ALACRON

FASTMOTION STRETCH LIBRARY

FASTMOTION STRETCH LIBRARY
USER'S MANUAL

COPYRIGHT NOTICE

Copyright © 200-2007 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	April 10, 2007

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

OTHER ALACRON MANUALS

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
30002-00150	FastSeries Library User's Manual
30002-00153	Fast I/O Hardware User's Manual
30002-00155	FastMem Hardware User's Manual
30002-00162	FOIL – **F**astSeries **O**bject **I**maging **L**ibrary User's Manual
30002-00169	ALRT Runtime Software Programmer's Guide & Reference
30002-00170	ALRT, ALFAST & FASTLIB Software Installation Manual for Linux
30002-00171	ALRT, ALFAST, & FASTLIB Software Installation for Windows NT
30002-00173	FastMem Programmer's Guide & Reference
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
30002-00185	FastVision Hardware User's Manual
30002-00186	FastVision Software User's Manual
30002-00187	FastFrame 1300 Hardware User's Manual

1. INTRODUCTION

This manual covers the Alacron **FASTMOTION LIBRARY** software. The FASTMOTION LIBRARY enables the software developer to embed FASTMOTION functionality into their application. The FASTMOTION LIBRARY and Application use the OCX model for windows development as diagrammed below.

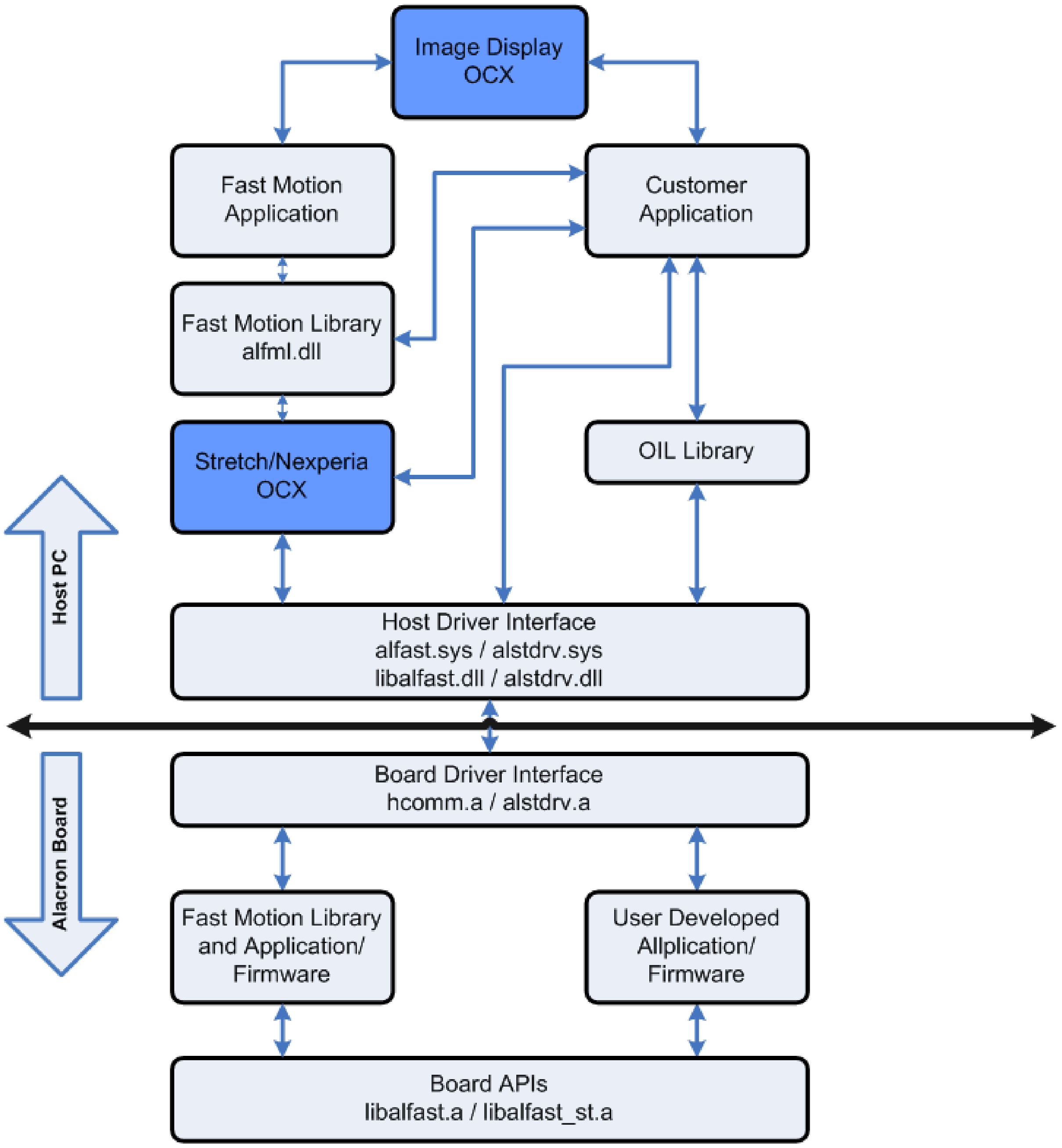


Image Display OCX
Fast Motion Application
Customer Application
Fast Motion Library alfml.dll
Stretch/Nexperia OCX
OIL Library
Host PC
Host Driver Interface alfast.sys / alstdrv.sys libalfast.dll / alstdrv.dll
Alacron Board
Board Driver Interface hcomm.a / alstdrv.a
Fast Motion Library and Application/ Firmware
User Developed Allplication/ Firmware
Board APIs libalfast.a / libalfast_st.a

1.1 Purpose

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

1.2 Audience

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

1.3 FASTMOTION STRETCH Library

The Alacron FastMotion Library for the FastSeries 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 application under Microsoft Windows XP and Linux.

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.

2. LIBRARY VERSIONS

The FastMotion Library is distributed with a dynamically linked library for the host computer and with a set of board programs 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 the Linux environment FastMotion Library is a staticly linked library alfml.a.

1.4 Include File

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

1.5 Data Types- Matrices and Images

A matrix or image is a two-dimensional array of values defined by a pointer (or array address), a vertical stride, the number of rows and the number of columns. These values may either be real or complex. The vertical stride defines the address increment from one row of the array to the next, and allows the referencing of “sub-arrays” of the image. The FastMotion Library uses row-major order. Successive locations in memory contain successive elements of a row, until the end of the row, which is followed by the first element of the next row. If a program desires column-major order, swap the row and column input arguments to achieve the desired result.

1.5.1 List of Functions

The FastMotion Library contains the following functions:

AlfmlOpen

AlfmlClose

AlfmlSetCallBack

AlfmlStartGrab

AlfmlStopGrab

AlfmlGetImage

AlfmlGetLastImageIndex

AlfmlGetBufferSize

AlfmlGetFramesCount

AlfmlSetChannel

AlfmlGetLastError

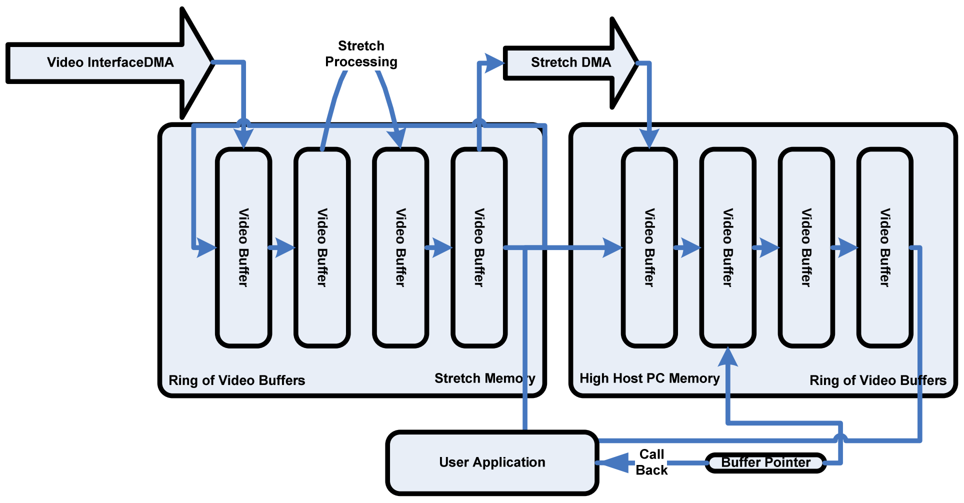
AlfmlGetErrorStr

The functions above are present in all versions of the FastMotion Library. Each specific product configuration can have additional interfaces.

1.6 FastMotion 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. The functions that are part of the FastMotion Library are executed on the PC. The FastMotion Library supports users who do not want to develop software for the Stretch processor on the Fast-X/e boards. The FastMotion Library block diagram is show below.



The data flows by successive DMA operations. Video data from the video source is DMAed into Stretch memory, in to a ring of buffers. At the same time the Stretch processor can be performing processing on a buffer to an output buffer. Again at the same time the Stretch DMA controller is transferring a completed buffer to the host memory. The FastMotion Library traps an event from the Stretch processor 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. Assuming all the DMAs and processors 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 empted by the transfer or DMA operation.

The buffer which is passed to the callback function is locked as long as the user application does not return from the callback. When the callback returns the FastMotion Library unlocks the buffer making it available for new data. The Depth of the ring buffers is settable up to the limit of available memory.

1.7 Memory Use

The FastMotion Library uses memory blocked from use by the host operating system. This is done by telling the OS it can't use a block of memory, at the top of physical memory. On Windows this is done by modifying the file c:\boot.ini which is a hidden system file. On Linux this is done with a boot parameter to the loader.

The size of this memory block is determined by a user settable parameter called 'DMABUF_LENGTH'.

Note: The FastMotion Library is not the only program that uses this method of obtaining some contiguous memory to do DMAs with. Several companies have used this method. Please be sure to check that the block of memory is not being used by some other program.

1.8 API Functions of the FastMotion Library

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 (zero). 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 is recieved. 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)`

The reason variable can be one of

```
AL_IMAGE_READY
AL_GRAB_FINISHED
AL_STDOUT
AL_GRAB_TIMEOUT
```

If the reason is AL_IMAGE_READY, then 'imageIndex' is the index of the just filled buffer in memory. Use AfmlGetImage to obtain an image pointer.

If the reason is AL_GRAB_FINISHED, then the cause of the call back was the completion of a grab operation.

If the callback is AL_STDOUT, the userData pointer points to a string in memory, which was printed to stdout by the frame grabber. These callbacks can contain usefull diagnostics, but need not be processed for the library to operate correctly.

If the reason is AL_GRAB_TIMEOUT image data was not received from the hardware within the timeout interval (10 seconds).

Example:

```
int MyCallBack(int reason, void* userPtr, int index)
{
       if(reason == AL_IMAGE_READY)
       {
        // new image grabbed
       }
       else if(reason == AL_GRAB_FINISHED)
       {
        // The buffer is full
       }
}

alfmlSetCollBack(fgrab, MyCallBack, userPtr);
```

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, at the library header file:

AL_GRAB_CONTINUOUS - The board grab 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 images continuously and stops when the frame buffer is full. 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. 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.

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

```
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:

3. FAST-X UXGA FUNCTIONS

The Fast-X with the UXGA option has two additional functions in the AFML interface as well as several different functions supported by the firmware loaded by the FastMotion Library.

1.9 API Functions of the FastMotion UXGA Option Library

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 CPU instance number in a multi-CPU application. Set this parameter to zero in most cases.
Reg	A storage for error message text.

Description

This function returns the contents of the 'Flag' register present in the UXGA version of the firmware. The registers contain diagnostic information about the internal functions of the firmware. There are 10 registers numbered 0 to 9. Providing an incorrect register number will return 0xBADDF00D, These Flag registers can change with revision of the software, so do not depend on the retaining any useful content.

Return Values:

This function returns the contents of the 'Flag' register present in the UXGA version of the firmware.

The registers contain:

Reg #	Contents
0	The write index of the shared memory area in host memory
1	The read index of the shared memory area in host memory
2	The write index of the internal buffer ring, which was last filled by the hardware.
3	The read index of the internal buffer ring, which was last started to DMA to the host shared memory area.
4	Time in microseconds sense the last interrupt from the hardware transferring an image into the board memory.
5	Time from the last interrupt to when the deferred procedure call completed processing the buffer for channel 0
6	Time from the last interrupt to when the deferred procedure call completed processing the buffer for channel 1
7	Last line number in the source code being executed.
8	Physical address of the shared memory area in host memory.
9	The offset from the base of SRAM area in the on board processor, to the control structure.

AlfmlSendCommand

C Usage

```
int AlfmlGetFlag (imdev_t fgrab, int cmd, int len, void *d)
```

Arguments

fgrab	A valid handle to the board previously allocated with AlfmlOpen() function.
cmd	Enum value defining the command being sent to the board.
len	Length of parameter data pointed to by the void pointer 'd'
d	Void pointer to parameter data.

Description

This function is provided for informational /diagnostic reasons, but it is an internal API not intended to be used by user applications. This function sends a command message to the processor on the target board (or boards) as determined by the enum value of 'cmd'. The command can also have parameter data which is provided for by the remaining arguments. The functionality of the commands are subject to change as this is an internal function.

Return Values:

This function returns zero if the message was sent successfully, non-zero if it was not sent successfully.

1.10 Command Summary

The commands and parameters in the UXGA firmware are:

Cmd #	Contents
256	**Select a channel. (0 or 1)**
257	**Update trigger settings. Not used by the UXGA option.**
258	**Select production of a focus image. Not used by the UXGA option.**
259	**Enable capturing. The following are the 'long' parameters:** **Grab mode (1=one shot 2=continuous)** **ImagesPerInterrupt (usually set to 0, or one image per evt)** **Skip (usually 0, give every image, 1 give every other image)**

260	Cause the firmware to exit. Crash the program.
261	Stop Capture. Parameter: When (0=now, nz=when post trigger count expires)
262	Draw digits of two counters on image. Parameter Stop (0=stop 1=do it) The two numbers are a time in microseconds, and a frame index. Only on simulated image data.
263	Make simulated images. Stop (0=stop 1=do it)
264	Trigger now. Stop capture after post trigger count expires.
265	Sent by firmware when images are not coming into the board.
266	Diagnostic message sent by board.
267	Re-arm the trace buffer in the FPGA
268	Adjusts the clock phase on the DMA channel from the FPGA to the Stretch. Parameter -8 to +8. Should be -4. Don't mess with this.
269	Set the serial baud rate. Parameter selects the channel (0 or 1)

270	Adjust the PLL divisor of the AD9888s
	P1=Channel (0=ch0 1=ch1 2 -1=both)
	P2=divisor (minus 1) see the AD9888 data sheet

271	Adjust the clock phase AD9888s on the UXGA board.
	P1=Channel (0=ch0 1=ch1 2 -1=both)
	P2=phase range is 0 to 31.

272	Change the input MUX setting for the selected channel.
	P1=Channel (0=ch0 1=ch1 2 -1=both)
	P2=setting (0=A 1=B 2=Off)

273	Set the PLL current of AD9888s P1=Channel (0=ch0 1=ch1 2 -1=both)

	P2=setting

274	Set the PLL range of AD9888s
	P1=Channel (0=ch0 1=ch1 2 -1=both)
	P2=setting

4. UXGA CAPTURE CONTROL FILE CONTENT

The operation of the UXGA interface and the firmware has many parameters which are stored in an editable file. This file is called a capture control file '.cap'. The format of the file is much like a Windows INI file. A modification to the format is the inclusion of data type flags before the actual value of the parameter.

```
# ------------------------------------------
# Example of a small Cap control file
# -------------------------------------------
[Stretch]                          # Section name
np = int 1                         # Number of processors
select0 = int 1                    # use this processor (0)
prog0 = string "uxgaFastX.exe"     # file to load

[PLL]                              # next section
PllPhase = int -4
```

The format is a section header between two square brackets '[' or ']'. The section name should be less than 64 characters, the case is important. After a section header is a number of lines containing parameters and values. The format of a parameter line is the parameter name which is limited to 64 characters, followed by an equal size ('='), followed by one of the type flags 'int', 'float', or 'string'. Followed by the value, which must be less than 64 characters.

The capture file format also supports include files. The parameter '!include' by the file name, will cause the file to be read and processed. The number of levels of include supported is large, limited by the size of the stack.

The capture file format also supports conditional processing delimiters, '!if', '!else' and '!endif'. The token following the '!if' can be '0' or '1'.

Finally a comment token '#' provided. Characters from the '#' to the end of the line are ignored. DO NOT use the // as a comment character it causes big problems.

The capture file constructs a tree in memory with the root pointing to a linked list of section headers, which point to linked lists of parameters with their values. This structure is queried by the host software (alfml.dll) and by the board firmware. Some section headers may not apply to a particular board type. (Yes this was created before XML…)

Capture control files use the extent '.cap' but this is not required, they can have any file name.

1.11 [Stretch] Section

This section informs the FastMotion Library which processor to load with what firmware. Firmware is matched to the options on the board. Typically this is not edited by a user.

1.11.1 np = int 1

Number of processors in application. Normally this is one.

1.11.2 select0 = int 1

This parameter selects whether a processor is included in the set controlled by FastMotion Library. The digit at the end of the parameter selects which processor, the value selects if this processor is to be used.

1.11.3 prog0 = string "uxgaFastX.exe"

The parameter selects which firmware to load to the board. Firmware is matched to the options on the board. Typically this is not edited by a user.

1.11.4 [FastMotion] Section

This section is descriptive and does not effect the operation of the hardware or software. This section is used by Fast Motion Application and is not used by the FastMotion Library.

1.11.5 CameraName = string "Fx-sxga-60"

Camera model name.

1.11.6 frame0 = string "1280,1024,60"

The width, height, and frame rate of the camera.

1.11.7 line0 = string "1280x1024x24@60"

This is displayed in the selection dialog.

1.11.8 line1 = string "Fast-X"

This is displayed in the selection dialog.

1.11.9 DISPLAY_SHIFT = int 0

This is used to shift the pixel values before they are displayed. Positive numbers mean left shift, negative numbers mean right shift. This is used by the application to select which bits are used to display an image.

1.11.10 SIMULATE = int 0

This parameter controls various simulation functions in the various firmware. Typical it causes the firmware to generate images without a camera, to test the setup.

1.11.11 ANNOTATE = int 0

This parameter controls if text is written to the image with various types of information. Typically the firmware adds a frame number and a time.

1.11.12 INPUT_CHANNELS = int 2

This parameter tells the Fast Motion Application how many channels are present in the current configuration.

1.12 [PLL] Section

1.12.1 PLL_CONTROL = int -4

This parameter tunes the clock phase on the 200 MHz data channel between the FPGA and the Stretch processor. Do not change this.

1.13 [BUFFER.1] Section

The Ping Pong settings should be used in applications which require images from more than one channel or mux setting. Ping Pong reduces the bandwidth to that of a single channel by alternating channels which are sampled. The alternation is done in a way that the time it takes for the AD9888 PLL to re-sync is masked by the other channel. To take images from all the inputs set Ping Pong Mux and Channel.

NOTE: IF A CHANNEL DOES NOT HAVE INPUT THE PING PONG WILL STOP ON THAT CHANNEL WAITING FOR A FRAME. BE SURE ALL THE SELECTABLE CHANNELS HAVE VIDEO INPUT.

1.13.1 PING_PONG_CHANNEL = int 0

This parameter informs the UXGA firmware to switch back a forth between channel 1 and 2. This is used for monitoring applications to get data from many channels as fast as possible.

1.13.2 PING_PONG_MUX = int 0

This parameter informs the UXGA firmware to switch through the input multiplexer channels for channel 0.

1.14 [BUFFER.2] Section

1.14.1 PING_PONG_MUX = int 0

This parameter informs the UXGA firmware to switch through the input multiplexer channels for channel 2.

1.15 [AFML] Section

This section defines the size and kind of image the host would like to obtain from the board(s). In many applications this is not free to be changed. The format of the image must be RGB24 for the UXGA option. The size of the image can change to match the size of the video source.

1.15.1 HOST_INPUT_PIXELS_PER_LINE = int 1280

1.15.2 HOST_INPUT_LINES_PER_BUFFER = int 1024

1.15.3 HOST_INPUT_TYPE = string RGB24

1.16 [UXGA.1] and [UXGA.2] Sections

This section controls the registers in the FPGA. The number at the end of the section name indicates the channel that is affected by this section.

1.16.1 ENABLE = int 1

The enable bit control whether the channel will be used in this application.

1.16.2 START_PIXEL = int 358
WIDTH = int 1280
START_LINE = int 20
END_LINE = int 1043

These four parameters determine the ROI to be extracted from the input video. The value for the start pixel is in clock after the leading edge of horizontal sync. The start line is the first line in the image which has useful video. The end line parameter determines the last line that will be input from the image. The start and end parameters include the line or pixel they select.

1.16.3 HS_ACTIVE_HI = int 1
VS_ACTIVE_LO = int 0
AUTO_POLARITY = int 1

These values control the polarity of the sync signals. If auto polarity is set the other two parameters are ignored. Typically you should use auto polarity. Should auto polarity not perform as desired, the set it to zero, and select the polarity with the other two parameters. Note that the parameters name what the value 1 will select. So HS_ACTIVE_HI if one selects an active high HSYNC. Note that the two parameters HS_ACTIVE_HI and VS_ACTIVE_LO have opposite sense. For example if both syncs are active low then HS_ACTIVE_HI is zero and VS_ACTIVE_HI is one.

1.16.4 INTERLACE = int 0

The interlace bit informs the FPGA that the input video is 2:1 interlaced. Note the ROI settings DO NOT change if the input is interlaced. The FPGA takes care of the even odd line counting.

1.16.5 CAPTURE_ENABLE = int 1

This bit enables video to be captured by the FPGA. Setting this to zero will prevent video from being captured on the selected channel, but the channel will be setup and ready to do it if the capture bit is changed.

1.16.6 MONOCHROME = int 0

The monochrome bit causes the FPGA to only use the green input. The red and blue inputs will be ignored. The image in memory will be a GREY8 image rather than a RGB24 image.

1.16.7 USEBUFFER1 = int 1
USEBUFFER2 = int 1

These two bits control the buffer management. Both of these should be set to one.

1.16.8 IGNORE_STRIDE = int 0

This bit causes the DMA engine to do a linear DMA and not step to the next line start, at the end of a line. This should be zero.

1.17 [Capture.1] and [Capture.2] Sections

These two sections define the input image to the firmware running on the board. These values are used to program the hardware, and define what is to be expected in memory when the hardware finishes a DMA.

1.17.1 NUMBER_OF_BUFFERS = int 4

This defines the number of buffers to be used in the input circular chain of buffers. This is typically not changed.

1.17.2 INPUT_PIXELS_PER_LINE = int 1280

As the name says.

1.17.3 INPUT_LINES_PER_BUFFER = int 1024

This should be set to greater than (+1 or more) the number of lines in the image. If the value is too small a partial image will be captured, with the balance spilling into the next buffer.

1.17.4 INPUT_XOFFSET = int 0
INPUT_YOFFSET = int 0

The value determines which line and pixel is the line that has useful image data. These are not the same values as those in the UXGA section, but apply to the image as provided by the hardware. This allows you to select a smaller area of the image to be transferred to the host.

1.17.5 INPUT_NX = int 1280
INPUT_NY = int 1024

This value determines the width and height of the ROI being extracted from the image in memory. Typically these match the ROI settings of the UXGA section.

1.17.6 INPUT_PIXEL_SIZE = int 3

This is the number of bytes in the pixel. It should be 3 or in the monochrome case it should be 1.

1.17.7 INPUT_PIXEL_TYPE = string RGB24

This field determines the format of the pixel. RGB24 means LSB is red, middle byte is green and the MSB is blue. This field is GREY8 for the monochrome case.

1.18 [AD9888B.1 and [AD9888B.2] Sections

The fields in this section define the register values to be programmed in the AD9888. One set of values for each AD9888. It is beyond the scope of this document to explain all these settings please refer to the Analog Devices data sheet for the AD9888B. These settings are important to the quality of the resulting images, and will required adjustment for different video sources. Typically you should only need to change the first four values to match your video source. For better color matches adjust the red, green and blue gains and offsets.

These values are example values taken from a 1280x1024 @ 60 Hz Toshiba laptop setting.

```
PLLDIV = int 1686
Range = int 2
Current = int 4
Phase = int 16
ClampLoc = int 16
ClampDur = int 64
HsyncWidth = int 112
RedGain = int 128
GreenGain = int 128
BlueGain = int 128
RedOffset  = int 128
GreenOffset = int 128
BlueOffset = int 128
VsyncOutSource = int 0
VsyncOverride = int 0
VsyncOutInvert = int 0
HsyncInSource = int 0
HsyncOverride = int 0
HsyncOutInvert = int 0
HsyncInInvert = int 0
HsyncPolOverride = int 0
PowerUp = int 1
PowerMode = int 0
CoastPolarity = int 0
CoastOverride = int 0
Coast Select = int 1
Clamp Polarity = int 0
Clamp Source = int 0
MustBeOne = int 1
BlueClamp = int 0
RedClamp = int 0
SyncOnGreenThreshold = int 15
SyncSeparatorThreshold = int 32
PreCoast = int 0
PostCoast = int 0
ExternalClock = int 0
InputBandwidth = int 3
InputMux = int 0
Format422 = int 0
ABInvert = int 0
OutputMode = int 1
ChannelMode = int 1
MustBeZero = int 0
MustBeOnes = int -1
```

5. THE AD9888 REGISTERS

The AD9888 is initialized and controlled by a set of registers that determine the operating modes. An external controller is employed to write and read the Control Registers through the 2-line serial interface port (I squared C). The information in this section is abstracted from the Analog Devices data sheet. You should obtain a copy of this data sheet. Search for AD9888 on Google.

Table V. Control Register Map

Hex Address	Read and Write or Read Only	Bits	Default Value	Register Name	Function
00H	RO	7:0		Chip Revision	An 8-bit register that represents the silicon revision level. Revision 0 = 0000 0000.
01H	R/$\overline{W}$	7:0	01101001	PLL Div MSB	This register is for Bits [11:4] of the PLL divider. Larger values mean the PLL operates at a faster rate. This register should be loaded first whenever a change is needed. (This will give the PLL more time to lock.)*
02H	R/$\overline{W}$	7:4	1101****	PLL Div LSB	Bits [7:4] LSBs of the PLL divider word.*
03H	R/$\overline{W}$	7:2	01******	VCO/CPMP	Bits [7:6] VCO Range. Selects VCO frequency range. (See PLL description.)
			001*		Bits [5:3] Charge Pump Current. Varies the current that drives the low-pass filter. (See PLL description.)
04H	R/$\overline{W}$	7:3	10000***	Phase Adjust	ADC Clock phase adjustment. Larger values mean more delay. (1 LSB = T/32)
05H	R/$\overline{W}$	7:0	00001000	Clamp Placement	Places the Clamp signal an integer number of clock periods after the trailing edge of the Hsync signal.
06H	R/$\overline{W}$	7:0	00010100	Clamp Duration	Number of clock periods that the Clamp signal is actively clamping.
07H	R/$\overline{W}$	7:0	00100000	Hsync Output Pulsewidth	Sets the number of pixel clocks that HSOUT will remain active.
08H	R/$\overline{W}$	7:0	10000000	Red Gain	Controls ADC input range (contrast) of each respective channel. Big ger values give less contrast.
09H	R/$\overline{W}$	7:0	10000000	Green Gain	
0AH	R/$\overline{W}$	7:0	10000000	Blue Gain	
0BH	R/$\overline{W}$	7:1	1000000*	Red Offset	Controls dc offset (brightness) of each respective channel. Bigger values decrease brightness.
0CH	R/$\overline{W}$	7:1	1000000*	Green Offset	
0DH	R/$\overline{W}$	7:1	1000000*	Blue Offset	

Table V. Control Register Map (continued)

Read and Hex Address	Write or Read Only	Bits	Default Value	Register Name	Function
0EH	R/$\overline{W}$	7:0	0*******	Sync Control	Bit 7—Hsync Polarity Override. (Logic 0 = Polarity determined by chip, Logic 1 = Polarity set by Bit 6 in Register 0EH.)
			*1******		Bit 6—Hsync Input Polarity. Indicates to the PLL the polarity of the in coming Hsync signal. (Logic 0 = active low, Logic 1 = active high.)
			0***		Bit 5—Hsync Output Polarity. (Logic 0 = Logic High Sync, Logic 1 = Logic Low Sync).
			0*		Bit 4—Active Hsync Override. If set to Logic 1, the user can select the Hsync to be used via Bit 3. If set to Logic 0, the active interface is selected via Bit 6 in Register 14H.
			****0***		Bit 3—Active Hsync Select. Logic 0 selects Hsync as the active sync. Logic 1 selects Sync-on-Green as the active sync. Note: the indicated Hsync will be used only if Bit 4 is set to Logic 1 or if both syncs are active (Bits 1, 7 = Logic 1 in register 14H).
			*****0**		Bit 2—Vsync Output Invert. (Logic 0 = Invert, Logic 1 = No Invert.)
			******0*		Bit 1—Active Vsync Override. If set to Logic 1, the user can select the Vsync to be used via Bit 0. If set to Logic 0, the active interface is selected via Bit 3 in Register 14H.
			*******0		Bit 0—Active Vsync Select. Logic 0 selects Raw Vsync as the output Vsync. Logic 1 selects Sync Separated Vsync as the output Vsync. Note: The indicated Vsync will be used only if Bit 1 is set to Logic 1.
0FH	R/$\overline{W}$	7:1	0*******		Bit 7—Clamp Function. Chooses between Hsync for Clamp signal or another external signal to be used for clamping. (Logic 0 = Hsync, Logic 1 = Clamp.)
			*1******		Bit 6—Clamp Polarity. Valid only with external Clamp signal. (Logic 0 = active high, Logic 1 selects active low.)
			0***		Bit 5—COAST select. Logic 0 selects the coast input pin to be used for the PLL coast. Logic 1 selects Vsync to be used for the PLL coast.
			0*		Bit 4—COAST Polarity Override. (Logic 0 = Polarity determined by chip, Logic 1 = Polarity set by Bit 3 in Register 0FH.)
			****1***		Bit 3—COAST Polarity. Changes polarity of external COAST signal. (Logic = 0 = active low, Logic 1 = active high.)
			*****1**		Bit 2—Seek Mode Override. (Logic 1 = allow low-power mode, Logic 0 = disallow low power mode.)
			******1*		Bit 1—$\overline{\text{PWRDN}}$. Full Chip Power-Down, active low. (Logic 0 = Full Chip Power-Down, Logic 1 = normal.)
10H	R/$\overline{W}$	7:3	01111***	Sync-on-Green Threshold	Sync-on-Green Threshold — Sets the voltage level of the Sync-on-Green slicer's comparator.
			*****0**		Bit 2—Red Clamp Select – Logic 0 selects clamp to ground. Logic 1 selects clamp to midscale (voltage at Pin 9).
			******0*		Bit 1—Blue Clamp Select – Logic 0 selects clamp to ground. Logic 1 selects clamp to midscale (voltage at Pin 24).
			*******0		Bit 0—Must be set to 1 for proper operation.
11H	R/$\overline{W}$	7:0	00100000	Sync Separator Threshold	Sync Separator Threshold – Sets how many internal 5 MHz clock periods the sync separator will count to before toggling high or low. This should be set to some number greater than the maximum Hsync or equalization pulsewidth.
12H	R/$\overline{W}$	7:0	00000000	Pre-COAST	Pre-COAST – Sets the number of Hsync periods that coast becomes active prior to Vsync.
13H	R/$\overline{W}$	7:0	00000000	Post-COAST	Post-COAST – Sets the number of Hsync periods that coast stays active following Vsync.

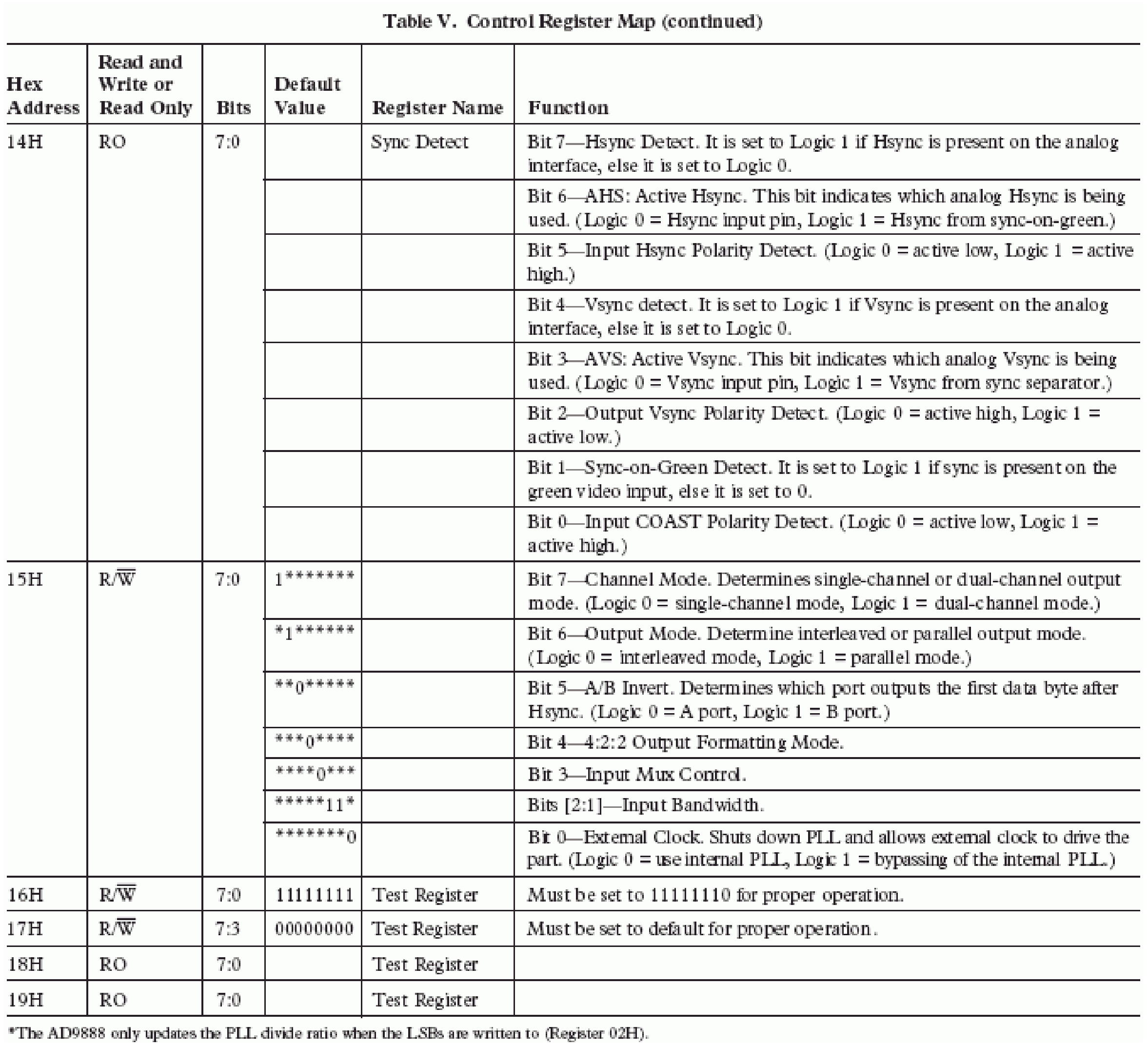


Table V. Control Register Map (continued)

Hex Address	Read and Write or Read Only	Bits	Default Value	Register Name	Function
14H	RO	7:0		Sync Detect	Bit 7—Hsync Detect. It is set to Logic 1 if Hsync is present on the analog interface, else it is set to Logic 0.
					Bit 6—AHS: Active Hsync. This bit indicates which analog Hsync is being used. (Logic 0 = Hsync input pin, Logic 1 = Hsync from sync-on-green.)
					Bit 5—Input Hsync Polarity Detect. (Logic 0 = active low, Logic 1 = active high.)
					Bit 4—Vsync detect. It is set to Logic 1 if Vsync is present on the analog interface, else it is set to Logic 0.
					Bit 3—AVS: Active Vsync. This bit indicates which analog Vsync is being used. (Logic 0 = Vsync input pin, Logic 1 = Vsync from sync separator.)
					Bit 2—Output Vsync Polarity Detect. (Logic 0 = active high, Logic 1 = active low.)
					Bit 1—Sync-on-Green Detect. It is set to Logic 1 if sync is present on the green video input, else it is set to 0.
					Bit 0—Input COAST Polarity Detect. (Logic 0 = active low, Logic 1 = active high.)
15H	R/$\overline{W}$	7:0	1*******		Bit 7—Channel Mode. Determines single-channel or dual-channel output mode. (Logic 0 = single-channel mode, Logic 1 = dual-channel mode.)
			*1******		Bit 6—Output Mode. Determine interleaved or parallel output mode. (Logic 0 = interleaved mode, Logic 1 = parallel mode.)
			0***		Bit 5—A/B Invert. Determines which port outputs the first data byte after Hsync. (Logic 0 = A port, Logic 1 = B port.)
			0*		Bit 4—4:2:2 Output Formatting Mode.
			****0***		Bit 3—Input Mux Control.
			*****11*		Bits [2:1]—Input Bandwidth.
			*******0		Bit 0—External Clock. Shuts down PLL and allows external clock to drive the part. (Logic 0 = use internal PLL, Logic 1 = bypassing of the internal PLL.)
16H	R/$\overline{W}$	7:0	11111111	Test Register	Must be set to 11111110 for proper operation.
17H	R/$\overline{W}$	7:3	00000000	Test Register	Must be set to default for proper operation.
18H	RO	7:0		Test Register	
19H	RO	7:0		Test Register	

*The AD9888 only updates the PLL divide ratio when the LSBs are written to (Register 02H).

AD9888 CONTROL REGISTER DETAIL

1.19.1 Reg[OO] Bit(s)[7-O] Chip Revision

Register 00 is an 8-bit register that represents the silicon revision.

Revision 0 = 0000 0000,

Revision 1 = 0000 0001.

1.19.2 PLL DIVIDER CONTROL

1.19.3 Reg[O1] Bit(s)[7-O] PLL Divide Ratio MSBs

The eight most significant bits of the 12-bit PLL divide ratio PLLDIV. (The operational divide ratio is PLLDIV + 1.) The PLL derives a master clock from an incoming Hsync signal. The master clock frequency is then divided by an integer value, such that the output is phase-locked to Hsync. This PLLDIV value determines the number of pixel times (pixels plus horizontal blanking overhead) per line. This is typically 20% to 30% more than the number of active pixels in the display. The 12-bit value of the PLL divider supports divide ratios from 2 to 4095. The higher the value loaded in this register, the higher the resulting clock frequency with respect to a fixed Hsync frequency. VESA has established standard timing specifications that will assist in determining the value for PLLDIV as a function of horizontal and vertical display resolution and frame rate (Table IV). However, many computer systems do not conform precisely to the recommendations, and these numbers should be used only as a guide. The display system manufacturer should provide automatic or manual means for optimizing PLLDIV. An incorrectly set

PLLDIV will usually produce one or more vertical noise bars on the display. The greater the error, the greater the number of bars produced. The power-up default value of PLLDIV is 1693 (PLLDIVM = 69H, PLLDIVL = DxH). The AD9888 updates the full divide ratio only when the LSBs are changed. Writing to this register by itself will not trigger an update.

1.19.4 Reg[O2] Bit(s)[7-4] PLL Divide Ratio LSBs

The four least significant bits of the 12-bit PLL divide ratio PLLDIV. The operational divide ratio is PLLDIV + 1. The power-up default value of PLLDIV is 1693 (PLLDIVM = 69H, PLLDIVL = DxH). The AD9888 updates the full divide ratio only when this register is written to.

1.19.5 CLOCK GENERATOR CONTROL

1.19.6 Reg[O3] Bit(s)[7-6] VCO Range Select

Two bits that establish the operating range of the clock generator. VCORNGE must be set to correspond with the desired operating frequency (incoming pixel rate). The PLL gives the best jitter performance at high frequencies. For this reason, in order to output low pixel rates and still get good jitter performance, the PLL actually operates at a higher frequency but then divides down the clock rate afterwards. Table VI shows the pixel rates for each VCO range setting. The PLL output divisor is automatically selected with the VCO range setting. The power-up default value is 01.

VCORNGE	Pixel Rate Range
00	10 - 41 MHz
01	41 - 82 MHz
10	82 - 150 MHz
11	>150 MHz

1.19.7 Reg[O3] Bit(s)[5-3] Charge Pump Current

Three bits that establish the current driving the loop filter in the clock generator. CURRENT must be set to correspond with the desired operating frequency (incoming pixel rate). The power-up default value is CURRENT = 001.

Table VII. Charge Pump Currents

Range	Current (μA)
000	50
001	100
010	150
011	250
100	350
101	500
110	750

1.19.8 Reg[O4] Bit(s)[7-3] Clock Phase Adjust

Register 04 bits 7 to 3 is a 5-bit value that adjusts the sampling phase in 32 steps across one pixel time. Each step represents an 11.25° shift in sampling phase. The power-up default value is 16.

1.19.9 CLAMP TIMING

1.19.10 Reg[O5] Bit(s)[7-O] Clamp Placement

Register 05 is an 8-bit register that sets the position of the internally generated clamp. When the external clamp control bit is set to 0, a clamp signal is generated internally, at a position established by the clamp placement and for a duration set by the clamp duration. Clamping is started (Clamp Placement) pixel periods after the trailing edge of Hsync. The clamp placement may be programmed to any value up to 255, except 0. The clamp should be placed during a time that the input signal presents a stable black-level reference, usually the back porch period between Hsync and the image. When the external clamp control bit is set to 1, this register is ignored.

1.19.11 Reg[O6] Bit(s)[7-O] Clamp Duration

Register 06 is an 8-bit register that sets the duration of the internally generated clamp. When the external clamp control bit is set to 0, a clamp signal is generated internally, at a position established by the clamp placement and for a duration set by the clamp duration. Clamping is started (Clamp Placement) pixel periods after the trailing edge of Hsync, and continues for (Clamp Duration) pixel periods. The clamp duration may be programmed to any value between 1 and 255. A value of 0 is not supported. For the best results, the clamp duration should be set to include the majority of the black reference signal time that follows the Hsync signal trailing edge. Insufficient clamping time can produce brightness changes at the top of the screen, and a slow recovery from large changes in the Average Picture Level (APL), or brightness. When the external clamp control bit is set to 1, this register is ignored.

1.19.12 HSYNC PULSE WIDTH

1.19.13 Reg[O7] Bit(s)[7-O] Hsync Output Pulse width

Register 07 is an 8-bit register that sets the duration of the Hsync output pulse.

The leading edge of the Hsync output is triggered by the internally generated, phase adjusted PLL feedback clock. The AD9888 then counts a number of pixel clocks equal to the value in this register. This triggers the trailing edge of the Hsync output, which is also phase adjusted.

1.19.14 INPUT GAIN

1.19.15 Reg[O8] Bit(s)[7-O] Red Channel Gain Adjust

Register 08 is an 8-bit word that sets the gain of the RED channel. The AD9888 can accommodate input signals with a full-scale range of between 0.5 V and 1.0 V p-p. Setting REDGAIN to 255 corresponds to an input range of 1.0 V. A REDGAIN of 0 establishes an input range of 0.5 V. Note that increasing REDGAIN results in the picture having less contrast (the input signal uses fewer of the available converter codes). See Figure 2.

1.19.16 Reg[O9] Bit(s)[7-O] Green Channel Gain Adjust

An 8-bit word that sets the gain of the GREEN channel. See REDGAIN (08).

1.19.17 Reg[OA] Bit(s)[7-O] Blue Channel Gain Adjust

An 8-bit word that sets the gain of the BLUE channel. See REDGAIN (08).

1.19.18 INPUT OFFSET

1.19.19 Reg[OB] Bit(s)[7-1] Red Channel Offset Adjust

A 7-bit offset binary word that sets the dc offset of the RED channel. One LSB of offset adjustment equals approximately one LSB change in the ADC offset. Therefore, the absolute magnitude of the offset adjustment scales as the gain of the channel is changed. A nominal setting of 63 results in the channel nominally clamping the back porch (during the clamping interval) to code 00. An offset setting of 127 results in the channel clamping to code 64 of the ADC. An offset setting of 0 clamps to code –63 (off the bottom of the range). Increasing the value of Red Offset decreases the brightness of the channel.

1.19.20 Reg[OC] Bit(s)[7-1] Green Channel Offset Adjust

A 7-bit offset binary word that sets the dc offset of the GREEN channel. See REDOFST (0B).

1.19.21 Reg[OD] Bit(s)[7-1] Blue Channel Offset Adjust

A 7-bit offset binary word that sets the dc offset of the BLUE channel. See REDOFST (0B).

1.19.22 Reg[OE] Bit(s)[7] Hsync Input Polarity Override

This register is used to override the internal circuitry that determines the polarity of the Hsync signal going into the PLL.

Table VIII. Hsync Input Polarity Override Settings

Override Bit	Result
0	Hsync Polarity Determined by Chip
1	Hsync Polarity Determined by User

The default for Hsync polarity override is 0 (polarity determined by chip).

1.19.23 Reg[OE] Bit(s)[6] HSPOL Hsync Input Polarity

A bit that must be set to indicate the polarity of the Hsync signal that is applied to the PLL Hsync input.

T**able IX. Hsync Input Polarity Settings**

HSPOL	Function
0	Active Low
1	Active High

Active Low means the leading edge of the Hsync pulse is negative-going. All timing is based on the leading edge of Hsync, which is the falling edge. The rising edge has no effect. Active High means the leading edge of the Hsync pulse is positive-going. This means that timing will be based on the leading edge of Hsync, which is now the rising edge. The device will operate if this bit is set incorrectly, but the internally generated clamp position, as established by Clamp Placement (Register 05H), will not be placed as expected, which may generate clamping errors. The power-up default value is HSPOL = 1.

1.19.24 Reg[OE] Bit(s)[5] Hsync Output Polarity

One bit that determines the polarity of the Hsync output and the SOG output. Table X shows the effect of this option. SYNC indicates the logic state of the sync pulse. The default setting for this register is 0.

Table X. Hsync Output Polarity Settings

Setting	SYNC
0	Logic 1 (Positive Polarity)
1	Logic 0 (Negative Polarity)

1.19.25 Reg[OE] Bit(s)[4] Active Hsync Override

This bit is used to override the automatic Hsync selection. To override, set this bit to Logic 1. When overriding, the active Hsync is set via Bit 3 in this register. The default for this register is 0.

Table XI. Active Hsync Override Settings

Override	Result
0	Auto determines the active interface.
1	Override, Bit 3, determines the active interface

1.19.26 Reg[0E] Bit(s)[3] Active Hsync Select

This bit is used under two conditions. It is used to select the active Hsync when the override bit is set (Bit 4). Alternately, it is used to determine the active Hsync when not overriding but both Hsyncs are detected. The default for this register is 0.

Table XII. Active Hsync Select Settings

Select	Result
0	Hsync Input
1	Sync-on-Green Input

1.19.27 Reg[0E] Bit(s)[2] Vsync Output Invert

A bit that inverts the polarity of the Vsync output. Table XIII shows the effect of this option. The default setting for this register is 0.

Table XIII. Vsync Output Polarity Settings

Setting	SYNC
0	Invert
1	Don't Invert

1.19.28 Reg[0E] Bit(s)[1] Active Vsync Override

This bit is used to override the automatic Vsync selection. To override, set this bit to Logic 1. When overriding, the active interface is set via Bit 0 in this register. The default for this register is 0.

Table XIV. Active Vsync Override Settings

Override	Result
0	Auto determines the active Vsync.
1	Override, Bit 0 determines the active Vsync.

1.19.29 Reg[0E] Bit(s)[0] Active Vsync Select

This bit is used to select the active Vsync when the over ride bit is set (Bit 1). The default for this register is 0.

Table XV. Active Vsync Select Settings

Select	**Result**
0	VSYNC Input
1	Sync Separator output

1.19.30 Reg[0F] Bit(s)[7] Clamp Input Signal Source

A bit that determines the source of clamp timing. A 0 enables the clamp timing circuitry controlled by clamp placement and clamp duration. The clamp position and duration is counted from the trailing edge of Hsync. A 1 enables the external CLAMP input pin. The three channels are clamped when the CLAMP signal is active. The polarity of CLAMP is determined by the Clamp Polarity bit (Register 0FH, Bit 6). The power-up default value is External Clamp = 0.

T**able XVI. Clamp Input Signal Source Settings**

External Clamp	**Function**
0	Internally Generated Clamp
1	Externally Provided Clamp Signal

1.19.31 Reg[0F] Bit(s)[6] Clamp Input Signal Polarity

A bit that determines the polarity of the externally provided CLAMP signal.

Table XVII. Clamp Input Signal Polarity Settings

Clamp Polarity	**Function**
0	Active Low
1	Active High

A Logic 1 means the circuit will clamp when CLAMP is Low, and pass the signal to the ADC when CLAMP is high. A Logic 0 means the circuit will clamp when CLAMP is High, and pass the signal to the ADC when CLAMP is low. The power-up default value is Clamp Polarity = 1.

1.19.32 Reg[0F] Bit(s)[5] COAST Select

This bit is used to select the active coast source. The choices are the coast input pin or Vsync. If Vsync is selected, the additional decision of using the Vsync input pin or the output from the sync separator needs to be made (Register 0EH, Bits 1, 0). The default for this register is 0.

Table XVIII. COAST Source Selection Settings

Select	**Result**
0	Coast Input
1	Vsync per Reg 0E bits 1-9

1.19.33 Reg[0F] Bit(s)[4] COAST Input Polarity Override

This register is used to override the internal circuitry that determines the polarity of the coast signal going into the PLL.

Table XIX. COAST Input Polarity Override Settings

Override	Result
0	COAST Polarity Determined by Chip
1	COAST Polarity Determined by User

1.19.34 Reg[OF] Bit(s)[3] COAST Input Polarity

A bit to indicate the polarity of the COAST signal that is applied to the PLL COAST input. Active LOW means that the clock generator will ignore Hsync inputs when COAST is low, and continue operating at the same nominal frequency until COAST goes high. Active High means that the clock generator will ignore Hsync inputs when COAST is high, and continue operating at the same nominal frequency until COAST goes low. This function needs to be used along with the COAST polarity override bit (Bit 4). The power-up default value is CSTPOL = 1.

Table XX. COAST Input Polarity Settings

CSTPOL	Function
0	Active Low
1	Active High

1.19.35 Reg[OF] Bit(s)[2] Seek Mode Override

This bit is used to either allow or disallow the low power mode. The low power mode (seek mode) occurs when there are no signals on any of the Sync inputs. The default for this register is 1.

Table XXI. Seek Mode Override Settings

Select	Result
0	Disallow Seek Mode
1	Allow Seek Mode

1.19.36 Reg[OF] Bit(s)[1] PWRDN

This bit is used to put the chip in power-down mode. In this mode, the chip's power dissipation is reduced to a fraction of the typical power (see the Electrical Characteristics table for exact power dissipation). When in power- down, the HSOUT, VSOUT, DATACK, DATACK, and all 48 of the data outputs are put into a high impedance state. (Note: the SOGOUT output is not put into high impedance.) Circuit blocks that continue to be active during power-down include the voltage references, sync processing, sync detection, and the serial register. These blocks facilitate a fast startup from power-down. The default for this register is 1.

Table XXII. Power-Down Settings

Select	Result
0	Power Down
1	Normal Operation

1.19.37 Reg[1O] Bit(s)[7-3] Sync-on-Green Slicer Threshold

This register allows the comparator threshold of the Sync- on-Green slicer to be adjusted. This register adjusts it in steps of 10 mV, with the minimum setting equal to 10 mV and the maximum setting equal to 330 mV. The default setting is 15 and corresponds to a threshold value of 0.16 V.

1.19.38 Reg[1O] Bit(s)[2] Red Clamp Select

A bit that determines whether the red channel is clamped to ground or to mid-scale. For RGB video, all three channels are referenced to ground. For YcbCr (or YUV), the Y channel is referenced to ground, but the CbCr channels are referenced to mid-scale. Clamping to mid-scale actually clamps to Pin 9. The default setting for this register is 0.

Table XXIII. Red Clamp Select Settings

CSTPOL	Function
0	Clamp to ground
1	Clamp to mid scale (pin 9)

1.19.39 Reg[1O] Bit(s)[1] Blue Clamp Select

A bit that determines whether the blue channel is clamped to ground or to midscale. Clamping to midscale actually clamps to Pin 24. The default setting for this register is 0.

Table XXIV. Blue Clamp Select Settings

Select	Function
0	Clamp to ground
1	Clamp to mid scale (pin 24)

1.19.40 Reg[11] Bit(s)[7-O] Sync Separator Threshold

This register is used to set the responsiveness of the sync separator. It sets how many internal 5 MHz clock periods the sync separator must count to before toggling high or low. It works like a low-pass filter to ignore Hsync pulses in order to extract the Vsync signal. This register should be set to some number greater than the maximum Hsync pulse width. Note: the sync separator threshold uses an internal dedicated clock with a frequency of approximately 5 MHz. The default for this register is 32.

1.19.41 Reg[12] Bit(s)[7-O] Pre-COAST

This register allows the COAST signal to be applied prior to the Vsync signal. This is necessary in cases where pre- equalization pulses are present. The step size for this control is one Hsync period. The default is 0.

1.19.42 Reg[13] Bit(s)[7-O] Post-COAST

This register allows the COAST signal to be applied following to the Vsync signal. This is necessary in cases where post-equalization pulses are present. The step size for this control is one Hsync period. The default is 0.

1.19.43 Reg[14] Bit(s)[7] Hsync Detect

This bit is used to indicate when activity is detected on the selected Hsync input pin. If HSYNC is held high or low, activity will not be detected. The sync processing block diagram shows where this function is implemented.

Table XXV. Hsync Detection Results

Detec t	Function
0	No Activity Detected
1	Activity Detected

1.19.44 Reg[14] Bit(s)[6] AHS – Active Hsync

This bit indicates which Hsync input source is being used by the PLL (Hsync input or Sync-on-Green). Bits 7 and 1 in this register are what determine which source is used. If both Hsync and SOG are detected, the user can determine which has priority via Bit 3 in register 0EH. The user can override this function via Bit 4 in Register 0EH. If the override bit is set to Logic 1, then this bit will be forced to whatever the state of Bit 3 in Register 0EH is set to. AHS = 0 means use the HSYNC pin input for HSYNC. AHS = 1 means use the SOG pin input for HSYNC. The override bit is in Register 0EH, Bit 4.

Table XXVI. Active Hsync Results

Bit 7 (Hsync Detect)	Bit 1 (SOG Detect)	Bit 4 Reg 0E (Override)	AHS
0	0	0	Bit 3 in OEh
0	1	0	1
1	0	0	0
1	1	0	Bit 3 in 0Eh
X	X	1	Bit 3 in 0Eh

1.19.45 Reg[14] Bit(s)[5] Detected Hsync Input Polarity Status

This bit reports the status of the HSYNC input polarity detection circuit. It can be used to determine the polarity of the HSYNC input. The detection circuit's location is shown in the Sync Processing Block Diagram (Figure 25).

Table XXVII. Detected Hsync Input Polarity Status

Hsync Polarity Status	Function
0	Hsync in negative
1	Hsync is positive

1.19.46 Reg[14] Bit(s)[4] Vsync Detect

This bit is used to indicate when activity is detected on the selected Vsync input pin. If Vsync is held high or low, activity will not be detected. The sync processing block diagram (Figure 25) shows where this function is implemented.

Table XXVIII. Vsync Detection Results

Detec t	Function
0	No Activity Detected
1	Activity Detected

1.19.47 Reg[14] Bit(s)[3] AVS – Active Vsync

This bit indicates which Vsync source is being used; the Vsync input or the output from the sync separator. Bit 4 in this register is what determines which is active. If both Vsync and SOG are detected, the user can determine which has priority via Bit 0 in Register 0EH. The user can override this function via Bit 1 in Register 0EH. If the override bit is set to Logic 1, this bit will be forced to whatever the state of Bit 0 in Register 0EH is set to. AVS = 1 means Sync separator. AVS = 0 means Vsync input. The override bit is in register 0Eh, Bit 1.

Table XXIX. Active Vsync Results

Bit 5 (Vsync Detect)	Override	AVS
0	0	1
1	0	0
X	1	Bit 0 in 0Eh

1.19.48 Reg[14] Bit(s)[2] Detected Vsync Output Polarity Status

This bit reports the status of the Vsync output polarity detection circuit. It can be used to determine the polarity of the Vsync input. The detection circuit's location is shown in the sync processing block diagram (Figure 25).

Table XXX. Detected Vsync Input Polarity Status

Vsync Polarity Status	Result
0	Vsync Polarity is Active high
1	Vsync Polarity is active low

1.19.49 Reg[14] Bit(s)[1] Sync-on-Green Detect

This bit is used to indicate when Sync activity is detected on the selected Sync-on-Green input pin. The Sync Processing Block Diagram (Figure 25) shows where this function is implemented.

Table XXXI. Sync-on-Green Detection Results

Detec t	Function
0	No Activity Detected
1	Activity Detected

1.19.50 Reg[14] Bit(s)[O] Detected COAST Polarity Status

This bit reports the status of the coast input polarity detection circuit. It can be used to determine the polarity of the COAST input. The detection circuit's location is shown in Figure 25.

Table XXXII. Detected COAST Input Polarity Status

Coast Polarity Status	Result
0	Coast Polarity is Active Negative
1	Coast Polarity is active Positive

1.19.51 MODE CONTROL

1.19.52 Reg[15] Bit(s)[7] Channel Mode

A bit that determines whether all pixels are presented to a single port (A), or alternating pixels are de-multiplexed to Ports A and B. When DEMUX = 0, Port B outputs are in a high impedance state. The maximum data rate for single-port mode is 110 MHz. The timing diagrams starting with Figure 13 show the effects of this option. The power-up default value is 1.

Table XXXIII. Channel Mode Settings

DEMUX	Function
0	All the data goes to port A
1	Alternate pixels go to port B

1.19.53 Reg[15] Bit(s)[6] Output Mode

A bit that determines whether all pixels are presented to Port A and Port B simultaneously on every second DATACK rising edge, or alternately on Port A and Port B on successive DATACK rising edges. When in single port mode (DEMUX = 0), this bit is ignored. The timing diagrams (Figure 17) show the effects of this option. The power-up default value is PARALLEL = 1.

Table XXXIV. Output Mode Settings

PARALLEL	Function
0	Data is interleaved
1	Data is simultaneous on every other data clock.

1.19.54 Reg[15] Bit(s)[5] Output Port Phase

One bit that determines whether even pixels or odd pixels go to Port A. In normal operation (OUTPHASE = 0) when operating in dual-port output mode (DEMUX = 1), the first sample after the Hsync leading edge is presented at Port A. Every subsequent ODD sample appears at Port A. All EVEN samples go to Port B. When OUTPHASE = 1, these ports are reversed and the first sample goes to Port B. When DEMUX = 0, this bit is ignored as data always comes out of only Port A.

Table XXXV. Output Port Phase Settings

OUTPHASE	First Pixel after Hsync
0	Port A
1	Port B

1.19.55 Reg[15] Bit(s)[4] 4:2:2 Output Mode Select

A bit that configures the output data in 4:2:2 mode. This mode can be used to reduce the number of data lines used from 24 down to 16 for applications using YUV, YCbCr, or YPbPr graphics signals. A timing diagram for this mode is shown in Figure 12. Recommended input and output configurations are shown in Table XXXVII. In 4:2:2 modes, the red and blue channels can be interchanged to help satisfy board layout or timing requirements, but the green channel must be configured for Y.

Table XXXVI. 4:2:2 Output Mode Select

Select	Output Mode
0	4:4:4
1	4:2:2

Table XXXVII. 4:2:2 Input/Output Configuration

Channel	Input Connection	Output Format
Red	V	U/V
Green	Y	Y

Blue	U	Tristate

1.19.56 Reg[15] Bit(s)[3] Input Mux Control

A bit that selects either analog inputs from Channel 0 or the analog inputs from Channel 1.

Table XXXVIII. Input Mux Control

Control	**Channel Selected**
0	Channel 0
1	Channel 1

1.19.57 Reg[15] Bit(s)[2-1] Analog Bandwidth Control

Two bits that select the analog bandwidth.

Table XXXIX. Analog Bandwidth Control

Bit 2	Bit 1	Analog BW
1	1	500 MHz
1	0	300 MHz
0	1	150 MHz
0	0	75 MHz

1.19.58 Reg[15] Bit(s)[O] External Clock Select

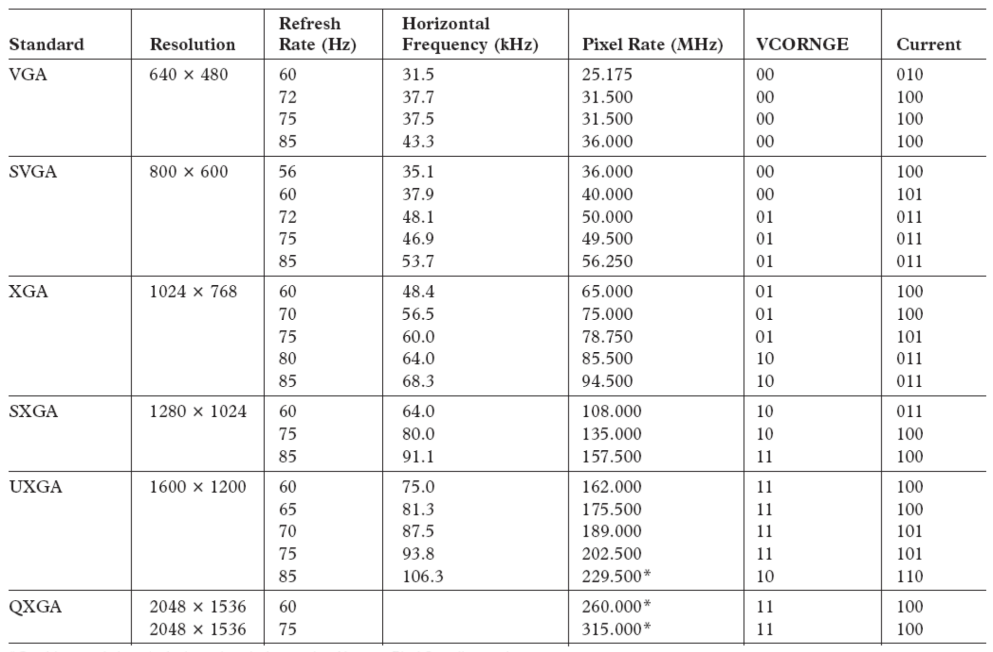
A bit that determines the source of the pixel clock. Logic 0 enables the internal PLL that generates the pixel clock from an externally provided HSYNC. A Logic 1 enables the external CKEXT input pin. In this mode, the PLL Divide Ratio (PLLDIV) is ignored. The clock phase adjust (PHASE) is still functional. The power-up default value is EXTCLK = 0.

Table XL. External Clock Select Settings

EXTCLK	Function
0	Internally Generated Clock
1	Externally Provided Clock Signal

6. ***EXAMPLE SETTINGS***

Table IV. Recommended VCO Range and Charge Pump Current Settings for Standard Display Formats



Standard	Resolution	Refresh Rate (Hz)	Horizontal Frequency (kHz)	Pixel Rate (MHz)	VCORNGE	Current
VGA	640 × 480	60	31.5	25.175	00	010
		72	37.7	31.500	00	100
		75	37.5	31.500	00	100
		85	43.3	36.000	00	100
SVGA	800 × 600	56	35.1	36.000	00	100
		60	37.9	40.000	00	101
		72	48.1	50.000	01	011
		75	46.9	49.500	01	011
		85	53.7	56.250	01	011
XGA	1024 × 768	60	48.4	65.000	01	100
		70	56.5	75.000	01	100
		75	60.0	78.750	01	101
		80	64.0	85.500	10	011
		85	68.3	94.500	10	011
SXGA	1280 × 1024	60	64.0	108.000	10	011
		75	80.0	135.000	10	100
		85	91.1	157.500	11	100
UXGA	1600 × 1200	60	75.0	162.000	11	100
		65	81.3	175.500	11	100
		70	87.5	189.000	11	101
		75	93.8	202.500	11	101
		85	106.3	229.500*	10	110
QXGA	2048 × 1536	60		260.000*	11	100
	2048 × 1536	75		315.000*	11	100

*Graphics sampled at 1/2 the incoming pixel rate using Alternate Pixel Sampling mode.

For a video source that is 1280x1024 at 60 Hz frame rate, non-interlaced looking in the table above the horizontal frequency is 64 kHz, and the dot clock is 108 MHz. The number of dot clocks in one line time is 108 MHz / 64 kHz = 1687.5 so we round down to 1687. Note the timing values published by display adapter manufactures are often wrong because the adapter has been re-designed after the documentation was written. You will always need to tune your settings to your video source. In this case the actual pixel clock frequency may be wrong, and it is actually 107.968 MHz, or the horizontal frequency is wrong and is actually 64,019 Hz. So the value for the PLL Divisor is 1687, which means we put 1686 into the PLL divisor register.

Again looking at the table we see that the current value is 3 (011 binary), and the VCO range value is 2 (10 binary). We setup our cap control file for these values, and then adjust the phase until the video is the highest quality.

If we also need a good match for the range of colors then we would also adjust the red, green and blue, offset and gain values. PLEASE read the AD9888 data sheet before you attempt this. The gain and offset interact in strange ways, and the gain register is opposite of what you might expect.

The values for the {AD9888B.1] and [AD9888B.2] sections need to be adjusted to match those in the capture file example above, to complete the example. These settings worked well with a Toshiba Laptop output.

7. <u>TUNING THE SETTINGS</u>

When tuning the settings consider the follow suggestions.

Use the highest quality cable you can use. A bad cable will prevent you from tuning the settings in any meaningful way. Cheap cables for KVM applications do not work well. The best cable uses 75 ohm coax for all the signals.

If you are trying to sample video for which you do not have the CORRECT timing, then measure it the video timing with an oscilloscope first. Try to measure the timing as accurately as possible as your settings will have to be accurate to less than a tenth of a percent typically. If your source has separated syncs, then use a frequency counter to measure the frequency. Some flat panel displays will tell you the frequencies they have detected.

Set the PLL Divisor first. The other settings can not improve on a bad PLL Divisor setting. Be prepared to shift the value up and down by small amounts.

The PLL Divisor affects the video in a possibly un-expected way. If you increase the PLL Divisor the pixel frequency will try to go up, which will add additional pixels in the line. So if the image looks stable, but there aren't enough pixels being displayed increase the PLL Divisor.

After setting the PLL Divisor, set the PLL current. The quality of the phase lock loop lock is strongly affected by the PLL current setting. The larger the current the higher the gain. First start with a lower current. If the setting is too high the PLL will oscillate causing trembling pixels, or wavy vertical columns. Look at the last pixels in the image (on the right) to see problems with oscillation, or bad Divisor values.

Set the Phase last. The phase will improve the sharpness of the image. A poorly set phase can cause odd gray vertical lines in the display. A poorly set phase will reduce the contrast in an image.

You will also need to set the number of lines in the image. If set the ROI too large the FPGA may not give you an image at all, so start with low settings on the number of lines.

Finally if you intend to use the interface in an unattended application, make sure your settings are not too tight. The PLL is affected by the amount of noise in the system. If you make the gain very high it can go into oscillations for some images.

8. GENERAL INFORMATION

1.20 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**

1.21 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.

1.21.1 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 the ALRT, ALFAST, or FASTLIB 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 98.

_____ 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.

1.21.2 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.

1.21.2.1 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