

Camera Link (Alpha Data) API User Guide (Win32)

Document version: 1.0.0.0
© Copyright 2005 Alpha Data

Contents

Introduction	3
Standard Camera Link Functions	5
clGetNumBytesAvail	5
clGetSupportedBaudRates	6
clSerialClose	7
clSerialInit.....	8
clSerialRead.....	10
clSerialWrite	11
clSetBaudRate	12
Alpha Data Camera Link Functions.....	13
clADAcquireConsecutiveFrames	13
clADGetFrame	15
clADSetRegionOfInterest.....	16
clADStartFrameAcquisition	18
clADStopFrameAcquisition	20
Transferred Frame Format	21

Introduction

The Camera Link (Alpha Data) API exposes a number of functions to interface with the XRM-CAMERALINK module and digital cameras.

The standard Camera Link API functions are prefixed with "cl".

The functions that are specific to Alpha Data are prefixed with "clAD".

The API is supported currently for Windows XP only.

The API uses ADM-XRC2 driver v3.10 or later.

The API is camera independent, however, the sample application was developed for the JAI CV-M4/M7 camera.

Header Files

To use the API, an application must include the API Header Files.

clserial.h	Function prototypes
clstatus.h	Status codes
clbaudrate.h	Baudrate codes

Supported Boards

ADM-XRC-II

ADM-XP (ADMXRC2_PRO)

ADM-XRC-4LX

ADM-XRC-4SX

ADM-XPI

Xilinx Bitstream File

The Xilinx bitstream file must be located in a directory as specified by the Environment Variable **ADMXRC_CL**.

The naming convention for the bitstream file is:

cl-*boardtype-fpgatype*.bit

boardtype must be one of the following codes:

Board Type	<i>boardtype</i>
ADM-XRC-II	xrc2
ADM-XP (ADM-XRC-II PRO)	xrc2p
ADM-XRC-4LX	xrc4lx
ADM-XRC-4SX	xrc4sx
ADM-XPI	xpi

fpgatype must be one of the following codes:

FPGA Type	<i>fpgatype</i>
Virtex-II 2V3000	2v3000
Virtex-II 2V4000	2v4000
Virtex-II 2V6000	2v6000
Virtex-II 2V8000	2v8000
Virtex-II 2V10000	2v10000
Virtex-II Pro 2VP70	2vp70
Virtex-II Pro 2VP100	2vp100
Virtex-II Pro 2VP125	2vp125
Virtex-4	4vlx60
Virtex-4	4vlx80
Virtex-4	4vlx100
Virtex-4	4vsx55

Standard Camera Link Functions

clGetNumBytesAvail

Prototype

```
int  
clGetNumBytesAvail(  
    void*          serialRef,  
    unsigned long* numBytes);
```

Arguments

Argument	Type	Purpose
SerialRef	In	Pointer to the device to as obtained by clSerialInit .
numBytes	Out	The number of bytes currently available to be read from the port.

Return Value

Value	Meaning
CL_ERR_NO_ERR	numBytes was updated successfully with the number of bytes available.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid

Description

This function outputs the number of bytes that are received at the port specified by **serialRef** but that are not yet read out. Use **clSerialRead** to read the data from the serial device.

clGetSupportedBaudRates

Prototype

```
int  
clGetSupportedBaudRates(  
    void* serialRef,  
    unsigned long* baudRates);
```

Arguments

Argument	Type	Purpose
SerialRef	In	Pointer to the device to as obtained by clSerialInit .
baudRates	Out	List of supported baud rates.

Return Value

Value	Meaning
CL_ERR_NO_ERR	baudRates was updated successfully with list of supported baud rates.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid

Description

This function returns the valid baud rates of the current interface.

baudRates is a bitfield that describes all the supported baud rates of the serial port. The baud rate values are represented by the **CL_BAUDRATE** constants.

cISerialClose

Prototype

```
void  
cISerialClose(  
    void*                serialRef);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by cISerialInit .

Description

This function closes the serial device and cleans up the resources associated with **serialRef**. Upon return **serialRef** is no longer usable.

clSerialInit

Prototype

```
int  
clSerialInit(  
    unsigned long    serialIndex,  
    void**           serialRefPtr);
```

Arguments

Argument	Type	Purpose
serialIndex	In	Zero-based index value
serialRefPtr	Out	Pointer to the opened device.

Return Value

Value	Meaning
CL_ERR_NO_ERR	The device was opened and a bitstream was loaded into the FPGA successfully.
CL_ERR_OUT_OF_MEMORY	System is out of memory and could not perform required actions
CL_ERR_INVALID_INDEX	Not a valid index
CL_ERR_AD_FILE_NOT_FOUND	The bitstream file or the file's location could not be found
CL_ERR_AD_FPGA_MISMATCH	The device targeted by the bitstream file did not match the device fitted to the card
CL_ERR_AD_INVALID_BOARD_TYPE	Not a valid board type
CL_ERR_AD_INVALID_FILE	The bit file appears not to be a valid bitstream
CL_ERR_AD_INVALID_FPGA_TYPE	The FPGA fitted to the board is not valid
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function initialises the device referred to by serialIndex and returns a pointer to an internal serial reference structure.

The pointer returned in the **serialRefPtr** parameter should be used in all further API calls that need access to this device. When access is no longer required, call **clSerialClose** to close the pointer and free the device.

The function detects the ADM-XRC board and the FPGA on the card is configured with a Xilinx bitstream file (.bit). The location of the bitstream file is specified by the System Environment Variable **CL_ADMXRC2**. The name of the bitstream file must be of the form **cl-boardtype-fpgatype.bit**.

boardtype must be one of the following codes:

Board Type	boardtype
ADM-XRC-II	xrc2
ADM-XP (ADM-XRC-II PRO)	xrc2p

fpgatype must be one of the following codes:

FPGA Type	<i>fpgatype</i>
Virtex-II 2V3000	2v3000
Virtex-II 2V4000	2v4000
Virtex-II 2V6000	2v6000
Virtex-II 2V8000	2v8000
Virtex-II 2V10000	2v10000
Virtex-II Pro 2VP70	2vp70
Virtex-II Pro 2VP100	2vp100
Virtex-II Pro 2VP125	2vp125

The local bus clock frequency is set to 33MHz.

clSerialRead

Prototype

```
int  
clSerialRead(  
    void*                serialRef,  
    char*                buffer,  
    unsigned long*       numBytes,  
    unsigned long        serialTimeout);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
buffer	In	Pointer to a user-allocated buffer.
numBytes	In/Out	The number of bytes requested by the caller.
serialTimeout	In	Indicates the timeout in milliseconds.

Return Value

Value	Meaning
CL_ERR_NO_ERR	numBytes were read successfully into buffer .
CL_ERR_INVALID_REFERENCE	The serial reference is not valid
CL_ERR_TIMEOUT	Operation not completed within the specified timeout period.
CL_ERR_OUT_OF_MEMORY	System is out of memory and could not perform required actions
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function reads **numBytes** from the serial device referred by **serialRef**.

clSerialRead will return when **numBytes** are available at the serial port or when the **serialTimeout** period has passed. Upon success, **numBytes** are copied into **buffer**. In the case of any error, including CL_ERR_TIMEOUT, **buffer** is not affected.

The caller must ensure that the **buffer** is at least **numBytes** in size.

clSerialWrite

Prototype

```
int  
clSerialWrite(  
    void*          serialRef,  
    char*          buffer,  
    unsigned long* bufferSize,  
    unsigned long  serialTimeout);
```

Arguments

Argument	Type	Purpose
SerialRef	In	Pointer to the device to as obtained by clSerialInit .
buffer	In	Contains data to write to the serial port.
bufferSize	In/Out	Contains the buffer size.
serialTimeout	In	Indicates the timeout in milliseconds.

Return Value

Value	Meaning
CL_ERR_NO_ERR	Data in the buffer was written successfully to the serial device.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid
CL_ERR_TIMEOUT	Operation not completed within the specified timeout period.
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function writes data in the **buffer** to the serial device referenced by **serialRef**.

bufferSize indicates the maximum number of bytes to be written. Upon a successful call, **bufferSize** contains the number of bytes written to the serial device.

clSetBaudRate

Prototype

```
int  
clSetBaudRate(  
    void*          serialRef,  
    unsigned long  baudRate);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
baudRate	In	The baud rate for the device.

Return Value

Value	Meaning
CL_ERR_NO_ERR	numBytes was updated successfully with the number of bytes available.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid
CL_ERR_BAUD_RATE_NOT_SUPPORTED	The baud rate specified is not supported by the device.

Description

This function sets the baud rate for the serial port of the selected device. Use **clGetSupportedBaudRates** to determine the supported baud rates.

baudRate must be one of the values represented by the **CL_BAUDRATE** constants.

Apha Data Camera Link Functions

clADAcquireConsecutiveFrames

Prototype

```
int  
clADTriggerFrames(  
    void*                serialRef,  
    void*                buffer,  
    unsigned long        bufferSize,  
    HANDLE               semaphore,  
    unsigned long        serialTimeout,  
    unsigned long        frameCount);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
buffer	In	Pointer to a user-allocated buffer.
bufferSize	In	Size in bytes of buffer .
semaphore	In	Pointer to user-defined semaphore or NULL.
serialTimeout	In	Indicates the timeout in milliseconds.
frameCount	In/Out	The number of frames to transfer.

Return Value

Value	Meaning
CL_ERR_NO_ERR	Successfully transferred frames from the camera to the host.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_BUFFER_TOO_SMALL	buffer not large enough to hold a frame.
CL_ERR_OUT_OF_MEMORY	System is out of memory and could not perform required actions.
CL_ERR_AD_INVALID_PARAMETER	Frame count is invalid.
CL_ERR_AD_NO_DMADESC	All DMA descriptors were in use.
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function notifies the camera to start capturing images and transferring a specified number of frames to the host. The size of the transferred frames must have been set previously by **Error! Not a valid bookmark self-reference..** The function terminates when all the frames have been transferred and the final **semaphore**, if used, has been caught by the host.

buffer is used to hold the transferred frames.

bufferSize indicates the size of **buffer** in bytes. **buffer** must be large enough to hold at least one transferred frame. The number of transferred frames that can be held in **buffer** will be

$$(int)(bufferSize / numBytes)$$

where **numBytes** was calculated in the function **Error! Not a valid bookmark self-reference..**

semaphore is an optional argument. **semaphore** can be NULL or a user-defined semaphore created by the Windows API function **CreateSemaphore**. Upon a successful call, **semaphore** will be in a non-signalled state.

If **semaphore** is defined, it is used to notify the caller that a frame has been transferred and can be obtained by the caller. The caller should wait on **semaphore**, preferably in a separate thread. Every time a frame has been transferred into **buffer**, **semaphore** is released by the Camera Link API. The caller should call **ciADGetFrame** to obtain the transferred frame. The caller must call **ciADGetFrame** the same number of times that **semaphore** has been released for this function to terminate.

ciADGetFrame cannot be used to obtain the transferred frames if **semaphore** is NULL. The transferred frames can be accessed from **buffer** when **ciADAcquireConsecutiveFrames** has terminated. If **buffer** is not large enough to hold all the transferred frames then only the last n frames will be available, where n is the number of frames that can be held in the **buffer** (see above for the number of frames calculation).

frameCount is the number of consecutive images to be captured and transferred as frames. **frameCount** must be greater than 0. Upon termination of this function, **frameCount** will contain the number of frames transferred successfully.

If the number of times **semaphore** is released exceeds its maximum count, then **status** will be returned with the value `CL_ERR_AD_TOO_MANY_POSTS`. Every time this occurs, **frameCount** will be decremented. The Camera Link API will continue to capture the images and transfer the frames.

If the DMA transfer overflows then **status** will be returned with the value `CL_ERR_AD_FRAME_OVERFLOW`. Every time this occurs, **frameCount** will be decremented. The Camera Link API will continue to capture the images and transfer the frames. DMA transfer overflow can be caused by interruptions by the operating system to the DMA process.

If `CL_ERR_AD_TOO_MANY_POSTS` and `CL_ERR_AD_FRAME_OVERFLOW` both occur then **status** will contain the value of the error that occurred last.

clADGetFrame

Prototype

```
int  
clADCaptureFrame(  
    void*                serialRef,  
    void*                frameBuffer,  
    unsigned long        frameBufferSize,  
    unsigned long        serialTimeout);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
frameBuffer	In	Pointer to a user-allocated buffer.
frameBufferSize	In	Size in bytes of frameBuffer .
serialTimeout	In	Indicates the timeout in milliseconds.

Return Value

Value	Meaning
CL_ERR_NO_ERR	Successfully copied a frame to the host.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_AD_FRAME_NOT_AVAILABLE	A transferred frame is not available to be copied.
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function copies to the host a frame transferred from the camera. The size of the transferred frame must have been set previously by **Error! Not a valid bookmark self-reference..** This function should be called for every **semaphore** released by **clADStartFrameAcquisition** or **clADAcquireConsecutiveFrames**. If a **semaphore** is not being used, then this function cannot be used to obtain the transferred frames.

frameBuffer is used to hold the transferred frames. See Transferred Frame for the definition of the format.

frameBufferSize indicates the size of the **frameBuffer** that will hold the transferred frame. **frameBufferSize** should be the same as the number of bytes required for a frame. This value was returned as **numBytes** in the function **Error! Not a valid bookmark self-reference..**

If **frameBufferSize** is less than **numBytes**, then only part of the image will be copied to buffer. A subsequent call to **clADGetFrame** will copy from the start of the next frame.

If **frameBufferSize** is greater than **numBytes**, then the captured image is copied into **frameBuffer**. The excess space in **frameBuffer** is not used. A subsequent call to **clADGetFrame** will copy from the start of the next frame.

If this function is called after transfer has been stopped by **clADStopFrameAcquisition**, then **status** will contain the value **CL_ERR_AD_FRAME_NOT_AVAILABLE**. **semaphore** as defined in **clADStartFrameAcquisition**, may still be in a signalled state.

If this function is called after transfer has stopped when **clADAcquireConsecutiveFrames** has terminated, then **status** will contain the value **CL_ERR_AD_FRAME_NOT_AVAILABLE**. **semaphore** as defined in **clADAcquireConsecutiveFrames**, will be in a non-signalled state. The caller must call this function the same number of times that **semaphore** has been released for **clADAcquireConsecutiveFrames** to terminate.

clADSetRegionOfInterest

Prototype

```
int  
clADSetRegionOfInterest(  
    void*          serialRef,  
    unsigned long  lineCount,  
    unsigned long  lineOffset,  
    unsigned long* pixelCountRef,  
    unsigned long  pixelOffset,  
    BOOL          bayerInterpolation,  
    unsigned long* numBytes);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
lineCount	In/Out	Number of lines to transfer from the captured image.
lineOffset	In/Out	Line offset of the captured image.
pixelCountRef	In/Out	Pointer to number of pixels per line to transfer from the captured image.
pixelOffset	In/Out	Pixel offset of the captured image.
bayerInterpolation	In/Out	Not used
numBytes	Out	Number of bytes required to transfer one frame.

Return Value

Value	Meaning
CL_ERR_NO_ERR	Successfully set the region of interest.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_AD_CAPTURE_IN_PROGRESS	Frame transfer is in progress.
CL_ERR_AD_INVALID_PARAMETER	lineCount or pixelCount is not valid.

Description

This function specifies a region of interest or an area of the image captured by the camera that is transferred as a frame to the host by the DMA process. This function must be called prior to the start of capturing images by either **clADStartFrameAcquisition** or **clADAcquireConsecutiveFrames**.

lineCount indicates the maximum number of lines in the frame that will be transferred from the captured image. **lineCount** must be greater than zero, otherwise, the region of interest's size would be zero.

lineOffset indicates the offset from the first line of the captured image that will be transferred.

If the sum of **lineCount** and **lineOffset** exceed the number of lines in the captured image then the frame transferred will be undefined.

pixelCountRef indicates the maximum number of pixels per line in the frame that will be transferred from the captured image. Pixels are transferred in pairs; therefore, if **pixelCountRef** is odd it will be rounded down to the nearest even number. **pixelCountRef** must be greater than zero, otherwise, the frame's size would be zero. Upon a successful call, **pixelCountRef** will contain the actual number of pixels per line that will be transferred.

pixelOffset indicates the offset from the first pixel of each line of the captured image that will be transferred.

If the sum of **pixelCountRef** and **pixelOffset** exceed the number of pixels in each line in the captured image then the frame transferred will be undefined.

ciADSetRegionOfInterest does not change or reference the camera's setting. The API is unable to determine the image size from the camera's setting, therefore, the caller must verify the values of **lineCount**, **lineOffset**, **pixelCount** and **pixelOffset** prior to calling **ciADSetRegionOfInterest**.

Upon a successful call, **numBytes** is the number of bytes required to hold one frame. It is calculated as:

$$\mathbf{lineCount} * (*\mathbf{pixelCount}) * \mathbf{BYTES_PER_PIXEL}$$

The value of *BYTES_PER_PIXEL* is two. See **Transferred Frame** for more information.

The region of interest cannot be changed while images are in the process of being captured from the camera. The caller must either call **ciADStopFrameAcquisition** to stop the process or wait for **ciADAcquireConsecutiveFrames** to finish executing.

bayerInterpolation is ignored.

clADStartFrameAcquisition

Prototype

```
int  
clADStartCapture(  
    void*                serialRef,  
    void*                buffer,  
    unsigned long        bufferSize,  
    HANDLE               semaphore,  
    unsigned long        serialTimeout);
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
buffer	In	Pointer to a user-allocated buffer.
bufferSize	In	Size in bytes of buffer .
semaphore	In	Pointer to user-defined semaphore or NULL.
serialTimeout	In	Indicates the timeout in milliseconds.

Return Value

Value	Meaning
CL_ERR_NO_ERR	Started successfully the transfer of the frames.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_OUT_OF_MEMORY	System is out of memory and could not perform required actions
CL_ERR_BUFFER_TOO_SMALL	buffer not large enough to hold frame
CL_ERR_AD_NO_DMADESC	All DMA descriptors were in use
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function notifies the camera to start capturing images and transferring frames to the host. The size of the transferred frames must have been set previously by **clADSetRegionOfInterest**. The function **clADStopFrameAcquisition** notifies the camera to stop capturing images and transferring frames to the host.

buffer is used to hold the transferred frames **clADSetRegionOfInterest**.

bufferSize indicates the size of **buffer** in bytes. **buffer** must be large enough to hold at least one transferred frame. The number of transferred frames that can be held in **buffer** will be

$$(int)(bufferSize / numBytes)$$

where **numBytes** was calculated in the function.

semaphore is an optional argument. **semaphore** can be NULL or a user-defined semaphore created by the Windows API function **CreateSemaphore**. Upon a successful call, **semaphore** will be in an undefined state.

If **semaphore** is defined, it is used to notify the caller that a frame has been transferred and can be obtained by the caller. The caller should wait on **semaphore**, preferably in a separate thread. Every time a frame has been transferred into **buffer**, **semaphore** is released by the Camera Link API. The caller should call **clADGetFrame** to obtain the transferred frame.

clADGetFrame cannot be used to obtain the transferred frames if **semaphore** is NULL. The transferred frames can be accessed from **buffer** when **clADAcquireConsecutiveFrames**

has terminated. If **buffer** is not large enough to hold all the transferred frames then only the last n frames will be available, where n is the number of frames that can be held in the **buffer** (see above for the number of frames calculation).

If the number of times **semaphore** is released exceeds its maximum count, then **status** will be returned with the value `CL_ERR_AD_TOO_MANY_POSTS`. The Camera Link API will continue to capture the images and transfer the frames.

If the DMA transfer overflows then **status** will be returned with the value `CL_ERR_AD_FRAME_OVERFLOW`. The Camera Link API will continue to capture the images and transfer the frames. DMA transfer overflow can be caused by interruptions by the operating system to the DMA process.

If `CL_ERR_AD_TOO_MANY_POSTS` and `CL_ERR_AD_FRAME_OVERFLOW` both occur then **status** will contain the value of the error that occurred last.

clADStopFrameAcquisition

Prototype

```
int  
clADStopCapture(  
    void*                serialRef,  
    unsigned long        serialTimeout)
```

Arguments

Argument	Type	Purpose
serialRef	In	Pointer to the device to as obtained by clSerialInit .
serialTimeout	In	Indicates the timeout in milliseconds.

Return Value

Value	Meaning
CL_ERR_NO_ERR	Stopped successfully the transfer of the frames.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_AD_FRAME_OVERFLOW	DMA transfer overflowed.
CL_ERR_AD_TOO_MANY_POSTS	Too many posts were sent to the semaphore.
CL_ERR_AD_OS_ERROR	Error caused by an operating system function.

Description

This function notifies the camera to stop capturing images and transferring frames to the host. The function **clADStartFrameAcquisition** notified the camera to start capturing images and transferring frames to the host.

If **semaphore** was defined in **clADStartFrameAcquisition** then transferred frames can be obtained using **clADGetFrame**. If **clADGetFrame** is called after the frame transfer has been stopped by this function, then **semaphore** may still be in a signalled state.

If the number of times **semaphore** was released exceeded its maximum count, then **status** will be returned with the value **CL_ERR_AD_TOO_MANY_POSTS**. The Camera Link API continued to capture the images and transfer the frames.

If the DMA transfer overflowed then **status** will be returned with the value **CL_ERR_AD_FRAME_OVERFLOW**. The Camera Link API continued to capture the images and transfer the frames. DMA transfer overflow can be caused by interruptions by the operating system to the DMA process.

If **CL_ERR_AD_TOO_MANY_POSTS** and **CL_ERR_AD_FRAME_OVERFLOW** both occurred then **status** will contain the value of the error that occurred last.

Transferred Frame Format

The transferred frame will be a greyscale image. Every four bytes of the transferred frame holds a pair of pixels' intensity values, a line indicator and a frame (field) indicator, therefore, a pixel is represented by two bytes. Each pair of pixels represents one even and one odd pixel on a line. The bit definition for each four bytes is:

Bit	Description
0	Even pixel data LSB
1	Even pixel data
2	Even pixel data
3	Even pixel data
4	Even pixel data
5	Even pixel data
6	Even pixel data
7	Even pixel data
8	Even pixel data
9	Even pixel data MSB
10	Not used
11	Not used
12	Not used
13	Not used
14	Not used
15	Not used
16	Odd pixel data LSB
17	Odd pixel data
18	Odd pixel data
19	Odd pixel data
20	Odd pixel data
21	Odd pixel data
22	Odd pixel data
23	Odd pixel data
24	Odd pixel data
25	Odd pixel data MSB
26	Not used
27	Not used
28	Not used
29	Not used
30	Line Counter 0 – even numbered line 1 – odd numbered line
31	Field Counter 0 – even numbered field 1 – odd numbered field

Using Bayer Interpolation or an alternative method the caller can convert the greyscale image into a colour image. The **bayerInterpolation** argument in the function **ciADSetRegionOfInterest** is not supported.