ALACRON

FASTSERIES

FASTMOTION LIBRARY
USER'S MANUAL

30002-00395

COPYRIGHT NOTICE

Copyright © 2008 by Alacron Inc.

Document Name:	Fast Motion User's Manual	
Document Number:	30002-00395	
Revision History:	1.0	Jan 25, 2003
	2.0	April 10, 2003
	3.0	June 11, 2008

Trademarks:

Alacron Inc.
71 Spit Brook Road, Suite 200
Nashua, NH 03060
USA

Telephone: 603-891-2750
Fax: 603-891-2745

Web Site:
http://www.alacron.com/

Email:
sales@alacron.com, or support@alacron.com

TABLE OF CONTENTS

<u>OTHER ALACRON MANUALS</u>

Alacron manuals cover all aspects of FastSeries hardware and software installation and operation. Call Alacron at 603-891-2750 and ask for the appropriate manuals from the list below if they did not come in your FastSeries shipment.

30002-00148	ALFAST Runtime Software Programmer's Guide & Reference
30002-00153	Fast I/O Hardware User's Manual
30002-00162	FOIL – **F**astSeries **O**bject **I**maging **L**ibrary User's Manual
30002-00170	ALRT, ALFAST & FASTLIB Software Installation Manual for Linux
30002-00176	FastImage 1300 Hardware User's Manual
30002-00180	Fast4 1300 Hardware User's Manual
30002-00184	FastSeries Getting Started Manual
30002-00183	FastImage 1300 Camera Integration User's Manual
30002-00187	FastFrame 1300 Hardware User's Manual
30002-00190	FastFrame 1303 Hardware User's Manual

INTRODUCTION

This manual covers the **Alacron FastMotion Library** (AFML)software. The **Alacron Fast Motion Library** enables the software developer to operate Alacron frame grabbers from the host without special knowledge of the internals of the frame grabber.

FASTMOTION LIBRARY

A. INTRODUCTION

Purpose

This manual provides calling specifications and descriptions for the Alacron FastMotion Library of fast image capture functions.

Audience

This section of the manual is intended for technical personnel responsible for developing video capture application software to run on Alacron boards. This manual assumes familiarity with operating system commands to configure the software and with the C programming language.

B. FASTMOTION LIBRARY

The Alacron FastMotion Library for the Fast Series family of processor boards is based on Alacron Runtime (ALRT) software. The FastMotion Library enables capturing sequences of high frame rate images from a variety of digital and analog sources into system memory (RAM). The FastMotion Library supports developing capture applications under Microsoft Windows 2000 and Microsoft Windows XP.

This section provides an overview of the FastMotion Library data types and functions. Function calling sequences, return values and other specifics are provided in the next chapter.

Library Versions

The FastMotion Library is distributed with a dynamically linked library for the host computer and with a set of board firmware and capture profile files suitable for the different types of boards and input sources. Host programs that wish to use FastMotion Library functions should link to alfml.lib. In Linux and Solaris systems the library is Libalfml.lib.

Include File

Application programs using the FastMotion Library should include **<alfml.h>,** which is in the **%ALFAST%\include** installation directory.

Data Images

An image is characterized in memory by its, number of rows, number of columns, pixel size,

Stride and image type.

Summary of Functions

The FastMotion Library contains the following functions:

AlfmlOpen

AlfmlClose

AlfmlSetCallBack

AlfmlStartGrab

AlfmlStopGrab

AlfmlGetImage

AlfmlGetLastImageIndex

AlfmlGetBufferSize

AlfmlGetFramesCount

AlfmlSetChannel

AlfmlGetLastError

AlfmlGetErrorStr

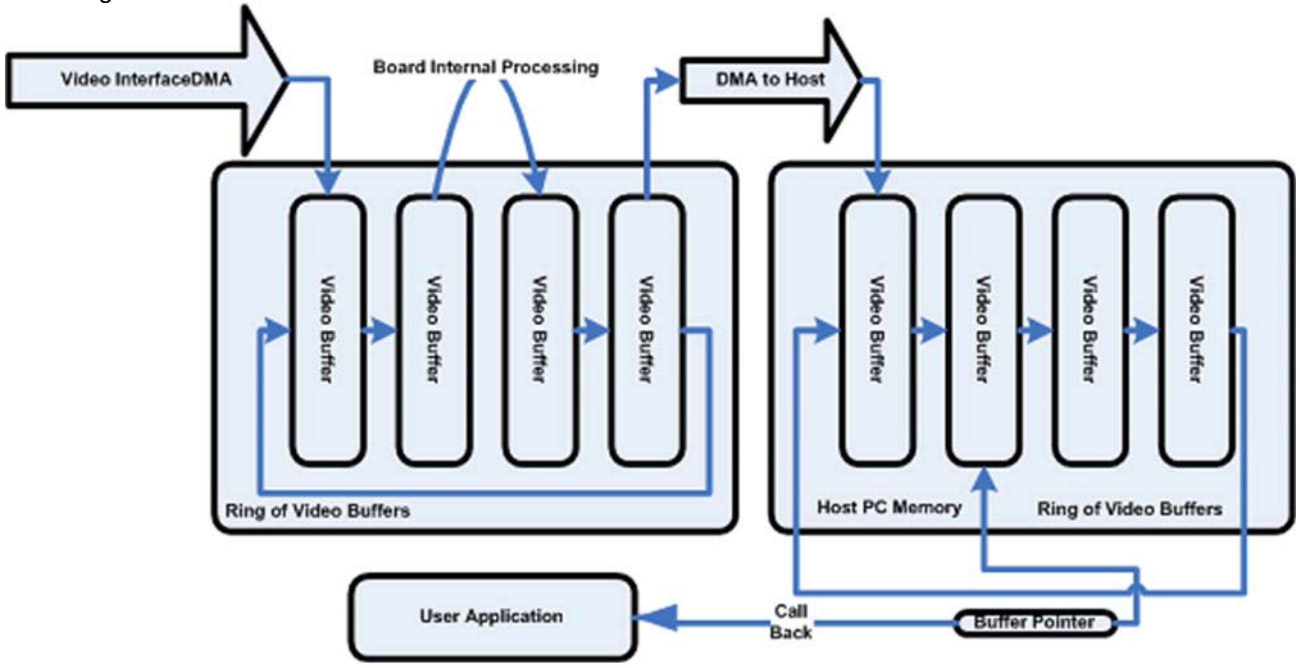
AlfmlGetFlag

AlfmlSendCommand

AlfmlGetdbbuf

C. *FASTMOTION LIBRARY STRUCTURE*

The FastMotion Library supports users who do not want to develop intimate knowledge of the inner workings of the frame grabber board.



The data flows by successive DMA operations. Video data from the video source is DMAed into board memory, into a ring of buffers. At the same time the board does processing on a buffer to an output buffer. Again at the same time the DMA controller is transferring a completed buffer to the host memory. The FastMotion Library traps an event from the board which identifies the new filled buffer in memory. The thread in the FastMotion Library (not a user thread) makes a callback to a user provided callback function and provides a buffer index, which is turned into a buffer address by a separate API. It is assumed all the DMAs and host processes can keep up data flows.

In the case where data flow is faster than processing, the slow part does not process or transfer its current buffer, then the DMA or processing advances to the next buffer to output into, which if not available, the source buffer is dropped as though it had been emptied by the transfer or DMA operation.
The buffer which is passed to the callback function is not protected from DMA when passed to the user's callback function. Should the user need to hold the data for a long time it must be copied to a separate memory area. The Depth of the ring buffers is settable up to the limit of available memory.

Memory Use

The FastMotion Library uses a contiguous memory block for the ring buffers. The block of memory is obtained from the operating system (Windows, Solaris) or is blocked from use by the OS (Linux).

This size of the buffer is determined in different ways depending on the OS, but ultimately ends up in a variable called DMABUF_LENGTH. On Windows a parameter in the registry call dmabuf_length, determines the size of the buffer. On Linux the buffer is created by using a boot parameter call 'mem' (mem=<size-of-OS-Memory). The memory remaining is used by the library. On Solaris the buffer size is determined by a shell command called tmhostbuf.

Fast Motion Library competes for memory with other applications, preventing it from obtaining the desired size buffer. The user must adjust the buffer size to find a setting which is compatible with his application.

FAST MOTION LIBRARY REFERENCE

This chapter provides detailed documentation on the functions in the FastMotion Library.
Each function is listed on a separate page showing input arguments, output arguments, and execution functionality.

AlfmlOpen

C Usage

```
int AlfmlOpen(char* filename, imdev_t* fgrab)
```

Arguments

Filename	A valid path and file name of the capture profile file.
Fgrab	A handle to the frame grabber if it exists. This handle must be used in other library functions that refer to the same session.

Description

This function attempts to find a frame grabber, to initialize it according the selected capture profile file and to establish a communication link between the frame grabber and the program.

Return Values:

On success, this function returns zero value and sets the fgrab variable to a valid handle. On failure, this function returns an error code and sets the fgrab variable to Null. To get extended error information, call AlfmlGetLastError().

Example:

```
…
int iAlRes;
imdev_t fgrab;
…
iAlRes = AlfmlOpen(“c:\usr\alfast\lib\capture\eia.cap”, &fgrab);
if(!iAlRes)
{
       // ShowErrorMessage..
}
```

AlfmlClose

C Usage

```
void AlfmlClose(imdev_t fgrab)
```

Arguments

Fgrab	A handle to the frame grabber that previously allocated with AlfmlOpen() function.

Description

Releases control of the frame grabber. After calling this function the fgrab handle is no longer valid.

AlfmlSetCallback

C Usage

```
int AlfmlSetCallback(imdev_t fgrab, void* Func, void* userData)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
Func	Address of the callback function.
userData	Pointer to a user structure that will be returned to the user as a callback parameter.

Description

Connects a callback function that will be fired every time an event received from the frame grabber. The event are received at the end of the buffer and after acquiring every imagePerEvent images.

The callback function definition is:

```
int AlfmlCallBack(int reason, void* userData, int imageIndex)
```

Example:

```
int MyCallBack(int reason, void* dlg, int index)
{
      MyDlgType* aDlg = (MyDlgType*)dlg ;

      if(reason == 1) // new image grabbed.
      else if(reason == 2) // The buffer is full.
}

// this structure is designed by the user.
UserData_t userdata;

alfmlSetCallBack(fgrab, MyCallBack, &userdata);
```

AlfmlStartGrab

C Usage

```
int AlfmlStartGrab(imdev_t fgrab, int grabMode, int
                   imagesPerEvent, int skipCount)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
grabMode	The type of acquisition to perform. Can be GRAB_CONTINUOUS or GRAB_ONE_SHOT.
imagesPerEvent	The number of images that are acquired before calling the Callback function with ImageReady event.
skipCount	The number of frames to skip between each acquired image. A value of 0 acquires all the images.

Description

Starts a continuous acquisition. The grabMode parameter can have two values, which are defined, in the library header file:

AL_GRAB_CONTINUOUS - The board grabs images continuously and when it reaches the end of the buffer, it continues placing the next image at the beginning of the frame buffer.

AL_GRAB_ONE_SHOT - The board grabs one image. At the end it generates an AL_GRAB_FINISHED event.

During the grabbing the frame grabber can produce events to notify the application when new image is available. The value of the imagesPerEvent parameter determines the number of frames the board acquired before generating an event (AL_IMAGE_READY).

Return Values:

On success, this function returns zero value. On failure, this function returns an error code. To get extended error information, call AlfmlGetLastError().

Example:

```
int iGrabRes;
iGrabRes = AlfmlStartGrab(fgrab, AL_GRAB_CONTINUOUS, 1, 0);
if(!iGrabRes)
{
       // ShowErrorMessage
}
```

AlfmlStopGrab

C Usage

```
int AlfmlStopGrab(imdev_t fgrab)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.

Description

Stops an acquisition in progress. Call this function only after starting a continuous acquisition. Once this function stops an acquisition, you can restart the acquisition with the AlfmlStartGrab() function.

Return Values:

On success, this function returns zero value. On failure, this function returns an error code. To get extended error information, call AlfmlGetLastError().

After receiving the stop command, the frame grabber generates an AL_GRAB_FINISHED event.

Example:

```
…
int iGrabRes;
iGrabRes = AlfmlStopGrab(fgrab);
if(!iGrabRes)
{
      // ShowErrorMessage
}
```

AlfmlGetImage

C Usage

```
image_t AlfmlGetImage(imdev_t fgrab, int imageIndex)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
imageIndex	The index of the image in the ring buffer.

Description

Returns an image from the ring acquisition buffer. Use this function to get access to the grabbed images.

Return Values:

On success, this function returns a variable of type image_t. On failure, this function returns NULL.

Example:

```
image_t img;
int iLast;

iLast = AlfmlGetLastImageIndex(fgrab);
img = AlfmlGetImage(fgrab, iLast);
```

AlfmlGetLastImageIndex

C Usage

```
int AlfmlGetLastImageIndex(imdev_t fgrab)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.

Description

Returns the index in the frames ring buffer of the last grabbed image.

Return Values:

Index of the last grabbed image.

Example:

```
image_t img;
int iLast;

iLast = AlfmlGetLastImageIndex(fgrab);
img = AlfmlGetImage(fgrab, iLast);
```

AlfmlGetFramesCount

C Usage

```
int AlfmlGetFramesCount(imdev_t fgrab)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.

Description

Returns the number of frames grabbed since the start of an acquisition.

AlfmlGetBufferSize

C Usage

```
int AlfmlGetBufferSize(imdev_t fgrab)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.

Description

`Returns the number of frames allocated at the ring buffer.`

AlfmlSetChannel

C Usage

```
int AlfmlSetChannel(imdev_t fgrab, int inputChannel)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
inputChannel	Selected input channel.

Description

Selects one of the video inputs to be active. The first channel is 0 and the last channel is according to the specified board been used. The default channel after calling AlfmlOpen() is channel 0. This API is only used by boards that can on capture from one channel at a time. When the channel is change it can take up to ½ a frame time for the horizontal phase locked loop on some boards to re-sync.

Return Values:

On success, this function returns zero value. On failure, this function returns an error code. To get extended error information, call AlfmlGetLastError().

AlfmlGetLastError

C Usage

```
int AlfmlGetLastError(imdev_t fgrab)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.

Description

Returns the error code of the last FastMotion Library function executed.

Return Values:

This function returns the last error code and 0 if there is no pending error.

AlfmlGetErrorStr

C Usage

```
Void AlfmlGetErrorStr(int errorCode, char* errStr)
```

Arguments

errorCode	A valid handle to the board previously allocated with AlfmlOpen() function.
errStr	A storage for error message text.

Description

This function returns the error text corresponding to an error code. The caller must allocate a minimum of 64 bytes for message storage before calling this function.

Return Values: none

AlfmlGetFlag

C Usage

```
Unsigned long AlfmlGetFlag (imdev_t fgrab, int cpu, int reg)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
Cpu	The zero based index of the cpu whose flags are requested.
Reg	The flag register desired

Description

This function returns the value contained in a flag register for the selected CPU. This is a diagnostic function and is not used by user applications. This function may be stubbed for some boards.

Return Values:

The return is the value in the flag register of the selected cpu.

AlfmlSendMessage

C Usage

```
Int AlfmlSendMessage (imdev_t fgrab, int cmd, int len, void *data)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
Cmd	A number defining the function of the data be sent.
Len	Length in bytes of the data pointed to by 'data'
Data	Pointer to the data part of the command

Description

This function sends a small block of data and a command to the frame grabber performing some predefined function determined by the 'cmd' value. This interface is used to extend the function of the API into board specific areas.

Return Values:

The return value is 0 if the message was successfully sent, otherwise non zero.

AlfmlGetdbuf

C Usage

```
Unsigned char AlfmlGetdbuf (imdev_t fgrab, int cpu, int index)
```

Arguments

Fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
Cpu	Zero based index used to select the cpu to get the dbuf data from
Index	The index of the byte to get from the dbuf
	Pointer to the data part of the command

Description

This function returns bytes from the debug buffer in the frame grabber, if it supports one..

Return Values:

The return value is the byte value from the debug buffer in the selected CPU memory. This API is not used by applications programs, it is used to debug frame grabber code.

D. EXTENSION COMMANDS FOR AFML SEND COMMAND

Introduction

The following enum defines all the commands that can be sent to the frame grabber: Some of these commands are applicable to particular boards (such as the Select AD9888 section).

```
typedef enum {
USER_CHANNEL = ALMSG_USERID,// 256 select the input channel
USER_UPDATE_TRIGGER,       // 257 set trigger settings
USER_FOCUS,                // 258 set the vga roi
USER_SNAP,                 // 259 get some images
USER_EXIT,                 // 260 kill the processor
USER_SNAP_STOP,            // 261 stop getting images
USER_ANNOTATE,             // 262 put counter on image
USER_SIMULATE,             // 263 make simulated images
USER_TRIGGER,              // 264 trigger now
USER_CAPTURE_TIMEOUT,      // 265 not getting images
USER_REPORT,               // 266 printf diagnostics
USER_REARM,                // 267 reset the trace and wr_rdy log
USER_CLOCKPHASE,           // 268 Set Sysad clock phase
USER_BAUD,                 // 269 Set Serial Baud rate
USER_PLL_DIVISOR,          // 270 Set the PLL Divisor
USER_PLL_PHASE,            // 271 Set the PLL Phase (Clock Phase)
USER_MUX_SET,              // 272 Set the input muxes
USER_PLL_CURRENT,          // 273 Set the pll current
USER_PLL_RANGE,            // 274 Set the pll phase
USER_HOST_DMA,             // 275 Enable/Disable DMA to Host Memory
USER_FEMEMCLOCKPHASE,      // 276 femem clock phase (Pll)
USER_SET_ROI,              // 277 change source and dest roi
USER_SELECT_AD9888_SECTION,// 278 change settings on AD9888
} e_message;
```

Extension commands

The extension commands are formatted with zero or more parameters. All the parameters are stored in an array of longs and sent with the commands. For example:

```
long arg[2];
arg[0] = AL_GRAB_ONE_SHOT;
arg[1] = 0;
AlfmlSendCommand (fgrab, USER_SNAP, 2*sizeof(long), (void *)arg);
```

This sends the user snap command with its two parameters. Remember that the Length parameter to the AlfmlSendCommand is in BYTES.

a) ***USER_EXIT:***

This command has no parameters. It causes the frame grabber program to exit. This command is not necessary to user applications as the function AlfmlClose does it for you.

b) ***USER_FOCUS:***

This command does nothing.

c) ***USER_SNAP:***

Parameters

ARG[0] = Capture mode

ARG[1] = Images per interrupt.

This command starts the capture of video and the delivery of images to host memory. The capture mode is one of:

AL_GRAB_ONE_SHOT

AL_GRAB_CONTINUOUS

One shot will deliver one image to host memory. In a multi-channel setting one shot still means a single image from one of the input channels. One shot command does not need to be stopped.

Continuous means capture will be on going with the deliver of images from all enabled channels as fast as possible. The continuous mode requires a USER_SNAP_STOP to stop capture. Note: This command is used by the AlfmlStartGrab function.

d) USER_SNAP_STOP:

This command has no parameters. The current image transfer is completed and the capture stops.

e) USER_UPDATE_TRIGGER:

Parameters

Arg[0] = number of images to grab after stop trigger.

Arg[1] = signal format (0=button,1=rising edge,2=falling edge)

This command updates the trigger settings in the application.

f) USER_SIMULATE:

Parameters

Arg[0] = Simulate settings

This command will turn on / off simulated images created by the frame grabber without a source connected. (nz=on zero=off).

g) USER_ANNOTATE:

Parameters

Arg[0] = Annotate (0=off nz=on)

This command will turn on / off annotation of the simulated images.

The annotation on the image is:

ZZ DDDDDDDDDD XX YYYYYYYYYYYY CCCC RRRR SSSS B

Z = Channel

D = counter

X = Buffer Index

Y = Buffer Address

C = Columns

R = Rows

S = Stride

B = Bytes per pixel.

h) ***USER_CHANNEL:***

This command selects a different channel for monitoring. This is not used by the Fast-X boards.

i) ***USER_REPORT:***

Parameters

Arg[0] = Report Selection

This command causes reports to be printed in the diagnostic window. This is for diagnostic use, typically not needed by the user (except #3).

```
0 = All, prints all the reports
1 = Sizes prints the image size registers in the fpga
2 = Trace prints the content of the logic trace buffer
3 = 9888 prints the settings of the 9888s
4 = FPGA prints the content of the FPGA registers
5 = Ticks prints table of times between varrious events.
```

j) ***USER_REARM:***

This command resets the logic trace system to capture a new logic trace. This is not use full to the end user, it contains signals selected by the hardware engineer.

k) ***USER_CLOCKPHASE:***

This API is only useful to the hardware engineer. Don't use this.

l) ***USER_FEMEMCLOCKPHASE:***

This API is only useful to the hardware engineer. Don't use this.

m) ***USER_BAUD:***

This API sets the baud rate used by the Camera Link interface.

n) ***USER_PLL_DIVISOR:***

Parameters

Arg[0] = Channel (which 9888).

Arg[1] = value to program into PLL divisor

This API allows direct adjustment of the PLL divisor used by the 9888s.

o) <u>USER_PLL_PHASE:</u>

Parameters

Arg[0] = Channel (which 9888).

Arg[1] = value to program into PLL phase

This API allows direct adjustment of the PLL phase used by the 9888s.

p) <u>USER_PLL_RANGE:</u>

Parameters

Arg[0] = Channel (which 9888).

Arg[1] = value to program into PLL range

This API allows direct adjustment of the PLL range used by the 9888s.

q) <u>USER_PLL_CURRENT:</u>

Parameters

Arg[0] = Channel (which 9888).

Arg[1] = value to program into PLL current

This API allows direct adjustment of the PLL current used by the 9888s.

r) <u>USER_MUX_SET:</u>

Parameters

Arg[0] = Channel (which 9888).

Arg[1] = 0=Subchannel0 1=subchannel1 2=off

This API allows direct adjustment of the input multiplexers on the 9888s.

s) <u>USER_SET_ROI:</u>

Parameters

arg[0] = channel	(which 9888, 0 or 1)
arg[1] = subchannel	(which mux setting 0 or 1)
arg[2] = modify hardware roi	(1=Update HDW ROI, 0=Don't)
arg[3] = new hardware left	
arg[4] = new hardware right;	
arg[5] = new hardware top;	
arg[6] = new hardware bottom;	
arg[7] = new hardware bpp;	
arg[8] = new hardware st;	
arg[9] = modify output roi	(1=Update Out ROI, 0=don't)
arg[10] = new output left;	
arg[11] = new output right;	
arg[12] = new output top;	
arg[13] = new output bottom;	
arg[14] = new output bpp;	
arg[15] = new output st;	

This command controls the resizing logic.

The hardware ROI settings are programmed into the FPGA and determine which pixels are captured from the input raster.

The output ROI determines the final size of the image, after it is resized from the ROI determined by the hardware ROI. This ROI is clipped by the original host image.

The range of these values are determined by the FPGA settings in the capture file. The entered ROI is clipped by the original ROI settings, to determine the actual ROI used. An ROI is defined by six parameters, left, right, top and bottom determine the size of the image. The stride ('st') is the number of bytes between the start of the first line to the start of the second line. It determines the storage format of the image. 'bpp' sets the number of bytes per pixel.

Arbitrary vertical and horizontal scale factors are supported. The ROIs are turned on between images, for the hardware ROI that is between images captured. For the output ROI that is between DMAs to the host. When changing an ROI you me see an intermediate image which is part way changed.

The size of the image is reflected in the image_t returned by GetImage.

t) ***USER_SELECT_AD9888_SECTION:***

This command does not use a table of longs. Instead it uses the following structure:

```
struct
{
       unsigned long channel;
       char SectionName [64];
}
```

The first parameter selects which 9888 is to be modified (0 or 1). The section name parameter selects which section from the cap file should be loaded to the 9888. When the program boots, the section [AD9888B.1] is loaded into 9888 zero, and the section [AD9888B.2] is loaded into 9888 one.

The section name is whatever you want in the cap file. CASE is important. Do not use spaces, otherwise the name can be up to 63 characters long.

On boot when the capture file is read by the board, it is stored in memory as a tree structure. The cap file remains in memory accessible to this command from the host.

GENERAL INFORMATION

E. TROUBLESHOOTING

There are several things you can try before you call Alacron Technical Support for help.

_____ Make sure the computer is plugged in. Make sure the power source is on.

_____ Go back over the hardware installation to make sure that the system is properly installed.

_____ Go back over the software installation to make sure you have installed all necessary software.

_____ Run the Installation User Test to verify correct installation of both hardware and software.

_____ Run the user-diagnostics test for your main board to make sure it's working properly.

_____ Insert the Alacron CD-ROM and check the various Release Notes to see if there is any information relevant to the problem you are experiencing.

The release notes are available in the directory: **/usr/alacron/alinfo**

F. ALACRON TECHNICAL SUPPORT

Alacron offers technical support to any licensed user during the normal business hours of 9 a.m. to 5 p.m. EST. We offer assistance on all aspects of processor board and PMC installation and operation.

Contacting Technical Support

To speak with a Technical Support Representative on the telephone, call the number below and ask for Technical Support:

Telephone: **603-891-2750**

If you would rather FAX a written description of the problem, make sure you address the FAX to Technical Support and send it to:

Fax: **603-891-2745**

You can email a description of the problem to ***support@alacron.com***

Before you contact technical support have the following information ready:

_____ Serial numbers and hardware revision numbers of all of your boards. This information is written on the invoice that was shipped with your products.

_____ Also, each board has its serial number and revision number written on either in ink or in bar-code form.

_____ The version of FASTMOTIONLIB software that you are using.

_____ You can find this information in a file in the directory: **/usr/alfast/alinfo**

_____ The type and version of the host operating system, i.e., Windows XP.

_____ Note the types and numbers of all your software revisions, daughter card libraries, the application library and the compiler

_____ The piece of code that exhibits the problem, if applicable. If you email Alacron the piece of code, our Technical-Support team can try to reproduce the error. It is necessary, though, for all the information listed above to be included, so Technical Support can duplicate your hardware and system environment.

Returning Products for Repair or Replacements

Our first concern is that you be pleased with your Alacron products.

If, after trying everything you can do yourself, and after contacting Alacron Technical Support, you feel your hardware or software is not functioning properly, you can return the product to Alacron for service or replacement. Service or replacement may be covered by your warranty, depending upon your warranty. The first step is to call Alacron and request a “Return Materials Authorization” (RMA) number. This is the number assigned both to your returning product and to all records of your communications with Technical Support. When an Alacron technician receives your returned hardware or software he will match its RMA number to the on-file information you have given us, so he can solve the problem you’ve cited.

When calling for an RMA number, please have the following information ready:

_____ Serial numbers and descriptions of product(s) being shipped back

_____ A listing including revision numbers for all software, libraries, applications, daughter cards, etc.

_____ A clear and detailed description of the problem and when it occurs

_____ Exact code that will cause the failure

_____ A description of any environmental condition that can cause the problem

All of this information will be logged into the RMA report so it’s there for the technician when your product arrives at Alacron.Put boards inside their anti-static protective bags. Then pack the product(s) securely in the original shipping materials, if possible, and ship to:

Alacron Inc.
71 Spit Brook Road, Suite 200
Nashua, NH 03060
USA

Clearly mark ***the outside of your package:***

Attention **RMA #80XXX**

Remember to include your return address and the name and number of the person who should be contacted if we have questions.

Reporting Bugs

We at Alacron are continually improving our products to ensure the success of your projects. In addition to ongoing improvements, every Alacron product is put through extensive and varied testing. Even so, occasionally situations can come up in the fields that were not encountered during our testing at Alacron.

If you encounter a software or hardware problem or anomaly, please contact us immediately for assistance. If a fix is not available right away, often we can devise a work-around that allows you to move forward with your project while we continue to work on the problem you've encountered.

It is important that we are able to reproduce your error in an isolated test case. You can help if you create a stand-alone code module that is isolated from your application and yet clearly demonstrates the anomaly or flaw.

Describe the error that occurs with the particular code module and email the file to us at:

support@alacron.com

We will compile and run the module to track down the anomaly you've found.

If you do not have Internet access, or if it is inconvenient for you to get to access, copy the code to a disk, describe the error, and mail the disk to Technical Support at the Alacron address below.

If the code is small enough, you can also:

FAX the code module to us at 603-891-2745

If you are faxing the code, write everything large and legibly and remember to include your description of the error.

When you are describing a software problem, include revision numbers of all associated software.

For documentation errors, photocopy the passages in question, mark on the page the number and title of the manual, and either FAX or mail the photocopy to Alacron.

Remember to include the name and telephone number of the person we should contact if we have questions.

Alacron Inc.
71 Spit Brook Road, Suite 200
Nashua, NH 03060
USA

Telephone: 603-891-2750
FAX: 603-891-2745

Web site:
http://www.alacron.com/

Electronic Mail:
sales@alacron.com
support@alacron.com