename");
```

See also:

alreset
almapload

AllocPage ClrMMU MapVtoP mp_vtop VtoP

AllocPage ClrMMU MapVtoP mp_vtop VtoP

C Usage:

```
#include <allib.h>
#include <al860.h>
ADDR  AllocPage (int numpages)
void  ClrMMU ()
void  MapVtoP (ADDR vaddr, ADDR paddr, int numpages, long flags)
ADDR  mp_vtop (int proc, ADDR vaddr)
ADDR  VtoP (ADDR vaddr)
```

Arguments

flags	Memory Management Unit (MMU) page table flags. See function **mapvtop**.
numpages	Number of pages
paddr	A physical address
proc	Processor number
vaddr	A virtual address

Description:

The MMU routines are used by **almapload** to setup the processor board page tables for operation with the MMU enabled. The functions **MapVtoP**, **AllocPage** and **ClrMMU** are not generally useful to anyone other than the most advanced users. Functions **mp_vtop** or **VtoP** will be used by nearly every Slave Mode program.

Functions **mp_vtop** and **VtoP** return the physical address corresponding to the virtual address given by *vaddr*. **VtoP** looks the address up in a cached copy of the MMU table of processor zero, while **mp_vtop** checks the tables of processor *proc*. Functions **mp_vtop** and **VtoP** return the value zero if the virtual address is not mapped to a physical address. Note, that there is significant overhead in executing these functions. If the physical address of a symbol is to be used more than once, it is more efficient to call **mp_vtop** or **VtoP** once and save the result in a program variable. After the processor board application calls board-level function **mapvtop** (not to be confused with **MapVtoP** which executes on the host), the host program should call **alinvalidate_caches** before calling **mp_vtop** or **VtoP**.

MapVtoP maps *numpages* consecutive pages, starting with the page that contains virtual address *vaddr*, to the physical page containing paddr. Second level page tables are allocated as needed. Second level page table entries are marked dirty and cached. The value of *flags* is ORed into the low order bits of the second level page table entry, providing a means of mapping the pages as uncached or read only. Refer to on-board processor documentation for more information on *flags*.

AllocPage allocates *numpages* pages of physical memory, returning the physical address of the block of pages. Calling **AllocPage** with an argument of -1 reinitializes (frees) all pages.

ClrMMU initializes and clears the directory page table, marking all page table entries ***not present*** and discarding all second level page tables.

Memory Layout:

Processor boards are run with the MMU activated. The MMU maps virtual addresses (addresses as viewed by the program in execution) to physical addresses on the processor board.

All address values are specified as 32 bit hexadecimal numbers. Physical addresses are preceded by a "P", and virtual addresses are preceded by "V". For the processor board Kernel, text and data pages are mapped to the same physical and virtual addresses.

The following Kernel mappings pertain to the AL860-AT only:

AL860 I/O Status Port	IOSTA	V FFFF5000	P 00200000	Writeable, Uncached
AL860 I/O Control Port	IOCTR	V FFFF6000	P 00400000	Writeable, Uncached
AL860 RTC Port	RTC	V FFFF7000	P 00600000	Writeable, Uncached
Kernel data		V FFFF8000	P FFFF8000	Writeable, Uncached
Kernel text		V FFFE0000	P FFFEFFFF	Unwriteable, Cached

Text (code), data, bss, heap and stack areas are located at the following processor board virtual addresses:

text segment	V F0400000
data/bss segment	V 00001000
heap	V top of bss
stack	V 80000000 - STACKSIZE

The size of executable file segments vary according to program size (text, data, bss), and run time specification (stack, heap). These memory pages are allocated from available physical memory.

Stack size is specified when an application is invoked,. The default is 65536 bytes. All remaining physical memory is made available to the heap for dynamic allocation.

Return Values:

AllocPage	physical address pointing to allocated buffer
ClrMMU	none
MapVtoP	none
mp_vtop	physical address of *vaddr* on processor *proc*
VtoP	physical address of *vaddr* on processor 0

Support:

AllocPage	AL860-AT processor boards
ClrMMU	AL860-AT processor boards
MapVtoP	AL860 and FT200 processor boards
mp_vtop	FT200 series processor boards
VtoP	AL860 and FT200 processor boards

Example:

```
ADDR Vadd, Padd;
Vadd = aladdr(“symbol_name”);
Padd = VtoP(Vadd);
```

See also:

aldev
alinvalidate_caches
almapload
mapvtop

almapload

C Usage:

```
#include <allib.h>
void   almapload (char *filename, long stacksize)
```

Arguments

filename	Processor board executable file residing on host disk file system.
stacksize	Size of stack for on-board processors

Description:

Function **almapload** loads the processor board executable file given by *filename* into processor board memory using memory management. Argument *filename* should be a file residing within the host file system. The file should be an executable image built using compiler tools specifically designed for the processor board. Function **almapload** sets up the MMU page tables in order to resolve virtual addresses in the processor board executable file, and allocates stack space in the amount specified by *stacksize*. The argument *stacksize* is the desired stack size specified in bytes. In addition, the Slave Mode Kernel is loaded and initialized.

Upon completion of **almapload**, the on-board processor reset is deasserted and program execution begins. At this point, the on-board Kernel initializes its data areas. Next, the on-board processor(s) wait for a function call request that may be made in a Slave Mode program by using the **alcall** or **mpcall** functions.

Function **almapload** constructs separate MMU tables for each individual processor on the currently selected board.

After a call to **almapload**, the Slave Mode program may look-up the virtual addresses of processor board program symbols using the function **aladdr**. Only the symbols loaded by the last call to **almapload** may be referenced by **aladdr.**

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
alopen (0);
aldev (0);
almapload(“filename”, 0x100000L);
```

See also:

alreset
almapload

C Usage:

```
#include <allib.h>
int    alopen (int dev)
```

Arguments

dev	The processor board device unit number. Normally, the first board is unit 0. The *dev* argument refers to the index provided See the description below.

Description:

The **alopen** function opens the processor board device specified by argument *dev*. The value 0 is returned if the open is successful, -1 otherwise. This function must be called on a device before any other Slave Mode functions may be called.

On MSDOS or Windows 3.X systems, the **alopen** function does not physically reference the device. The **alopen** call will pass if the appropriate environment variable, **AL860CFG** or **AL860XPCFG,** contains an entry for the unit specified by the *dev* argument. Refer to function **alsize** to detect if a device is present.

On Unix systems, the argument *dev* refers to the minor device number of the device special file that corresponds to the desired processor board.

When running OS/2, the *dev* argument is the device unit number that was entered into the CONFIG.SYS file for the specified device.

Return Values:

SUCCESS	Device present. Open succeeded.
FAILURE	Open failed.

Support:

AL860 and FT200 processor boards.

Example:

```
int dev;
int fd[MAX_I860_DEVS];

for (dev=0; dev<MAX_I860_DEVS; dev++) {
```

```
fd[dev] = alopen (dev);
if (fd[dev]==SUCCESS) {
      printf (“Device[%d] opened\n”,dev);
      }
}
```

See also:

alcall
aldev
almapload
alsize

alputs W3Slv860Setup

alputs
W3Slv860Setup

C Usage:

```
#include <allib.h>
void   alputs (char far *text)
void   W3Slv860Setup (HWND hWndMain)
```

Arguments

environ	A pointer to the host application's environment
hWndMain	A window handle
text	The processor board

Description:

The Slave Mode library has been built as a Windows 3.1 Dynamically Linked Library (DLL) which may be linked with any Windows application that wishes to interface to the Alacron processor board. All functions in the standard Slave Mode library are supported in the Windows DLL version. Two additional functions have been provided to handle Windows specific functionality.

The processor board shares the same *stdin*, *stdout* and *stderr* file streams as the Slave Mode application. In MSDOS, *stdout* and *stderr* output are, by default, sent to the console. Windows programs, on the other hand, must name a window to receive *stdout* and *stderr* output. Function **W3Slv860Setup** allows the Windows application to designate a specific window for *stdout* and *stderr* output from the processor board. The argument *hWndMain* specifies a window to which the slave mode library directs DLL debug and error text, output from **alputs** and re-directed processor board program output.

The *environ* argument, that was shown in previous versions of **W3Slv860Setup** is no longer used. The environment information that is required, is embedded in a file in the Windows main directory; default is C:\WINDOWS. The file name that should be used is AL860.INI for AL860 users and FT200.INI for FT200 users. The format of the file is shown below. The semantics of the AL860, AL860CFG and AL860XPCFG variables are described in the MSDOS software installation procedure in the *Processor Board Installation Manual*.

Example AL860.INI file for AL860 series processor boards:

```
[environment]
AL860=C:\USR\AL860\MSDOS
AL860CFG=(0,110,15)
```

Example FT200.INI file for FT200 series processor boards:

```
[environment]
AL860=C:\USR\FT200\MSDOS
AL860XPCFG=(0,110,15)
```

Function **W3Slv860Setup** is optional. By default (or if argument *hWndMain* is null), standard error and standard output are directed to a windows debug screen using the windows SDK function **OutputDebugString**

The **W3Slv860Setup** function assumes that you are using the large memory model for arguments that are passed in.

The **alputs** function takes the text string pointed by argument *text* and displays it using Windows 3 text display functions. Text is directed to the window whose handle was previously specified by a call to **W3Slv860Setup**. Text is displayed using the currently selected font, and the window is scrolled as required. Text is not redrawn when the window is resized.

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
HWND hWndMain;
char far text[]="This is a test string";

W3Slv860Setup(hWndMain);
alputs(text);
```

See also:

none

alsize

C Usage:

```
#include <allib.h>
unsigned long      alsize(void)
```

Arguments

none

Description:

The **alsize** function returns the total memory size of the currently selected processor board. The procedure that is used to size memory writes to descending memory locations at 1 MB intervals. If a program is active on the processor board, the **alsize** function will overwrite processor board application and Kernel data and code areas. For this reason, **alsize** should be called before a program is started on the processor board via a call to **alcall** or **mpcall**.

Return Values:

size	Total size of processor board DRAM.
-1	Processor board not opened.

Support:

AL860 and FT200 series processor boards.

Example:

```
int dev=0;
unsigned long size;


alopen (dev);
aldev(dev);
size = alsize();
printf (“Size of DRAM = 0x%X\n”,size);
```

See also:

alcall
aldev

C Usage:

```
#include <allib.h>
void   alreset (int enable)
```

Arguments

enable	A value of non-zero asserts processor board reset. A value of zero deasserts reset.

Description:

The **alreset** function asserts or deasserts the reset line of the currently selected processor board. Applications will not normally need to call this function as reset is accessed by the **almapload** function. This function has no effect on processor board memory.

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
int dev=0;
unsigned long size;


alopen (dev);
aldev(dev);
alreset(1);
```

See also:

aldev
alopen

4

Processor Board Library Reference

Application code that runs on the Alacron board is linked with the processor board runtime library i860lib.a. This library provides support for features that are not addressed in the standard ANSI C or FORTRAN specification. This section is a reference for the generic set of runtime library functions which are available on all Alacron processor boards.

Functions that provide multiple processor support for FT200 boards are described in the next section.

C Usage:

```
#include <i860lib.h>
int    atioctl (int command, ...)
```

Arguments

command	One of the optional commands from the list below
...	Optional argument(s) required by specific **atioctl** commands. (Refer to the *funcname* argument below and the AT_ATTACHINT command)
funcname	A function pointer which will be installed as the host interrupt service routine. The function will be invoked whenever the host performs an **alinterrupt** or **mpinterrupt** call.

Description:

The **atioctl** function provides a mechanism for allowing user application code running on the processor board to be called whenever the host generates a host interrupt by calling Slave Mode functions **alinterrupt** or **mpinterrupt**. The argument *command* may take on the following values:

AT_ATTACHINT	A second argument *funcname* contains the address of a function that is to be called whenever an interrupt is received from the host from the host (AT) processor. The function will be invoked whenever the host performs an **alinterrupt** or **mpinterrupt** call. The interrupt handler should be of type void* and is called with no arguments.
AT_ENABLE	Enables interrupts from host processor. No second argument required.
AT_DISABLE	Disables interrupts from host processor. No second argument required.

Note, the reference to "AT" (as in PC/AT) is retained purely for historical reasons. This function is supported on PCI and VME boards also.

Return Values:

SUCCESS	Function succeeded.
FAILURE	Function failed.

Support:

AL860 and FT200 series processor boards.

Example:

```
void funcname(void)
{
/* host interrupt handler code here */
}

/* somewhere in program main body */
atioctl(AT_ATTACHINT, funcname);
atioctl(AT_ENABLE);
```

See also:

alinterrupt
mpinterrupt

imalloc ufree umalloc upmalloc

C Usage:

```
#include <i860lib.h>
void   imalloc()
void   ufree (void *buffer)
void   *umalloc (int numbytes)
void   *upmalloc (int numpages, int writeprotect);
```

Arguments

buffer	A pointer to a buffer previously allocated by umalloc.
numbytes	The number of bytes to allocate.
numpages	The number of pages to allocate.
writeprotect	The write protect flag. If *writeprotect* is non-zero, then the allocated pages will be write protected. Writes by the on-board processor to write protected pages result in page faults. If *writeprotect* is zero, the allocated pages will not be write protected.

Description:

Function **imalloc** initializes the multiple processor safe memory allocation routines. It must be called on processor 0, when processor 1 is idle, before an application may call any multi-processor allocation routines. The functions malloc, calloc, realloc and free have less efficient, but multiple processor safe, implementations. When an application links with i860lib.a, it uses the standard single processor version of these functions. When the library libcr.a is linked ahead of i860lib.a, calls to malloc, calloc, realloc and free are safe for multi-processing, multi-thread applications. Library libcr.a is located in the same directory as i860lib.a.

The multiple processor safe versions of the memory allocation routines are reentrant and as such, may be called by either processor at any time during the application after the imalloc routine is called.

Function **umalloc** returns the address of a 32 byte aligned block of memory that is at least *numbytes* bytes in length. This function is modeled after the ANSI standard function **malloc**, except that **malloc** returns a pointer to a *cached buffer* while **umalloc** returns a pointer to an *uncached buffer*. If the request cannot be satisfied, NULL is returned. Buffers that are allocated by **umalloc** may be freed by **ufree**. Memory that is freed is available for subsequent allocation.

The buffer allocated by umalloc is 32-byte aligned and is a multiple of 32 bytes. If “numbytes” is not a multiple of 32, then umalloc will round up. This insures that the buffer returned will not share a cache line with any other data. This is quite important, if your program is going to use multiple buffers that have different cache modes. Accessing the same cache line with two different cache modes may lead to cache incoherency.

Function upmalloc allocates a page-aligned region of *numpages* pages of memory. The page size on AL860 and FT200 processor boards is 4 KB.

Return Values:

imalloc	none
ufree	none
umalloc	A virtual pointer to a 32-byte aligned uncached region of size *numbytes*. If NULL is returned, the allocation failed.
upmalloc	A virtual pointer to a page aligned uncached region of size *numbytes*. If NULL is returned, the allocation failed.

Support:

imalloc	FT200 series processor boards
ufree	AL860 and FT200 series processor boards
umalloc	AL860 and FT200 series processor boards
upmalloc	AL860 and FT200 series processor boards

Example:

```
unsigned char *buffer;

imalloc();
buffer = umalloc (640*480);
if ((int) buffer == (int) NULL) {
       printf (“malloc failed\n”);
       }

/* later */
ufree(buffer);
```

See also:

none

C Usage:

```
#include <i860lib.h>
int    ismapped (void *padd)
int    mapvtop (void *vadd, void *padd, int numpages, int flags)
void   *vtop (void *vadd)
```

Arguments

flags	Memory Management Unit (MMU) flags. See list below.
numpages	The number of pages to allocate.
padd	Physical Address
vadd	Virtual Address

Description:

Function **ismapped** returns non-zero if the physical address specified by *padd* is mapped to a virtual address.

Function **mapvtop** maps *numpages* pages of physical memory starting at the page containing address *padd*, to virtual address space starting at the page containing *vadd*. Argument *flags* specifies the MMU flags used for the second level page table entry and are defined as follows:

0x00000002	make pages writeable
0x00000008	FT200 only: make pages write-through cached. This is only effective when bit 4 (0x00000010) is zero. Snooping will occur only when jumper enabled (factory default).
0x00000010	make pages uncached
0x00000080	FT200 only: make pages 4 Megabytes in size - *numpages* refers to the number of 4 Megabyte pages.

Second level page tables are allocated as needed. No checking is done to keep the program from mapping over existing memory, reserved memory or I/O pages. ***Do not use write-through cached or 4 megabyte pages on an AL860 processor board.***

Return Values:

ismapped	Non-zero if the physical address *padd* is mapped. Zero if it is not.
mapvtop	SUCCESS or FAILURE
vtop	Physical address corresponding to virtual address *vadd*

Support:

AL860 and FT200 series processor boards

Example:

```
unsigned long vadd, padd;

 vadd = (unsigned long) umalloc (640*480);
if ((int) vadd == (int) NULL) {
       printf (“malloc failed\n”);
       }

padd = (unsigned long) vtop((void*)vadd);
```

See also:

MapVtoP
VtoP

C Usage:

```
#include <i860lib.h>
void   inthost (void)
```

Arguments

none

Description:

The **inthost** function provides a mechanism for allowing board level user application code to generate an interrupt to the host computer. The semantic of the interrupt is left to be defined by the user. The runtime software will treat an interrupt generated by **inthost()** as a spurious interrupt.

A good way for the processor board to send an asynchronous signal to the host is by using user extensibility. User extensibility provides a way for the host Slave Mode program to identify the source of an interrupt and to invoke the proper handler code.

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
inthost();
```

See also:

alinterrupt
mpinterrupt

C Usage:

```
#include <i860lib.h>
void  rtcioctl (command, arg)
int command;
union {int ivalue; void (*rtc_handler) ()} arg;
```

Arguments

command	One of the optional commands from the list below
ivalue	An abstract 32-bit value whose semantic is different for each of the commands shown below.
rtc_handler	A function pointer to a function which will be invoked upon receipt of a RTC interrupt.

Description:

The **rtcioctl** function provides a mechanism for allowing user application code to access the Real Time Clock hardware of the processor board. The Real Time Clock delivers periodic interrupts, at a fixed frequency, to the calling processor of the processor board.

The argument *cmd* takes on one of the following values:

RTC_ATTACHINT	A second argument *void (*rtc_handler)()* contains the address of a function that is invoked upon receipt of an RTC interrupt. RTC_ATTACHINT may be called multiple times to chain handlers. Upon receipt of a RTC interrupt, each registered interrupt handler is called in succession. No assumption should be made about the order in which the RTC interrupt handlers are called. The maximum number of interrupt handlers that may be registered (currently 5) is defined by the RTC_NCHAIN symbol in file include/rtc.h.
RTC_DETACHINT	A second argument void (**rtc_handler*)() contains the address of the function to remove from the list of RTC interrupt handler functions. The interrupt handler should have been attached using RTC_ATTACHINT.
RTC_ENABLE	Enables RTC interrupts. By default. RTC interrupts are disabled
RTC_DISABLE	Disables RTC interrupts.
RTC_SETFREQ	A second integer argument *ivalue* contains the requested frequency (in HZ) of RTC interrupts delivered to the calling processor. For example, passing in a value of 10

	for *ivalue*, configures the Real Time Clock to deliver interrupts to the calling processor at a rate of 10 per second. It is not recommended to set the interrupt frequency at higher than 1000 Hz as this forces the processor to spend so much time context switching that it is accomplishing little else. If the processor board is not able to support the desired frequency, it will match it as closely as possible.
RTC_GETFREQ	A second argument *float *pfvalue* contains the address of a 32 bit floating point variable, whose value is set to the actual RTC frequency
RTC_GETCOUNT	A second argument *int *pivalue* contains the address of a 32 bit integer variable, whose value is set to the current RTC count. The current RTC count is a count of RTC interrupts.
RTC_SETCOUNT	A second the argument *int *pivalue* contains the address of a 32 bit integer variable, whose value is used to set the current RTC count

On *FT200-AT, AL860-PCI, and FT200-PCI boards only*, the RTC_SETFREQ command supports additional functionality. Passing in argument *ivalue* as -1 or 0xFFFFFFFF is a special case. In this mode, the clock functions as a 32-bit down counter ticking once every microsecond. The clock starts at FFFF:FFFF and ticks down to 0000:0000. The function **mp_clock_val** may be used to obtain the current value of the clock. (On AL860-PCI and FT200-PCI baords, the hardware counter is actually clocked at 2.5 Mhz, and the **mp_clock_val** function converts it to 1.0 Mhz). When the clock reaches zero, an interrupt is delivered if RTC_ENABLE has been called. Afterward, the clock rolls over. In other words, it reloads the value FFFF:FFFF and continues to count down.

With RTC interrupts disabled, this mode is especially useful for timing critical sections of code. Application code can obtain the current value of the clock (using function **mp_clock_val**), execute a critical section of code and then get another clock value. The difference between the two clock values is the number of microseconds that have elapsed during the execution of the code. Robust code should handle the case in which the clock rolls over between the first and second calls to function mp_clock_val. This feature is not available on other processor boards.

Return Values:

none

Support:

AL860 and FT200 series processor boards.

Example:

```
void rtc_handler(void)
{
/* Real Time Clock handler */
}

rtcioctl(RTC_DISABLE);
rtcioctl(RTC_SETFREQ, 30);       /* 30 interrupts per second */
rtcioctl(RTC_ATTACHINT, rtc_handler);
rtcioctl(RTC_ENABLE);
```

See also:

atioctl
mp_clock_val

C Usage:

```
#include <i860lib.h>
int    usertrap (func, arg1, arg2, arg3, arg4)
int func, arg1, arg2, arg3, arg4;
```

Arguments

func	A unique function code associated with a specific handler on the host.
arg1, arg2, arg3, arg4	Four arguments to the host handler.

Description:

The function usertrap provides a mechanism for user application code to initiate a user defined system call. In User Extensibility, the developer adds handler code either into the Slave Mode program or the rt860 utility, running on the host, which handles the system call code specified by *func*.

For more information, refer to the section on *User Extensibility* in the *RT860 Software User's Guide.*

Return Values:

SUCCESS	Function succeeded.
FAILURE	Function failed.

Support:

AL860 and FT200 series processor boards.

Example:

```
/* initiate a user-defined system call */
usertrap (1001, 1, 2,3, 4);
```

See also:

none

5

Multiple Processor Extensions

The processor board runtime library (i860lib.a) has been enhanced to provide support for multiple processor boards. These functions will run on any FT200 series card. However, many functions may not be meaningful on a single processor FT200 processor board.

mp_assert
mp_errprintf
mp_printf

C Usage:

```
#include <i860lib.h>
void   mp_assert (int expression)
void   mp_errprintf (char *fmt, ...)
void   mp_printf (char *fmt, ...)
```

Arguments

expression	an expression that evaluates to true or false
fmt	same as ANSI C printf

Description:

These routines are provided as multiprocessor replacements for printf, fprintf(stderr), and assert, respectively. Calls to these functions by multiple processors are serialized. The functions **mp_assert, mp_errprintf** and **mp_printf** are used in the exact same way as standard ANSI C functions **assert,** and **printf**. The function **mp_errprintf** results directs formatted output to the *stderr* device.

Function **mp_assert** is a macro that may be used to test an *expression* anywhere that that expression is expected to be true. If *expression* evaluates to false, i.e. zero, then the program will display a diagnostic message on the standard error output and will exit. The message displayed contains the expression that failed, and the file name and line number of the executing **mp_assert** statement in the application source file.

A processor board application should not generate a print statement from within an interrupt handler.

Return Values:

none

Support:

FT200 series processor boards only

Example:

```
int condition = 0;

mp_printf (“prints to standard output - usually console\n”);
mp_assert(condition != 0);
```

See also:

mp_proc_id
mp_thread_start
mp_thread_query

C Usage:

```
#include <i860lib.h>
void   mp_catch_int (int proc, void (*func)(int))
void   mp_gen_int (int proc)
```

Arguments

func	A function pointer to an interrupt handler
proc	A processor number

Description:

These routines allow processors to send interrupts to one another. The function **mp_catch_int** is used to install an interrupt handler. The handler *func* will be invoked in response to an interrupt from another processor. When *func* is called, it will be passed an integer argument containing the number of the processor that sent the interrupt. The function **mp_gen_int** sends an interrupt to processor *proc*.

Return Values:

none

Support:

FT200 series processor boards only

Example:

```
/* handle interrupts from other processor */
void func(int proc)
{
/* handle interrupt from processor proc */
}

/* main program on processor 0 */
mp_catch_int(1, func);  /* install handler for interrupts from proc 1 */
mp_gen_int(1);  /* interrupt processor 1 */
```

Multiple Processor Extensions

See also:

mp_proc_id
mp_thread_start
mp_thread_query

mp_clock_val   mp_clock_val

C Usage:

```
#include <i860lib.h>
unsigned long      mp_clock_val()
```

Arguments

none	

Description:

This function returns the current value of the Real Time Clock (RTC) counter.

On the FT200-AT, AL860-PCI, and FT200-PCI, the RTC is a 32-bit down counter that ticks every microsecond. The value returned is the current count of the RTC between 0000.0000 and FFFF.FFFF. When the RTC counts down to zero, it reloads the initial value that was set by **rtcioctl** and continues counting down.

Note, that the 32-bit microsecond down counter is only available on the FT200-AT, AL860-PCI, and FT200-PCI processor boards. For other processor boards, the **mp_clock_val** function is simulated using the **times** library function, and results in timing resolution of 1/100th second.

*Note, that the 32-bit down counter on the AL860-PCI, and FT200-PCI is actually clocked at 2.5 Mhz and is converted to a 1Mhz count by **mp_clock_val**.*

Return Values:

Real Time Clock value

Support:

AL860 and FT200 series processor boards, but only FT200-AT, AL860-PCI, and FT200-PCI processor boards contain 32-bit down counter hardware.

Example:

```
#define TICK 1e-6
unsigned long start, end, diff;
double elapsed;

start = mp_clock_val();

/* code that will be timed */

end = mp_clock_val;
diff = start - end;
elapsed = (double) diff * TICK;
printf (“Code took: %d ticks, %g seconds\n”,diff,elapsed);
```

See also:

mp_proc_id
mp_thread_start
mp_thread_query

C Usage:

```
#include <i860lib.h>
void   *mp_4meg_addr ( void *address, unsigned long int prop )
```

Arguments

address	A 32-byte aligned address of a data buffer. The address should use 4 KB paging.
prop	The desired caching property represented by one of the following constants: MP_KADDR_WBC, MP_KADDR_WTC, MP_KADDR_UNC.

Description:

The Memory Management Unit (MMU) of the FT200 series processor board supports 4 Megabyte paging. The default for applications running on the FT200 is 4 KB paging in order to maintain compatibility with the AL860 series processor board. Use of 4 MB paging can speed-up applications which perform random memory accesses on large buffers. It is recommended that FT200 applications use 4 MB paging on any data buffers much larger than 4 KB. The paging mode used by the i860 MMU affects chip execution speed, but in no way affects software source or binary compatibility.

The entire physical memory space of the FT200 processor board has multiple footprints in the virtual memory space of each processor. In other words, different virtual regions are aliases for the same physical region. However, the different virtual regions are configured with different paging and caching modes.

The Intel 860 processor has an internal data and code cache. A memory region that is *uncached* is never brought into the processor's cache. A memory region that is *write-back* cached is brought into cache on writes. The third caching scheme, *write-through cached*, supports multiprocessor cache coherency by insuring that multiple processors notify each other whenever a memory location that is cached by more than one processor is updated in cache. Software in which data regions are shared between processors (including the host processor) must be written with an awareness of the caching mode being used.

Function **mp_4meg_addr** takes the 4 KB virtual address specified by argument *address* and finds the alias for that address that both uses 4 Megabyte paging and contains the cache property specified by argument *prop*.

Set *prop* to MP_KADDR_WBC to receive a write-back cached address. Argument *prop* should be set to MP_KADDR_WTC for a write-through cached address. Uncached addresses may be obtained by setting *prop* to MP_KADDR_UNC.

NOTE: Due to the caching implementation within the i860, address conversion using **mp_4meg_addr** should only be done on addresses which are cache-row aligned; currently 32-bytes. A 32 byte aligned address may be obtained from function **umalloc**.

When using MP_KADDR_WTC on an AL860 processor board this routine will return an uncached addressed using 4 KB paging.

Return Values:

Returns a virtual address

Support:

This function is supported on all processor boards except the AL860-AT. Four megabyte paging and write-through caching are only supported on FT200 series processor boards.

Example:

```
unsigned char *buf4k, *buf4m;

buf4kb = (unsigned char *) umalloc(0x100000);
if ((int) buf4kb != NULL) {
       buf4m = mp_4meg_addr (buf4kb, MP_KADDR_UNC);
       }
```

See also:

mp_proc_id
mp_printf

mp_sem_acquire mp_sem_allocate mp_sem_free mp_sem_release

C Usage:

```
#include <i860lib.h>
int    mp_sem_acquire (void *sem, int wait)
void   *mp_sem_allocate (void)
void   mp_sem_free (void **sem)
void   mp_sem_release (void *sem)
```

Arguments

sem	An abstract number representing a semaphore
wait	If 0, return only when semaphore has been acquired. If non-zero, try to acquire the semaphore *wait* times,then return even if unsuccessful.

Description:

The **mp_sem_allocate** function returns a semaphore identifier that may be passed to other semaphore routines. The semaphore is initially set to *available.* If no semaphore is allocated, then **mp_sem_allocate** returns NULL.

The **mp_sem_free** routine will return the semaphore whose address is given by *sem* to the semaphore free list. If the semaphore *sem* is subsequently used, undetermined results will occur. The location holding the semaphore address will be set to NULL prior to returning (eg. *sem = NULL).

The **mp_sem_acquire** routine will attempt to acquire the semaphore associated with *sem*. If *wait* is zero, then **mp_sem_acquire** will not return until the semaphore is acquired. If *wait* is non-zero, **mp_sem_acquire** will loop through the acquisition code up to *wait* times to acquire the semaphore. If the semaphore was acquired, then a 1 will be returned. The function will return 0 if the semaphore is unavailable. Function **mp_sem_release** will release a semaphore, making it available to be acquired by other processors invoking **mp_sem_acquire**.

Return Values:

mp_sem_acquire	1 if the semaphore was acquired and 0 if it was not.
mp_sem_allocate	On success, a semaphore is returned. On failure, NULL is returned.
mp_sem_free	none

mp_sem_release	none

Support:

FT200 series processor boards

Example:

```
void *mutex;

mutex = mp_sem_allocate();

for (;;) {
      mp_sem_acquire(mutex, 0);

      /* critical section */

      mp_sem_release(mutex);
      }

mp_sem_free(&mutex);
```

See also:

none

mp_thread_query
mp_thread_start

C Usage:

```
#include <i860lib.h>
int    mp_thread_start (int proc, void (*func)(void *param), void *param)
int    mp_thread_query (int proc, int wait)
```

Arguments

func	A pointer to a function to start on the remote processor.
param	A pointer to an abstract 32-bit number which is passed to function *func.* Argument *param* may have any user-defined specification such as a value or a buffer pointer.
proc	The processor number upon which function *func* will be started.
wait	The flag which indicates whether to wait for the remote thread to complete.

Description:

The **mp_thread_start** function will attempt to start a thread of execution on the processor specified. If successful, the remote processor will begin executing **func* with an initial parameter of *param*.

The **mp_thread_query** routine will return the status of the thread running on the remote processor specified. If *wait* is non-zero, the completion of the call will pend until the remote thread completes execution. If *wait* is zero then the call will immediately return the processor status. Possible return values are listed below.

Note to AL860 and FT200 Users: the *-Mreentrant* compiler flag should be used whenever building a multiple processor application using the Portland Group C or F77 i860 compiler. On dual processor FT200 boards, the processor id numbers are 0 and 1. Processor 0 is the main processor.

Return Values:

mp_thread_query	-1 if an invalid proc was specified or
	0 if the remote processor is idle, i.e. no threads have been started or
	1 if the remote processor is currently executing a thread or
	2 if the remote thread completed successfully.
mp_thread_start	-1 if an invalid *proc* was specified or
	0 if the thread was started successfully.

Support:

FT200 series processor boards

Example:

```
void func(void *param)
{
int pid;
int p;

p = (int) param;
pid = mp_proc_id();
mp_printf (“Processor %d received argument %d\n”,pid, p);
}

/* Processor zero executing the main program */
mp_thread_start(1, func, 101);
func(100);   /* both processors execute func simultaneously */
mp_thread_query(1, 1);
```

See also:

mp_proc_id
mp_printf

6

PCI Processor Board DMA Library

Overview

This section presents a functional description of the DMA library that provides the AL860-PCI and FT200-PCI boards with the ability to DMA data over the PCI bus. This library is not supported on the ISA and VME Alacron processor boards.

The AL860-PCI and FT200-PCI boards support up to two independent and simultaneous DMA transport mechanisms. However, the second channel is currently reserved for host usage. An application can issue a request for a stream of back-to-back DMA transfers without application intervention. This is called a DMA *chain*. This library is multi-processor safe, which means that multiple processors can make calls to the library without creating side effects or affecting each other.

The include file dma.h, which is found in the include directory underneath the RT860 directory tree, contains type definitions, constants, and function prototypes which are used with this library. A module that uses the functions below should include the file dma.h. Applications that use these functions should link with the processor board library dma.a which may be found in the lib directory underneath the Alacron RT860 software root directory.

The DMA library functions will return zero on success and a negative number on failure. Each error code is defined by a constant in dma.h. The following is a list of possible errors:

DMA_E_ACTIVE	A transfer is currently ongoing
DMA_E_BUSY	A transfer is already under way
DMA_E_CHANNEL	An invalid channel was specified
DMA_E_IDLE	There is no transfer occurring
DMA_E_LEN	An invalid length was specified
DMA_E_OPEN	A connection is currently open
DMA_E_NOPEN	A connection is not currently opened

dma_close

C Usage:

```
#include <dma.h>
int    dma_close (int channel)
```

Arguments

channel	A DMA channel number.

Description:

The **dma_close** function will remove a connection with the on-board DMA device. If successful a 0 will be returned, else it will return a negative number indicating the error:

Return Values:

DMA_E_ACTIVE

DMA_E_CHANNEL

DMA_E_OPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret ;

dma_errorprt(1);
iret = dma_close (channel);
if (iret) {
       dma_perror(channel, “dma error\n”);
       }
```

See also:

dma_open

dma_errno

C Usage:

```
#include <dma.h>
int    dma_errno (int channel)
```

Arguments

channel	A DMA channel number.

Description:

This routine will return the last error number reported for the channel specified. If the channel specified is invalid this will return DMA_E_CHANNEL.

Return Values:

DMA_E_CHANNEL

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;
int err;

iret = dma_close (channel);
if (iret) {
      err = dma_errno(channel);
      if (err != DMA_E_CHANNEL) {
            dma_perror(channel, “dma_close failed”);
            }
      else {
            printf(“dma_close or dma_errno failed on invalid channel”);
            }
      }
```

See also:

dma_errorprt
dma_perror

dma_errorprt

C Usage:

```
#include <dma.h>
void   dma_errorprt (int on_off)
```

Arguments

on_off	0 - Disable automatic printing of error messages Non-zero - Enable automatic printing of error messages

Description:

This routine will enable or disable automatic generation of error messages when an error is detected. The *on_off* argument enables or disables messages across all processors present on the processor-board. If the *on_off* parameter is non-zero, then automatic error messages are displayed. Otherwise, error messages will only be displayed if invoked via the dma_perror routine.

Return Values:

none

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_open (channel);
if (iret) {
      printf("dma error\n");
      }
```

See also:

dma_error
dma_perror

C Usage:

```
#include <dma.h>
int    dma_ioctl (int channel, DMA_ABORT)
```

Arguments

channel	A DMA channel number.

Description:

The DMA_ABORT request will stop any active DMA transfer. The status of the transfer (with respect to the amount of data transferred) is not available.

If successful this invocation will return 0. Otherwise, it will return one of the following indicating the error condition:

Return Values:

DMA_E_CHANNEL

DMA_E_IDLE

DMA_E_NOPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_ioctl (channel, DMA_ABORT);
if (iret) {
      dma_perror(channel, “dma error\n”);
      }
```

See also:

DMA_CHAIN
DMA_PAUSE
DMA_RESUME

dma_ioctl (DMA_ATTACHINT)

C Usage:

```
#include <dma.h>
int    dma_ioctl (int channel, DMA_ATTACHINT,
                   (void *handler)(int channel, int status))
```

Arguments

channel	A DMA channel number.
handler	A pointer to a function to start on the remote processor.
status	The status of the channel when *handler* is called

Description:

This function will install the function *handler* as the interrupt handler for interrupts generated by DMA activity on *channel*. When the handler is called it will be passed the *channel* and the *status* of the channel at the time of the interrupt reception. The *status* is a mask of flags defined as follows:

Bit	Constant	Definition
0	DMA_STATUS_DONE	A DMA transfer (or part of a chained transfer) has completed
1	DMA_STATUS_ABORT	A DMA transfer was aborted by an DMA_ABORT request
2	DMA_STATUS_TGTABT	A DMA transfer was aborted due to an abort indication from the target
3	DMA_STATUS_BUSY	A DMA transfer is running due to a chained request on-going

If the handler provided is NULL then interrupt propagation from the Kernel will be disabled. Refer to the DMA_POLL command for further information regarding the status bits.

If this function is successful this invocation will return 0. Otherwise, it will return one of the following indicating the error condition:

Return Values:

DMA_E_CHANNEL

DMA_E_NOPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
void handler (int channel, int status)
{
if (status & DMA_GET_STATUS_DONE) {
      /* insert done handler here */
      }

if (status & DMA_GET_STATUS_ABORT) {
      /* insert abort handler here */
      }

if (status & DMA_GET_STATUS_TGTABT) {
      /* insert target abort handler here */
      }

if (status & DMA_GET_STATUS_BUSY) {
      /* insert busy handler here */
      }
}


int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_ioctl (channel, DMA_ATTACHINT, handler);
if (iret) {
      dma_perror(channel, “dma error\n”);
      }
```

See also:

DMA_CHAIN
DMA_ABORT
DMA_PAUSE
DMA_POLL
DMA_RESUME

C Usage:

```
#include <dma.h>
int    dma_ioctl (int channel, DMA_CHAIN, dma_chain_t *chain_head)
```

Arguments

chain_head	A 16-byte aligned physical address which points to a linked list of DMA chain packets
channel	A DMA channel number..

Description:

This command allows the caller to submit a chain of DMA operations, which will be executed one after the other by the DMA controller. The argument *chain_head* is a pointer to a memory resident linked list of DMA request packets. Argument *chain_head* should be a 16-byte aligned physical address which contains the address of the first packet in the chain. The linked list supplied to this function should not be altered after it has been submitted to the DMA engine. Each packet, or DMA chain template, in the list is of type dma_chain_t:

```
typedef struct {
      unsigned long       pci_address;
      unsigned long       local_address;
      unsigned long       transfer_size;
      unsigned long       next_descriptor;
} dma_chain_t;
```

The first field, *pci_address,* contains the remote PCI address to transfer to or from. The *local_address* field specifies a 32-byte aligned, on-board, physical DRAM address. The *transfer_size* field specifies the number of bytes to transfer (must be less than 8 MB). The *next_descriptor* field contains the following components:

Bit	Description
0	Reserved, set to zero
1	End of chain indicator. A 1 value indicates an end-of-chain, a 0 value indicates that the upper bits contain the physical address of the next link in the chained request.
2	Interrupt after terminal count is reached. If set to 1 then an interrupt will be generated after this element has completed. A 0 value will cause the hardware to not generate an interrupt after this element. [Note: An interrupt is always generated at the end of the chain, and intermediate interrupts will be passed on to the application if the

	DMA_ATTACH_INT request has been made.]
3	Transfer direction: a 1 value indicates transfer from the local to the remote address, a 0 value indicates transfers from the remote to the local address.
4-31	Next descriptor physical address. If bit 1 is non-zero then the next chain element will be addressed using this 16-byte aligned address.

Note that each chain element (including the one pointed to by *chain_head*) must be an uncached 16 byte aligned physical address. This can most easily be accomplished using the standard **umalloc** interface. Note also that creating a circular chain is allowed. In other words, any chain element can point to any previous element in the chain.

If successful this invocation will return 0. Otherwise, it will return one of the following indicating the error condition:

Return Values:

DMA_E_ACTIVE

DMA_E_CHANNEL

DMA_E_NOPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
#define SIZE              0x1000
#define PCI_MEM_ADDRESS   0x100000;

int channel = 0;
int iret;
dma_chain_t *chain_head, *chain_buffer;
unsigned long vadd, padd

vadd = (unsigned long) umalloc (SIZE);
padd = (unsigned long) vtop ((void*)vadd);

chain_buffer = (dma_chain_t *) umalloc(sizeof(dma_chain_t));
chain_buffer->pci_addr = PCI_MEM_ADDRESS;
chain_buffer->local_address = padd;
chain_buffer->transfer_size = SIZE;
chain_buffer->next_descriptor = 0x2; /*end chain, no int, read*/
chain_head = (dma_chain_t) vtop (chain_buffer);

dma_errorprt(1);
iret = dma_ioctl (channel, DMA_CHAIN, chain_head);
if (iret) {
      dma_perror(channel, “dma error\n”);
      }
```

See also:

DMA_ATTACHINT
DMA_ABORT
DMA_PAUSE
DMA_POLL
DMA_RESUME

C Usage:

```
#include <dma.h>
int    dma_ioctl (int channel, DMA_PAUSE)
```

Arguments

channel	A DMA channel number..

Description:

The DMA_PAUSE will cause any active DMA transfer to be paused. That is, the transfer will not proceed until resumed using the DMA_RESUME command.

If successful this invocation will return 0. Otherwise, it will return one of the following indicating the error condition:

Return Values:

DMA_E_CHANNEL

DMA_E_IDLE

DMA_E_NOPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_ioctl (channel, DMA_PAUSE);
if (iret) {
       dma_perror(channel, “dma error\n”);
       }
```

PCI Processor Board DMA Support

See also:

DMA_ATTACHINT
DMA_ABORT
DMA_CHAIN
DMA_POLL
DMA_RESUME

dma_ioctl (DMA_POLL)

C Usage:

```
#include <dma.h>
int    dma_ioctl (int channel, DMA_POLL, int *status_p))
```

Arguments

channel	A DMA channel number.
status_p	A pointer to a buffer which will be written with a status mask.

Description:

The DMA_POLL request will return status of the current or last DMA using the following bit positions in the data stored at *status_p:

Bit	Constant	Definition
0	DMA_STATUS_DONE	A DMA transfer (or part of a chained transfer) has completed
1	DMA_STATUS_ABORT	A DMA transfer was aborted by a DMA_ABORT request
2	DMA_STATUS_TGTABT	A DMA transfer was aborted due to an abort indication from the target
3	DMA_STATUS_BUSY	A DMA transfer is running due to an active chained request, or the last DMA has not yet completed.
4	DMA_STATUS_IDLE	The device is idle

Descriptions of different states for the DMA engine:

- If the **DMA_STATUS_IDLE** bit is set, then no other bit will be set, and the status of the last operation has already been obtained (either by a previous invocation of the **DMA_POLL** command or by asynchronous notification via the interrupt handler).
- If **DMA_STATUS_DONE** is the only bit set, then the last DMA operation successfully completed, and the device is currently idle.
- If **DMA_STATUS_DONE** and **DMA_STATUS_BUSY** are both set, then an intermediate DMA operation (part of a chained operation) has completed, but the device is still busy with further chained operations.

- If either (or both) of the two abort bits are set, then the done bit will be set, the busy bit will be clear, and the DMA engine is idle.

If successful this invocation will return 0. Otherwise, it will return one of the following indicating the error condition:

Return Values:

DMA_E_CHANNEL

DMA_E_NOPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;
int status;

dma_errorprt(1);
iret = dma_ioctl (channel, DMA_POLL, &status);
if (iret) {
       dma_perror(channel, "dma error\n");
       }
```

See also:

DMA_ATTACHINT
DMA_ABORT
DMA_CHAIN
DMA_PAUSE
DMA_RESUME

dma_ioctl (DMA_RESUME)

C Usage:

```
#include <dma.h>
int    dma_ioctl (int channel, DMA_RESUME)
```

Arguments

channel	A DMA channel number.

Description:

The DMA_RESUME will resume a DMA transfer that was stopped using the DMA_PAUSE command.

If successful this invocation will return 0. Otherwise, it will return one of the following indicating the error condition:

Return Values:

DMA_E_BUSY

DMA_E_CHANNEL

DMA_E_IDLE

DMA_E_NOPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_ioctl (channel, DMA_RESUME);
if (iret) {
       dma_perror(channel, “dma error\n”);
       }
```

See also:

DMA_ATTACHINT
DMA_ABORT
DMA_CHAIN
DMA_PAUSE
DMA_POLL

dma_open

C Usage:

```
#include <dma.h>
int    dma_open (int channel)
```

Arguments

channel	A DMA channel number.

Description:

The **dma_open** function will establish a connection to the on-board DMA device. If successful, it will return 0, else it will return a negative number indicating the error:

Return Values:

DMA_E_CHANNEL

DMA_E_OPEN

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_open (channel);
if (iret) {
       dma_perror(channel, “dma error\n”);
       }
```

See also:

dma_close

C Usage:

```
#include <dma.h>
void  dma_perror (int channel, char *message)
```

Arguments

channel	A DMA channel number.
message	A user specified, variable length, character string.

Description:

This routine will display the given message and a description of the last error detected for the channel specified.

Return Values:

none

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
int channel = 0;
int iret;

dma_errorprt(1);
iret = dma_open (channel);
if (iret) {
      dma_perror(channel, “dma error\n”);
      }
```

See also:

dma_errno
dma_errorprt

dma_read

C Usage:

```
#include <dma.h>
int    dma_read (int channel, unsigned long remote_address,
                   unsigned long local_address, int numbytes)
```

Arguments

channel	A DMA channel number.
numbytes	The number of bytes to transfer.
local_address	A 32 byte aligned physical address in processor board memory.
remote_address	An address in PCI space

Description:

The **dma_read** routine will initiate a movement of *numbytes* bytes of data from PCI address *remote_address* to an on-board dynamic RAM address *local_address*. Accesses to invalid PCI addresses will produce unreliable results. A board may not DMA to or from itself over the PCI bus. The transfer length, *numbytes*, must be less than 8 MB in length. The address *local_address* must be a 32 byte aligned physical address. A physical address may be obtained by using function **vtop** to convert a virtual address to a physical one. The on-board processor should access the buffer at *local_address* in uncached mode.

There are two mechanisms provided for determining when the transfer is complete: an asynchronous notification can be received using the function provided with the **DMA_ATTACH_INT** dma_ioctl command, or the DMA status can be polled using the **DMA_POLL** dma_ioctl command. If successful, this function will return a 0, otherwise it will return a negative number indicating one of the following error conditions:

Return Values:

DMA_E_ACTIVE

DMA_E_CHANNEL

DMA_E_LEN

DMA_E_OPEN

PCI Processor Board DMA Support

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
#define SIZE              0x1000;
#define REMOTE_ADDRESS    0x100000;  /* a PCI memory slave */

unsigned long padd, vadd;
int iret;

vadd = (unsigned long) umalloc(SIZE);
padd = (unsigned long) vtop ((void*) vadd);

dma_errorprt(1);
iret = dma_read (channel, REMOTE_ADDRESS, padd, SIZE);
if (iret) {
      dma_perror(channel, “dma error\n”);
      }
```

See also:

vtop
DMA_ATTACH_INT
DMA_POLL
dma_write

dma_write

C Usage:

```
#include <dma.h>
int    dma_write (int channel, unsigned long remote_address,
                  unsigned long local_address, int numbytes)
```

Arguments

channel	A DMA channel number.
numbytes	The number of bytes to transfer.
local_address	A 32 byte aligned physical address in processor board memory.
remote_address	An address in PCI space

Description:

The **dma_write** routine will initiate a movement of *numbytes* bytes of data to PCI address *remote_address* from an on-board dynamic RAM address *local_address*. Accesses to invalid PCI addresses will produce unreliable results. A board may not DMA to or from itself over the PCI bus. The transfer length, *numbytes*, must be less than 8 MB in length. The address *local_address* must be a 32 byte aligned physical address. A physical address may be obtained by using function **vtop** to convert a virtual address to a physical one. The on-board processor should access the buffer at *local_address* in uncached mode.

There are two mechanisms provided for determining when the transfer is complete: an asynchronous notification can be received using the function provided with the **DMA_ATTACH_INT** dma_ioctl command, or the DMA status can be polled using the **DMA_POLL** dma_ioctl command. If successful, this function will return a 0, otherwise it will return a negative number indicating one of the following error conditions:

Return Values:

DMA_E_ACTIVE

DMA_E_CHANNEL

DMA_E_LEN

DMA_E_OPEN

PCI Processor Board DMA Support

Support:

AL860-PCI and FT200-PCI processor boards.

Example:

```
#define SIZE              0x1000;
#define REMOTE_ADDRESS    0x100000;  /* a PCI memory slave */

unsigned long padd, vadd;
unsigned char *cbuf;
int iret;
int i;

vadd = (unsigned long) umalloc(SIZE);
padd = (unsigned long) vtop ((void*) vadd);

/* initialize data buffer with a ramp (to be written out) */
cbuf = (unsigned char *) vadd;
for (i=0; i<SIZE; i++) {
      cbuf[i] = (unsigned char) i % 256;
      }

dma_errorprt(1);
iret = dma_write (channel, REMOTE_ADDRESS, padd, SIZE);
if (iret) {
      dma_perror(channel, “dma error\n”);
      }
```

See also:

vtop
DMA_ATTACH_INT
DMA_POLL
dma_read

A

System call support

The following table describes which system calls are supported under each of the configurations. *A* means all systems support the call. *D* means the call is supported on MS-DOS systems only, and *U* means the call is supported on UNIX systems only.

Multiple processor applications should use semaphores around system calls to insure that only one processor calls a particular function at a time. System calls are not reentrant.

System Call	AL860-AT	FT200-AT FT200-PCI AL860-PCI	FT200-V
accept	U	U	U
access	A	A	A
alarm	A	A	A
bind	U	U	U
brk	A	A	A
chdir		A	A
chmod	A	A	A
chown		U	U
chroot	U	U	U
close	A	A	A
connect	U	U	U
creat	A	A	A
dup	A	A	A
exit	A	A	A
fcntl	U	U	U
fstat	A	A	A
getegid		U	U
geteuid		U	U
getgid		U	U
gethostbyname	U	U	U
gethostname	U	U	U
getpeername	U	U	U
getpid	A	A	A

System Call Support

System Call	AL860-AT	FT200-AT FT200-PCI AL860-PCI	FT200-V
getsockname	U	U	U
getsockopt	U	U	U
getuid	U	U	U
inet_addr	U	U	U
ioctl			U[1]
link	U	U	U
listen	U	U	U
lseek	A	A	A
mkdir	A	A	A
mknod	A		
msgctl	U	U	U
msgget	U	U	U
msgsnd	U	U	U
msgrcv	U	U	U
noop			A[2]
open	A	A	A
pause	A	A	U
poll	U	U	U
read	A	A	A
recv	U	U	U
recvfrom	U	U	U
rename		A	A
rmdir	A	A	A
sbrk	A	A	A

[1] Supports TCGETA & TCSETA only
[2] Used for timing analysis only.

System Call Support

System Call	AL860-AT	FT200-AT FT200-PCI AL860-PCI	FT200-V
select	U	U	U
semctl	A	U	U
semget	A	U	U
semop	A	U	U
send	U	U	U
sendto	U	U	U
setgid		U	U
setpgrp	A	U	U
setsockopt	U	U	U
setuid	A	U	U
shutdown	U	U	U
signal	A	A	A
socket	U	U	U
stat	A	A	A
time	A	A	A
times	A	A	A
ulimit	A	A	A[3]
umask	A	A	A
uname	A	A	A
unlink	A	A	A
utime			U
write	A	A	A

[3] Except for LynxOS

System Call Support

B

Getting Help

Troubleshooting

If you have turned to this section, you are probably in the middle of application development and are seeing unexpected behavior in hardware and/or software. Alacron offers a variety of technical support services that are designed to assist our user community, but before you call us, there are a few simple things that you should check.

Check the release notes for any relevant explanations. All known bugs and potential problems have been noted for you.

We recommend a systematic approach to help pinpoint your problem. You first want to try to determine if the problem is related to the Alacron boards that you are using or not. Try running the diagnostics and a short “hello world” program. You want to be sure that your processor board is working properly and that the RT860 software is installed and configured correctly.

If you do have a hardware problem, you may return your board(s) directly to Alacron for servicing. See the section below for information on contacting Alacron Technical Support.

If you believe that your problem is in software, check the reference section in this manual and the programming examples supplied with the RT860 software to insure that you are calling the processor board libraries correctly. Check the return status from functions and any input arguments. Simplify the program as much as possible to isolate the failing condition. You can do this by removing any extraneous code that doesn’t directly contribute to the failure (don’t forget to save your original work).

Technical Support

Alacron offers technical support to any licensed user during normal business hours (EST). We will offer assistance on all aspects of board installation and operation. If you wish to speak with a Technical Support Representative on the telephone, please call the number shown below and ask for *Technical Support*. In situations which involve a great deal of detail, it may be more convenient for you to contact us by fax or electronic mail.

If you have a relatively short piece of code that exhibits a problem, fax it to us and our Technical Support Representative will type it in and try to reproduce your error. We can serve you faster if you send us sample code over the Internet. Sample code is easiest to review if any extraneous operations that aren't directly related to the problem are deleted. Note: we may not be able to run your code if we cannot replicate your hardware environment.

Alacron, Inc.
71 Spitbrook Road, Suite 204
Nashua, NH 03060 USA

telephone:	(603)891-2750
fax:	(603)891-2745

electronic mail:
sales@alacron.com
support@alacron.com

Before You Call

Please help us serve you better by having the following information ready:

- The serial numbers and hardware revisions of all of your boards. This information is written on the invoice that is shipped with our products. Also, each board has it's serial number and revision written on it either in pen or on a bar code.
- The version of Runtime and other software that you are using.
- The host operating system.
- The type of system that you are running on.

Returning Hardware For Repair

If you are convinced that your hardware is in need of repair, call Alacron and request a Return Materials Authorization (RMA) number. Address the shipment to Alacron at the address above, *Attention: RMA #########* where the '#" marks represent the RMA number. The RMA number is our means of matching a returned shipment to our problem report database. When the technician starts to work on your boards, she'll have access to all of the information that you have given us.

When you call for an RMA, please have the following information ready:

- Serial numbers of all boards that will be shipped back.
- If you are returning an AL860 or FT200, please note the number of processors (1 or 2), the processor speed, and the size of memory.
- Provide an unambiguous description of the problem. If your board isn't failing our standard diagnostics, please provide us with code that will cause

the failure or give us an exact description of the conditions that will elicit the problem. Note if the problem is sensitive to some environmental condition. This information will be logged in the RMA report and will be reviewed by the technician responsible for fixing your board.

Reporting Bugs

We continually strive to improve our product quality in order to best guarantee the success of your project. Occasionally, however, problems will surface in the field that were not encountered during our extensive testing. If you experience a hardware or software anomaly, please contact us immediately for assistance. Often, if a fix is not available immediately, we can devise a work-around that allows you to move forward.

It is important that we are able to reproduce your error in an isolated test case. Create a stand-alone code module that is isolated from your application and demonstrates the flaw. Describe the error within the code module and email the file to us at **support@alacron.com**. We will compile and run the module, and track down the anomaly in question. If you do not have Internet access, copy the code to a disk and mail it to us, or if small enough, fax a listing of the code module to us at **603-891-2745**.

When describing the software problem, please include the revision of all associated software. This includes the libraries, the Portland Group Tools, the host C compiler, and any Alacron development software. Also indicate the Alacron processor board (FT200/AL860) in use and any attached daughter cards.

For documentation errors, photocopy the page(s) of the manual in question and clearly mark the sections of interest. Along the top of the page(s), write the name and revision level of the manual. Then fax the page(s) to us at **603-891-2745**.

Index