BitFlow SDK

Reference Manual

BitFlow, Inc.
400 West Cummings Park, Suite 5050
Woburn, MA 01801
USA
Tel: 781-932-2900
Fax: 781-933-9965
Email: support@bitflow.com
Web: www.bitflow.com
Revision G.8

Revision History:

Rev.	**Date**	**Details**
A.1	1996 -11-01	First Printing
A.2	1996-12-10	Format changes and corrections
B.1	1998-12-01	Second Printing
B.2	1999-01-16	Second Printing, minor changes
B.3	1999-05-30	Second Printing, minor changes
C.0	2001-06-01	Third Printing
C.1	2002-04-15	Formatting, corrections, updating to latest releases
D.0	2003-3-15	Ported to structured format. Typeface changes. Added R64 API.
E.0	2005-12-15	Formatting changes, synchronized with SDK 4.50
G.0	2008-06-01	Updates to synchronize with SDK 5.00
G.1	2009-12-01	Updates to synchronize with SDK 5.30
G.2	2010-11-19	Updates to synchronize with SDK 5.40
G.3	2011-12-16	Updates to synchronize with SDK 5.60
G.4	2012-07-19	Updates to synchronize with SDK 5.70
G.5	2014-10-01	Updates to synchronize with SDK 5.90
G.6	2016-05-17	Updates to synchronize with SDK 6.20
G.7	2018-07-01	Updates to synchronize with SDK 6.30
G.8	2020-01-07	Updates to synchronize with SDK 6.40

Table Of Contents

P - Preface

1 - SDK Introduction

1 - BufIn Introduction

2 - BufIn Board Functions

3 - BufIn Camera Functions

4 - BufIn Acquisition Functions

5 - BufIn Memory Functions

6 - BufIn Sequence Capture Management

7 - BufIn Circular Capture Management

8 - BufIn Trigger Functions

9 - BufIn Disk I/O Functions

10 - BufIn Status Functions

11 - BufIn Error Functions

12 - Camera Interface (Ci) Introduction

13 - Ci System Open and Initialization

14 - Ci Camera Configuration

15 - Ci Signal Functions

16 - Ci LUTs

17 - Ci Acquisition

18 - Ci Mid-Level Control Functions

19 - Ci Quad Table Functions

20 - Ci Control Tables

21 - Road Runner and R3 Introduction

22 - Road Runner/R3 System Open and Initialization

23 - Road Runner/R3 Acquisition

24 - Road Runner/R3 Camera Configuration

25 - Road Runner/R3 Interrupt Signals

26 - Road Runner/R3 Camera Control Functions

27 - Road Runner/R3 LUTS

28 - Road Runner/R3 Mid-Level Control Functions

29 - Road Runner/R3 Data Control Functions

30 - Road Runner/R3 Quad Table Functions

31 - Road Runner/R3 Register Access

32 - Road Runner/R3 Control Tables

33 - Road Runner Quad Tables

34 - Road Runner/R3 Error Handling

35 - R64 Introduction

36 - R64 System Open and Initialization

37 - R64 Acquisition

38 - R64 Camera Configuration

39 - R64 Interrupt Signals

40 - R64 Quad Table Functions

41 - R64 Mid-Level Control Functions

42 - R64 Control Functions

43 - R64 Control Tables

44 - R64 Dual Port Memory

45 - Camera Link Specification Serial Interface

46 - Display Functions

47 - BitFlow Common Functions Introduction

48 - CoaXPress specific functions

49 - BitFlow Error Handling

50 - BitFlow Register Access

51 - BitFlow Version Control Functions

52 - BitFlow Miscellaneous Functions

53 - BitFlow Disk I/O Functions

54 - BitFlow Types

Preface

Chapter P

P.1 Purpose

This Software Reference Manual is intended for anyone using the BitFlow Software Development Kit (BitFlow SDK). This manual is primarily meant as a reference manual for users writing their own applications. It is not intended to be a users' manual for the utilities and examples provided with the BitFlow SDK.

This preface explains the various layers and Application Programming Interfaces (APIs) of the BitFlow SDK. Please read the following section to determine which API best suits your needs.

The BitFlow SDK was first release in 1996 and has been updated continuously every since. While the main purpose of the API, getting images into user memory, has not changed over the years, the features and ease of use has evolved considerably. In addition, the SDK originally only support one product family, not is supports seven. BitFlow has made every effort to keep backwards compatible with each new release, while adding new powerful features and support for new product families. This has been a challenge, and might prompt one to ask why bother, as the API might seem large and unwieldy. The reason is that most of our customers build our product into machines that need to be support for many many years. Our goal is to support these long term customers, while adding features that will convince new customers are easy to use and robust enough for long term industrial use.

P.2 The History of the BitFlow APIs

The BitFlow SDK originally had two APIs, the high level R2 API and the low level BF API. This supported our one product family, the Road Runner. When we released our Raven family the architecture was substantially different and required its own API, so the Rv API was released at the same time. The R3 family came next, but it was identical in architecture to the Road Runner so no new API was needed. Soon we built a new API on top of these two call the Ci API, which would work with any board installed in the system. The idea was to let customers write one application using the Ci API, and be able to seamlessly move back and forth between product families. We also added this API with an eye to the future, promising support for all new product families under the Ci API.

When the R64 family was released, it again has a very different architecture, so a new API was added as well, the R64 API. It became clear at this point the customers did not and should not care about product family when writing their applications, so we began de-emphasizing the individual APIs (R3, Rv and R64), and started promoting the Ci API exclusively.

At the same time we realized the many of our customers were writing the same type of applications using our functions. We decided to save the customers the trouble, by encapsulating the most common functions in a new buffer management API, the Bi API. This new API supported both sequence capture and circular buffer management with a few simple function calls. Since the Bi API was built on the Ci API, it was automatically board family independent.

As of the writing of this manual, we are on our 6th full point release of the SDK, and some of the older product families are going into end-of-life status. Supported APIs for these older boards begins to make less sense. For this reason, we are encouraging all new applications to be built on the Ci and/or Bi APIs . Of course, for low level access, the BF API will always be there. While all of the older APIs are documented in this manual, future support is not guaranteed. The Bi/Ci/BF APIs offer all the functionality provided by these older APIs, plus a lot more.

P.3 The APIs

This SDK works with all of BitFlow's current camera interface products: the Road Runner, the R3, the R64, the Karbon, the Neon and the Alta.. The SDK consists of six APIs:

Road Runner (prefix R2) - Road Runner/R3 family
R64 (prefix R64) - R64, Karbon, Neon and Alta families
Gen 2 (prefix Gn2) - Aon, Axion and Cyton families
Camera Interface (prefix Ci) - All families
Buffer Management, BufIn (prefix Bi) - All families
Low level access (prefix BF) - All families

The Road Runner and the R3 are so similar that they share the same API. The R64, R64e, Karbon, Neon and Alta are all based on the original R64 architecture so they can all be programmed via the R64 API. The Gen2 API is covers the latest families: Aon, Axion and Cyton. There are two generic high level APIs, called the Camera Interface API (Ci functions) and BufIn (Bi functions), which are designed to work with any of the BitFlow products. Applications calling the Ci or Bi layer need not worry about which type of board is installed.

At the lowest level there is a common layer, called the BitFlow API (BF functions). Most applications can be written using only the high level APIs, however, occasionally low level access is needed, thus the BF API is available. The Figure P-1 diagram illustrates the organization of these layers.

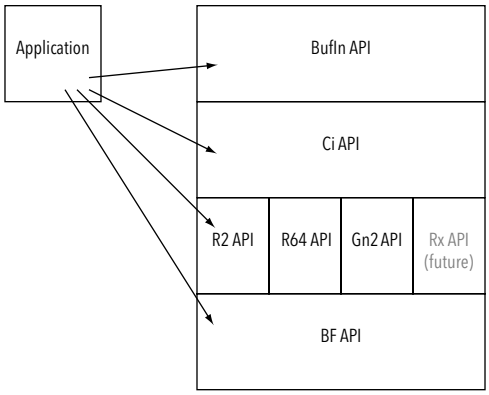

Figure P-1 BitFlow SDK Layers

The concept is that each layer calls the layer below it and that some API works with one family of boards while other APIs work with all boards. However, the functionality available with each API is different. For example, if a Neon is installed in a system: Bi, Ci, R64 and BF APIs can all be used, but the R2 API can not (although there would be no need for this API). Writing a complete application using only the BF API would be extremely difficult, its role is mainly as a low level foundation for the other functions to

build upon. If a Raven is installed, you could use either the R64 API (not recommended) or the Ci API, the functionality is roughly equivalent. The Bi API can be used to add buffer management. However, an application written with the R64 API will only work when Karbons or Neons are installed, but an application written using the Ci or Bi APIs will work with any family of board in the system, including families released in the future.

In addition, the R2, R64 and Gn2 layers are broken into separate tiers. Most applications can be written using only the highest level functions. However a mid-level function may be required to make a small tweak to the board setup. The lowest level is basically used for direct access to the memory and registers on the board.

P.4 Which API Should I Use?

As mention in previous section, all new applications should be written with the Ci API or the Bi API. We are discouraging use of the R2, R64 and Gn2 APIs, and support of these APIs is not guaranteed in future releases. The BF API can be used if low level access is required.

The following table should help you choose the correct API for your application.

Application	API	SDK Example Application
The application only needs to acquire into one buffer at a time.	Ci	CiView - live video display application CiSimple - Super simple console application
The application will need to acquire into multiple buffers continuously.	Bi - Circular functions	Circ -Continuous circular buffer acquisition and display application Circular.c - Simple circular acquisition console application
The application needs to acquire sequences of images to memory.	Bi - Sequence functions	BiFlow - Sequence capture/ display/save appliction Sequence.c - Simple sequence capture example
The application needs only low level access to the board's registers.	BF	

P.5 Organization

The manual is partitioned into eight books corresponding to the major APIs as follows:

BufIn functions (Bi)
Camera Interface Functions (Ci)
Road Runner and R3 Functions (R2)
R64 functions (R64)
Camera Link Specification Serial Port Functions (cl)
Display Functions (Disp)
BitFlow Functions(BF)

All of the books are covered by the same Table of Contents at the beginning of the manual and the same Index at the end.

Note: The Gen 2 functions are not documented. While the Gn2 API exists and could potentially be used, there is no added functionality over the Ci API. Using the Gn2 API also restricts usage to only Gen 2 boards.

P.6 Support Services

BitFlow provides both sales and technical support for the all of our hardware and software products.

P.6.1 Technical Support

Our web site is www.bitflow.com.

Technical support is available at 781-932-2900 from 9:00 AM to 6:00 PM Eastern Standard Time, Monday through Friday.

For technical support by email (support@bitflow.com) please include the following:

Product name
Camera type and mode being used
Software revision number
Computer CPU type, PCI chipset, bus speed
Operating system
Example code (if applicable)

P.6.2 Sales Support

Contact your local BitFlow Sales Representative, Dealer, or Distributor for information about how BitFlow can help you solve your most demanding camera interfacing problems. Refer to the BitFlow, Inc. website (www.bitflow.com) for a list of sales representatives.

SDK Introduction

Chapter 1

1.1 Overview

The BitFlow Software Development Kit (SDK) consists of a kernel driver, Dynamic Link Libraries (DLLs), example applications with source code and utilities for all of BitFlow's current frame grabbers. The applications included are quite powerful and can be used "out of the box" to acquire and save images and sequences. The applications can also be used to setup and test cameras, to modify a camera's modes and to create corresponding camera configuration files. However, the real purpose of the SDK is to provide a platform for the customer to quickly and easily build their imaging application. The goal of our Application Programming Interface (API) is make control over the camera, image acquisition, image buffer management and image storage simple and painless. At the same time, the goal is to use as little of the computers resources as possible. BitFlow has gone through enormous efforts to anticipate customer needs and incorporate them into this SDK.

The boards are accessed through two main APIs: Bi (BufIn) and Ci functions. The BufIn API is a very high level buffer management layer, and works with all boards. The BufIn API is accessible from C, C++ and C#. The Ci API is also board independent and lets an application work with which ever board is plugged into the system, no recompiling is required when the board type changes. There are also two legacy APIs, R2 and R64. These APIs only allow access to their corresponding family, and it is recommended that these functions not be used. Finally there is a low level hardware access API, BF, which is mainly used for direct control of memory and registers.

1.2 Camera Configuration Files

All BitFlow frame grabbers are initialized with a camera configuration file when the board is first opened in software (via CiBrdOpen, BiBrdOpen, etc.). The camera configuration file sets the board up for interfacing with a particular camera. The BitFlow SDK comes with over a 1000 camera files. There were generally constructed when BitFlow's engineers interfaced and tested the camera directly in the BitFlow labs. If a camera configuration file for your camera does not exists in the SDK, please contact BitFlow customer support and they will get you the file that you need.

There are a number of different ways to specify which camera configuration file should be used. These different methods are explained in the following sections. However, the simplest is to run the utility SysReg and "attach" a camera file to a particular board in your system. Once this is done, all BitFlow applications will configure the board to use this file.

Each family of BitFlow frame grabber family has its own camera configuration file format. The formats are as follows:

- The Axion-CL - this family uses XML camera files with the extension "bfml"
- The Aon-CXP/Cyton-CXP - this family uses XML camera files with the extension "bfml"
- The Karbon-CXP - this family uses files with the extension "kcxp"
- The Alta - this family uses files with the extension "anlg"
- The R64/Karbon/Neon - this family uses files with the extension ".r64". All models of R64 use this camera configuration file format.
- The Road Runner/R3 Camera Link models - this family uses files with the extension ".rcl".
- The Road Runner/R3 Differential models - this family uses files with the extension ".cam". There is a camera configuration cross reference located in the file "BitFlow SDK X.XX\Docs\Camera File List.txt".

Note that the Cyton camera configuration files are XML files. This is a change from all previous BitFlow camera configuration files, which were a proprietary format. These BFML files can be edited in any text editor. Their schema is documented on the download page of BitFlow's web site. However, since SDK 6.30, the SDK comes with a dedicated BFML file editor called CamML, which makes modifying these files very easy.

1.3 Specifying Camera Configuration Files

A camera configuration file is used to initialize a board to work with a specific camera in a specific mode. When a board is opened and initialized, a camera configuration file must be specified. There area a few different methods to specify a camera configuration file. The following subsections enumerate these methods.

1.3.1 Camera Configuration File Specified Via SysReg

This is by far the most common method to specify a camera configuration file. Each board installed in the system is listed in SysReg. A camera configuration file can be "attached" to each board in SysReg's Board Details dialog. When a board is opened via CiBrdOpen or BiBrdOpen, the camera file specified in SysReg is used to configure the board.

1.3.2 Specifying Multiple Camera Configuration Files in SysReg

Multiple camera configuration files can be "attached" to a single board in SysReg. It is then possible to switch between which camera file the board is currently initialized to via the functions CIBrdCamSel or BiCamSel. The current configuration is selected using an index which corresponds to the list of camera file specified for the board in SysReg.

1.3.3 Camera Configuration File Specified Via a Board Open Function

The camera configuration file can be specified in the board open function. There are a few function that support this method:

CiBrdOpenCam
BiBrdOpenCam
BiBrdOpenCamEx

When this method is used, the camera configuration file specified in SysReg is ignored. However, it is still suggested that a default camera file be specified in SysReg as many applications still rely on this method.

1.3.4 Specifying the Camera Configuration File After the Board is Opened

Once the board is opened and initialized, the board can be reconfigured with a different camera file by using the CiCamOpen/BiBrdOpen and CiBrdCamSetCur/BiBrdSetCur. These functions allow for full control of which camera configuration file the board is currently initialized to. The configuration file can be change as often as needed. However, the camera configuration can only be change when acquisition is not set up.

1.3.5 Specifying a Default Camera Configuration File in the Registry

It is possible to specify a default camera configuration file in the registry by creating the following registry value:

Got to the key:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\BitFlow

Create a new string value:

Value Name = DefaultR64File
Value Data = CameraFileName.r64

The camera file set above in the registry is only used if the camera configuration file is not specified in SysReg. If there is camera file specified in SysReg, it will be used.

The Value Name above is different depending on the major family type of the board. See Table 1-1 for the Value Name needed for each family. More than one entry can be used to support multiple families on the same PC.

Table 1-1 Registry Value Name for Each Family

Models	Value Name
R64, Karbon, Neon	DefaultR64File
Alta	DefaultAnlgFile
Road Runner/R3 Differential	DefaultCamFile
Road Runner/R3 Camera Link	DefaultRclFile
Karbon CXP	DefaultKcxpFile
Cyton CXP	DefaultCtnFile
Aon CXP	DefaultAonFile
Axion CL	DefaultAxnFile
Claxon CXP	DefaultClxFile

1.4 SDK Utilities

The BitFlow SDK includes a number of utilities to facilitate the development and deployment of BitFlow applications. The following table briefly lists the more important utilities shipped with the SDK.

Table 1-2 BitFlow SDK Utilities

Name	Purpose
AxionStats	Tool for debugging Camera Link interfaces (Gen 2 only)
BFDX	Low level hardware debugging tool
BFCom	Communications terminal for sending/receiving serial commands from Camera Link cameras
BFLog	Debugging and logging utiltiy
BitFlowCapture	Capture sequence of image to host memory (and save them offline to disk)
BitFlowPreview	Live video preview (also save captured images to disk)
CamEd	Edit *.r64 camera configuration files
CamML	Edit *.bfml camera configuration files
CamVert	Low level binary format camera configuration file editor
CXPRegTool	Command line tool to send/receive commands from CoaXPress cameras
CytonStats	Tool for debugging CoaXPress interfaces
PCIWalk	Enumerates BitFlow boards installed in the system along with PCI space details
SysReg	Set system wide settings, Attach camera configuration files to each board
RegTool	Command line tool to read/write board registers
VerCheck	Collect system information that can be useful in debugging system issues.
Ximilon	Camera configuration utility for GenICam based cameras (mainly CoaXPress cameras).

1.5 SDK Example Applications

The BitFlow SDK includes a number of example applications that can be used to help understand the how the BitFlow API works. Source code for all of these applications is included. Some examples are capable on there own to be used in basic applications. Many of the examples are console base (i.e. command line applications), these make it very easy to understand how to work with the BitFlow API.

Examples are located in the "\BitFlow SDK X.XX\Examples" folder. Almost every aspect of the BitFlow API is illustrated in at least one example.

1.6 Support for Other Languages

The BitFlow SDK is primarily written for C/C++ applications. However, it supports other environments as well. The follow are supported in one way or another:

C++ - Any C API can be called from with a C++ class. However, the BitFlow SDK also offers a set of C++ classes which offer most of the high level functionality of the C API. There are number of C++ examples included in the SDK and the documentation is available on our web side.

C# - A separate .NET interface is available for download from our web site. This, like our C++ interface, offers most of the high level functionality of our C API. Documentation and examples are also provided in the .NET download.

VB.NET - The .NET interface also supports VB.NET.

BufIn Introduction

Chapter 1

1.1 Overview

The primary purpose of the Buffer Interface API, (BufIn) is to provide an easier means of using the advanced acquisition functionality of BitFlow's products. The BufIn API simplifies the programming effort by significantly reducing the number of function calls needed to develop two of the most fundamental components of an imaging application, sequence capture and circular buffer management.

A developer that is simply capturing a predetermined sequence of frames or lines relative to an event trigger, and is not required to process or view the image data in real-time, may want to use sequence capture within their application. For sequence capture, an array of buffers are allocated and the buffers are then filled with image data automatically, starting at the first buffer and ending at the last buffer. The user has the ability to specify the start and end buffers with the only limitation on the number of buffers allocated, being the amount of memory installed in the computer system. Once the sequence of images are in memory, they can be viewed or saved to disk.

A developer requiring real-time processing of image data may choose to use circular buffer management. For circular buffer management, an array of buffers is allocated and the buffers are filled with image data starting from the first buffer, filling the rest of the buffers in a round- robin fashion. When the end of the array is reached, the first buffer will be overwritten with new data. Unlike sequence management, where acquisition is stopped after the last buffer is filled, circular management will continue to acquire image data and overwrite previous data, until the user specifies acquisition to end. The idea behind circular buffer management is that a buffer can be processed by the CPU while BitFlow's board acquires into one of the other buffers. This separation between acquisition and processing used by BufIn is helpful in situations where the processing time is not constant. Buffer processing time can vary considerably, as long as the average processing time is equal to or less than the acquisition time, the number of buffers will not need to be increased to keep acquisition from catching up to the buffer being processed. If the average processing time where to increase for any reason, (i.e. an increased swing of processing times of the buffers), more buffers would need to be required to keep acquisition from catching up to the buffer being processed. In the ideal situation, enough buffers will be allocated to keep the acquisition of the image data from catching up to the buffer being processed, but this is not always the case. Therefore, BufIn provides a means of marking buffers so that they will not be overwritten. Similarly to sequence management, the buffers can be viewed or saved to disk as a BMP, TIFF, AVI or RAW file formats.

The BufIn API supports every model of every BitFlow product and will also support future BitFlow camera interface boards.

The BufIn API can support any size image that is supported by the particular BitFlow interface board. The BufIn API supports 8, 10, 12, 24, 32, 36, 42 and 48-bit data formats and any image formats that are supported by the camera interface card (two taps, reverse scan, etc.).

The BufIn API supports it's own simplified error handling and viewing functions. The API also provides the ability to internally handle errors such as FIFO overflows, buffers being overwritten, and hardware exceptions.

The internal timeout values that Bufin uses comes from the camera file for non-triggered modes such as free run. For any triggered modes, (one-shot, start/stop), the timeout value used is INFINITE. Using INFINTE as a timeout value causes the acquisition engine to wait forever for a frame to be acquired. In non-triggered modes the timeout value can be adjusted by adjusting the timeout value in the camera file.

Sequence capture and circular buffer management applications can be developed solely with the BufIn API. The API provides all functions needed to open the board, setup acquisition, control acquisition, handle errors, clean up and close the board. No additional BitFlow function calls should be needed. Examples in the SDK have been provided to show how to use the BufIn API for sequence capture and circular buffer management. For comparison purposes, the sequence capture example BiFlow uses the BufIn API where the example application Flow dose not use the BufIn API. For examples of the circular management functions refer to example applications Circ and BiProcess.

A normal program would use the following sequence of functions for sequence capture:

```
Bd   Board;
BIBA BufArray;

// Open the first R64 family board in the system.
BiBrdOpen(BiTypeR64,  0, hBoard);

// Allocate 25 buffers, use camera file information for
// image size and bit depth.
BiBufferAllocCam(hBoard, &BufArray, 25);

// Setup sequence acquisition using Host Qtabs and
// DMA Engine J.
BiSeqAqSetup(hBoard, &BufArray, BiAqEngJ);

// Start image acquisition asyncronously.
BiSeqControl(hBoard, &BufArray, BISTART, BiAsync);

// Wait for 5 seconds for acquisition to complete.
// (This can be placed in a separate thread that will
//  wake up when acquistion is complete.)
BiSeqWaitDone(hBoard, &BufArray, 5000);

// At this point the image data is in memory. Now it
// can be viewed, processed and/or saved to disk.

// Save the sequence of images starting a 5 through 15,
// to disk as a TIFF. Call the files SeqDemo.
BiDiskBufWrite(hBoard, &BufArray, BITIF, 5, 10, SeqDemo, 0);
```

```
// Clean up sequence acquisiton.
BiSeqCleanUp(hBoard, &BufArray);

// Free buffer memory that has been allocated.
BiBufferFree(hBoard, &BufArray);

// Close the board
BiBrdClose(hBoard);
```

A normal program would use the following sequence of functions for circular capture:

```
Bd   Board;
BIBA BufArray;

// Open the first R64 family in the system.
BiBrdOpen(BiTypeR64,  0, hBoard);

// Allocate 25 buffers, use camera file information for
// image size and bit depth.
BiBufferAllocCam(hBoard, &BufArray, 25);

// Setup circular acquisition using Host Qtabs and
// DMA Engine J.
BiCircAqSetup(hBoard, &BufArray, BiAqEngJ);

// Start image acquisition asyncronously.
BiSeqControl(hBoard, &BufArray, BISTART, BiAsync);
// Loop here until and display the image data to
// the screen, until acquisition is stopped.
while(ACQUISITION IS STILL RUNNING)
{
   BiCirWaitDoneFrame(hBoard, &BufArray, INFINITE,
&CirHandle);

   // DISPLAY IMAGE DATA.
}

// Save all images to disk as BMP. Call the files CircDemo.
BiDiskBufWrite(hBoard, &BufArray, BIBMP, 0, 25, CirDemo, 0);

// Acquisiton has been stopped.
// Clean up circular acquisiton.
BiSeqCleanUp(hBoard, &BufArray);

// Free buffer memory that has been allocated.
BiBufferFree(hBoard, &BufArray);

// Close the board
```

```
BiBrdClose(hBoard);
```

BufIn Board Functions

Chapter 2

2.1 Introduction

The functions described in this chapter are quite simple. The idea being to open the board for acquisition. When acquisition is complete, close the board, thus cleaning up all resources allocated in the open function.

A normal program would uses these functions, in this order:

```
BiBrdOpen

// acquisition and processing

BiBrdClose
```

If you want to open two boards, the flow would be as follows:

```
BiBrdOpen   // open board 0
BiBrdOpen   // open board 1

// acquisition and processing

BiBrdClose   // close board 0
BiBrdClose   // close board 1
```

The handle return by the function BiBrdOpen is used in all subsequent function calls. If you are using two or more boards, open each board and store each handle in a separate variable. Whenever you want to talk to board X, pass the handle for board X to the function.

There is no need to call BiBrdOpen more than once per process per board. Because this function takes a fair amount of CPU time and allocated resources, we discourage users from repeatedly calling BiBrdOpen and the BiBrdClose in a loop. We recommend opening the board once, when the application starts, and closing it once when the application exits. If you are using a program that has multiple threads, open the board once in the first main thread and then pass the board handle to every thread that is subsequently created. You must call BiBrdClose for every board that is open with BiBrdOpen. You should also call BiBrdClose in the same thread that BiBrdOpen was called.

2.2 BiBrdOpen

Prototype BIRC BiBrdOpen(BFU32 *BrdType*, BFU32 *BrdNumber*, Bd **pBrdHandle*)

Description Opens a board for access. This function must return successfully before any other BI functions are called.

Parameters

BrdType

Type of board to open. The types of boards to open are as follows:

BiTypeR2 - RoadRunner or R3 type board.
BiTypeR64 - R64, R64e, Karbon, Neon or Alta type board.
BiTypeGn2 - Aon, Axion, Cyton or Claxon type board.
BiTypeAny - Opens boards by number, ignoring the board type.

BrdNumber

Specifies the board number to open. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots as long as the number of boards is the same and the boards are in the same PCI slots.

pBrdHandle

A pointer to the board handle after successfully opening a board. This handle is used for all further accesses to the newly opened board.

Returns

BI_OK	A board was found and opened.
BI_ERROR_BOARD_NOT_FOUND	There is no board with this number.
BI_ERROR_UNKNOWN_TYPE	The board type specified by BrdType is unknown.
BI_ERROR_SYSTEM	A error occurred while searching for the board information in the registry.
BI_ERROR_OPENING	Board was found but could not be opened.

Comments This function opens the board for all accesses. The board must be opened before any other functions can be called. When you are finished accessing the board you must call BiBrdClose, before exiting your process. Failure to call BiBrdClose will result in incorrect board open counts used by the driver.

When using BiTypeAny for *BrdType*, the board is opened only using the board number. For instance, if board 0 is a R2, board 1 is a R3 and board 2 is an R64 and the following function calls are made with the following results:

```
BiBrdOpen(BiTypeAny, 0, &hBoard); // Opens the R2
BiBrdOpen(BiTypeAny, 1, &hBoard); // Opens the R3
BiBrdOpen(BiTypeAny, 2, &hBoard); // Opens the R64
```

If BiBrdOpen fails, you cannot access the board, and you do not need to call BiBrd-Close.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board. You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call BiBrdClose first.

2.3 BiBrdOpenEx

Prototype BIRC BiBrdOpenEx(BFU32 *BrdType*, BFU32 *BrdNumber*, Bd **pBrdHandle*, BFU32 *Options*)

Description Opens a board for access. This function must return successfully before any other BI functions are called.

Parameters

BrdType

Type of board to open. The types of boards to open are as follows:

BiTypeR2 - RoadRunner or R3 type board.
BiTypeR64 - R64, R64e, Karbon, Neon or Alta type board.
BiTypeGn2 - Aon, Axion, Cyton or Claxon type board.
BiTypeAny - Opens boards by number, ignoring the board type.

BrdNumber

Specifies the board number to open. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots as long as the number of boards is the same and the boards are in the same PCI slots.

**pBrdHandle*

A pointer to the board handle after successfully opening a board. This handle is used for all further accesses to the newly opened board.

**Options*

Special board open options. Can be one or more of the following:

BFSysInitialize - Initialize the system.
BFSysExclusive - If not already open, open exclusively.
BFSysNoIntThread - Do not activate interrupt thread.
BFSysNoCameraOpen - Do not open any camera configurations.
BFSysNoAlreadyOpenMess - Supress already open warning message.
BFSysNoOpenErrorMess - Supress all error popups in open function.
BFSysSecondProcessOpen - Allow the board to be opened twice in the same process (includes some of the above modes).
BFSysAllowTwoOpens - Allow the board to be opened twic ein the same process, and initialized.
BFSysNoPoCLChange - This flag forces the system to leave the PoCL system as is (don't change its state).
BFSysPoCLUpOnly - This flag will power up PoCL if it is off, but won't turn it off, if it is on.
BFSysSerialPortOpen - used when opening the serial port, included some of the above flags
BFSysNoCXPInit - Don't initialize the CXP subsystem

BFSysNoGenTLInit - Don't use GenTL camera control during board initialization.
BFSysNoIOReset - Do not reset I/O outputs before setting them as per configuration file

Returns

BI_OK	A board was found and opened.
BI_ERROR_BOARD_NOT_FOUND	There is no board with this number.
BI_ERROR_UNKNOWN_TYPE	The board type specified by BrdType is unknown.
BI_ERROR_SYSTEM	A error occurred while searching for the board information in the registry.
BI_ERROR_OPENING	Board was found but could not be opened.

Comments

This function works exactly like BiBrdOpen except that it provides for more options for opening the board. The options provided here are the exact same as the options support for the function CiBrdOpen.

When BiBrdOpen is used, the default options are used. In this case BiBrdOpen opens the board with the option BFSysInitialize. When using BiBrdOpenEx, it is recommend that you use at least the option BFSysInitialize as well as other options as needed.

2.4 BiBrdOpenCam

Prototype BIRC BiBrdOpenCam(BFU32 *BrdType*, BFU32 *BrdNumber*, Bd **pBrdHandle*, PBFCHAR *ForceCamFile*)

Description Opens a board for access and opens the given camera file. This function must return successfully before any other BI functions are called.

Parameters

BrdType

Type of board to open. The types of boards to open are as follows:

BiTypeR2 - RoadRunner or R3 type board.
BiTypeR64 - R64, R64e, Karbon, Neon or Alta type board.
BiTypeGn2 - Aon, Axion, Cyton or Claxon type board.
BiTypeAny - Opens boards by number, ignoring the board type.

BrdNumber

Specifies the board number to open. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots as long as the number of boards is the same and the boards are in the same PCI slots.

**pBrdHandle*

A pointer to the board handle after successfully opening a board. This handle is used for all further accesses to the newly opened board.

ForceCamFile

The camera file to open. The camera file should include the name and the file extension. If only the file name and extension are given, the camera configuration path is searched for the camera file. (The camera configuration path by default is the Config folder under the SDK root.) If the full path is given, the camera file will try and be opened from that location.

Returns

BI_OK	A board was found and opened.
BI_ERROR_BOARD_NOT_FOUND	There is no board with this number.
BI_ERROR_UNKNOWN_TYPE	The board type specified by BrdType is unknown.
BI_ERROR_SYSTEM	A error occurred while searching for the board information in the registry.
BI_ERROR_OPENING	Board was found but could not be opened.

Comments

This function opens the board for all accesses. The board must be opened before any other functions can be called. When you are finished accessing the board you must call BiBrdClose, before exiting your process. Failure to call BiBrdClose will result in incorrect board open counts used by the driver.

When using BiTypeAny for *BrdType*, the board is opened only using the board number. For instance, if board 0 is a R2, board 1 is a R3 and board 2 is an R64 and the following function calls are made with the following results:

```
// Opens the R2
R2BiBrdOpen(BiTypeAny, 0, &hBoard, "SomeCamFile.cam");

// Opens the R3
BiBrdOpen(BiTypeAny, 1, &hBoard, "SomeCamFile.rvc");

// Opens the R64
BiBrdOpen(BiTypeAny, 2, &hBoard, "SomeCamFile.r64");
```

If BiBrdOpenCam fails, you cannot access the board, and you do not need to call BiBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board. You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call BiBrdClose first.

2.5 BiBrdOpenCamEx

Prototype BIRC BiBrdOpenCamEx(BFU32 *BrdType*, BFU32 *BrdNumber*, Bd **pBrdHandle*, PBFCHAR *ForceCamFile*, BFU32 *Options*)

Description Opens a board for access and opens the given camera file. This function must return successfully before any other BI functions are called. This open function supports a number of options.

Parameters

BrdType

Type of board to open. The types of boards to open are as follows:

BiTypeR2 - RoadRunner or R3 type board.
BiTypeR64 - R64, R64e, Karbon, Neon or Alta type board.
BiTypeGn2 - Aon, Axion, Cyton or Claxon type board.
BiTypeAny - Opens boards by number, ignoring the board type.

BrdNumber

Specifies the board number to open. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots as long as the number of boards is the same and the boards are in the same PCI slots.

**pBrdHandle*

A pointer to the board handle after successfully opening a board. This handle is used for all further accesses to the newly opened board.

ForceCamFile

The camera file to open. The camera file should include the name and the file extention. If only the file name and extention are given, the camera configuration path is searched for the camera file. (The camera configuration path by default is the Config folder under the SDK root.) If the full path is given, the camera file will try and be opened from that location.

**Options*

Special board open options. Can be one or more of the following:

BFSysInitialize - Initialize the system.
BFSysExclusive - If not already open, open exclusively.
BFSysNoIntThread - Do not activate interrupt thread.
BFSysNoCameraOpen - Do not open any camera configurations.
BFSysNoAlreadyOpenMess - Supress already open warning message.
BFSysNoOpenErrorMess - Supress all error popups in open function.
BFSysSecondProcessOpen - Allow the board to be opened twice in the same process (includes some of the above modes).
BFSysAllowTwoOpens - Allow the board to be opened twic ein the same

process, and initialized.
BFSysNoPoCLChange - This flag forces the system to leave the PoCL system as is (don't change its state).
BFSysPoCLUpOnly - This flag will power up PoCL if it is off, but won't turn it off, if it is on.
BFSysSerialPortOpen - used when opening the serial port, included some of the above flags
BFSysNoCXPInit - Don't initialize the CXP subsystem
BFSysNoGenTLInit - Don't use GenTL camera control during board initialization.
BFSysNoIOReset - Do not reset I/O outputs before setting them as per configuration file

Returns

BI_OK	A board was found and opened.
BI_ERROR_BOARD_NOT_FOUND	There is no board with this number.
BI_ERROR_UNKNOWN_TYPE	The board type specified by BrdType is unknown.
BI_ERROR_SYSTEM	A error occurred while searching for the board information in the registry.
BI_ERROR_OPENING	Board was found but could not be opened.

Comments

This function opens the board for all accesses. The board must be opened before any other functions can be called. When you are finished accessing the board you must call BiBrdClose, before exiting your process. Failure to call BiBrdClose will result in incorrect board open counts used by the driver.

If BiBrdOpenCam fails, you cannot access the board, and you do not need to call BiBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board. You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call BiBrdClose first.

When BiBrdOpen is used, the default options are used. In this case BiBrdOpen opens the board with the option BFSysInitialize. When using BiBrdOpenCamEx, it is recommend that you use at least the option BFSysInitialize as well as other options as needed.

2.6 BiBrdOpenSWConnector

Prototype BIRC BiBrdOpenSWConnector(BFU32 *BrdType,* BFU32 *Switch*, BFU32 *Connector*, Bd **pBrdHandle*)

Description Opens a board with the given switch value and connector number for access. This function must return successfully before any other BI functions are called.

Parameters

BrdType

Type of board to open. The types of boards to open are as follows:

BiTypeR2 - RoadRunner or R3 type board.
BiTypeR64 - R64, R64e, Karbon, Neon or Alta type board.
BiTypeGn2 - Aon, Axion, Cyton or Claxon type board.
BiTypeAny - Opens boards by number, ignoring the board type.

Switch

Specifies the switch setting of the board that you wish to open. The swith is a small mechanical switch that is mounted on the upper edge of the board. See that hardware reference manual for more details on location. The acceptable values are 0 to 3.

Connector

Specifies the connector number of the board that you wish to open. This parameter is only for use for boards that have more than one Virtual Frame Grabber (VFG). For boards with only one VFG, this value must be 1. Connector numbers start with 1, as per the hardware manual. For example, the Neon-CLQ has connectors CL1, CL2, CL3 and CL4. Therefore, to open CL2, this parameter must be set to two.

**pBrdHandle*

A pointer to the board handle after successfully opening a board. This handle is used for all further accesses to the newly opened board.

Returns

BI_OK	A board was found and opened.
BI_ERROR_BOARD_NOT_FOUND	There is no board with this number.
BI_ERROR_UNKNOWN_TYPE	The board type specified by BrdType is unknown.
BI_ERROR_SYSTEM	A error occurred while searching for the board information in the registry.
BI_ERROR_OPENING	Board was found but could not be opened.
BFSYS_ERROR_NOTFOUND	No board was found with the given Switch and Connector values.

Comments

This function opens the board for all accesses. The board must be opened before any other functions can be called. When you are finished accessing the board you must call BiBrdClose, before exiting your process. Failure to call BiBrdClose will result in incorrect board open counts used by the driver.

If BiBrdOpenSWConnect fails, you cannot access the board, and you do not need to call BiBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board. You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call BiBrdClose first.

This function provides and alternate way to open up a board. In a system with more than one board and/or when a board has more than one Virtual Frame Grabber(VFG), this function can make opening the desired board much easier.

For example, if a system has two NEO-PCE-CLQ boards installed, there are actually 8 VFGs in the system. This means opening the exact board needed can be somewhat complicated, as the normal board open functions take a board number, in this case 0 to 8. It can be complicated correlating this board number with the connectors in the back of the PC. This function is designed to handle this case. Make sure to set the switch settings of each board differently (up to four boards can be differentiated). Then set the *Connector* value to the CL connector of the desired board (and camera).

Let's look closer at this example. Say the system has two NEO-PCE-CLQ boards installed, one has its switch set to 0 and the other to 1. Let's say that we want to open the camera connected to the CL4 connector of the board that has its switch set to 1. The following function call should be made:

```
BiBrdOpenSWConnector(BiTypeAny, 1, 4, &hBoard);
```

2.7 BiBrdInquire

Prototype BIRC BiBrdInquire(Bd Board, BFU32 InquireVar, PBFU32 pVal)

Description Used to inquire the system characteristics of the board. Can also be called with CiCamInquire members which are then passed to that function using the current camera.

Parameters *Board*

Board to handle.

InquireVar

Parameter to inquire about:

BiBrdInqModel - returns the board model. The parameter *pVal* will point to one of:

BFBrdValUnknown
BFBrdValModel11
BFBrdValModel12
BFBrdValModel13
BFBrdValModel14
BFBrdValModel23
BFBrdValModel24
BFBrdValModel44
BFBrdValModel010
BFBrdValModel110
BFBrdValModel220
BFBrdValModel330
BFBrdValModel440
BFBrdValModelR64Dif
BFBrdValModelR64Cl
BFBrdValModelR64DifB
BFBrdValModelR64ClB
BFBrdValModelR64DifH
BFBrdValModelR64ClH
BFBrdValModelR64DifHB
BFBrdValModelR64ClHB

BiBrdInqSpeed - returns the board receivers speed for the RoadRunner. The parameter *pVal* will point to one of:

BFBrdValSpeedNormalR2
BFBrdValSpeed40MHzR2

BiBrdInqLUT - the type of LUT mounted on the board. The parameter *pVal* will point to one of:

BFBrdValLUT8And12

BFBrdValLUT16
BFBrdValLUTNone

BiBrdInqIDReg - the current setting of the ID switch on the board (0,1,2,3).

BiBrdInqScanType - returns the scan type for the Raven only. The parameter *pVal* will point to one of:

BFBrdValStandard - board will only work with standard scan cameras.
BFBrdValVariable - board will work with variable scan cameras and standard scan cameras.

BiBrdInqAnalogType - returns the type of analog video input the Raven is setup for. The parameter *pVal* will point to one of:

BFBrdValDifferential - the Raven has differential video input.
BFBrdBalSingle - the Raven has single ended video input.

BiBrdInqNumCams - returns the number of cameras the Raven is configured for.

Camera inquiry parameters are also valid. The *pVal* parameter will point to the value for the board's current camera. The *InquireVar* must be one of the following:

BiCamInqXSize - width of image in pixels.
BiCamInqYSize0 - camera 0 height of image in lines.
BiCamInqYSize1 - camera 1 height of image in lines. (for Raven use only)
BiCamInqYSize2 - camera 2 height of image in lines.(for Raven use only)
BiCamInqYSize3 - camera 3 height of image in lines.(for Raven use only)
BiCamInqFormat - image format.
BiCamInqBitsPerPix - depth of pixel in bits, as acquired to host.
BiCamInqBytesPerPix - depth of pixel in bytes, as acquired to host.
BiCamInqBytesPerPixDisplay - depth of pixel in bytes, as acquired to display. (for RoadRunner use only)
BiCamInqBitsPerSequence - depth of multi-channel pixel in bits, as acquired to host.for RoadRunner use only)
BiCamInqBitsPerSequenceDisplay - depth of multi-channel pixel in bits, as acquired to display.for RoadRunner use only)
BiCamInqFrameSize0 - camera 0 total size of image in bytes, as acquired to host.
BiCamInqFrameSize1 - camera 1 total size of image in bytes, as acquired to host. (for Raven use only)
BiCamInqFrameSize2 - camera 2 total size of image in bytes, as acquired to host. (for Raven use only)
BiCamInqFrameSize3 - camera 3 total size of image in bytes, as acquired to host. (for Raven use only)
BiCamInqDisplayFrameSize0 - total size of image in bytes, as acquired to display. (for RoadRunner and R3 use only)
BiCamInqFrameWidth - width of image in bytes, as acquired to host.
BiCamInqDisplayFrameWidth - width of image in bytes, as acquired to display. (for RoadRunner and R3 use only)

BiCamInqAqTimeout - number of milliseconds to wait before acquisition command times out.
BiCamInqCamType - camera type.
BiCamInqControlType - type of camera control accessible through API. (for RoadRunner and R3 use only)

pVal

Pointer returned containing the requested value.

Returns

BI_OK	Function was successful.
BI_ERROR_BAD_BOARDPTR_INQ	Bad board pointer, or board type unknown.
BI_ERROR_UNKNOWN_PARAMETER	*InquireVar* parameter is unknown for this board.

Comments

This function is used to inquire of the system characteristics of the board.

2.8 BiBrdClose

Prototype BIRC BiBrdClose(Bd Board)

Description Closes the board and frees all associated resources.

Parameters *Board*

Board to handle.

Returns

BI_OK	In all cases.

Comments This function closes the board and releases associated resources. This function must be called whenever a process exits regardless of the reason the process is exiting. The only time that this function does not have to be called is if BiBrdOpen fails. This function decrements the internal counters that are used to keep track of the number of processes that have opened the board.

BufIn Camera Functions

Chapter 3

3.1 Introduction

One of the most powerful features of BitFlow's interface boards, is the ability for the board to interface to an almost infinite variety of cameras. The knowledge behind these interfaces is stored in the camera configuration files.

The normal way a BitFlow application works is that the board is initialized to interface to the camera currently attached to the board. The currently attached camera is selected in the SysReg utility program. Normally an application is written so that it will work with whatever camera is attached. The board is initialized for the currently attached camera when BiBrdOpen is called. If an application is written this way there is no need to call any of the functions in this chapter. However, some users may want to manage what cameras are attached and how the user switches between them using their own software. For this reason, these camera configuration functions are provided.

The normal flow for an application that wants to manage its own camera files is as follows:

```
BiBrdOpen
BiCamOpen
BiCamSetCur
// processing and acquisition
BiCamClose
BiBrdClose
```

If using more than one camera:

```
BiBrdOpen
BiCamOpen            // open camera 0
BiCamOpen            // open camera 1
BiCamSetCur          // configure for camera 0

// processing and acquisition
BiCamSetCur          // configure for camera 1

// processing and acquisition
BiCamClose           // close camera 0
BiCamClose           // close camera 1
BiBrdClose
```

3.2 BiCamOpen

Prototype

BFRC BiCamOpen(Bd *Board*, PBIBA *pBufArray*, PCHAR *CamName*, PBFCNF **pCam*)

Description

Allocates a camera configuration object, opens a camera configuration file, and loads the file into the object.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

CamName

The name of the camera file to open. Do not include the path. The camera file must be in the configuration directory (see the SysReg application). For example: "GenRS170-PLL.rvc".

**pCam*

A pointer to a camera object. The memory to hold the object is allocated in this function.

Returns

BI_OK	If successful.
BI_ERROR_CAM_OPEN	An error occured while trying to open the camera file. Please see the event viewer for more information on this failure. Most times the problem is that the camera file specified does not exist or could not be found.

Comments

This function allocates memory to hold a camera configuration object, locates the given camera configuration file in the configuration directory, checks the file for errors, then loads the camera configuration parameters into the camera object. The camera object is used to tell the system how to set up the board to acquire from a particular camera. Use the program CamVert to edit camera configuration files.

The resulting camera object can be passed to other functions such as BiCamSetCur.

The resources allocated by the function must be freed by calling BiCamClose.

3.3 BiCamClose

Prototype BFRC BiCamClose(Bd *Board*, PBIBA *pBufArray*, PBFCNF *pCam*)

Description Frees resources used by a camera object.

Parameters *Board*

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

pCam

Camera object.

Returns

BI_OK	If successful.
BI_ERROR_CAM_CLOSE	An error occured while closing the camera file.

Comments This function frees all resources used by a camera object.

3.4 BiCamSel

Prototype BFRC BiCamSel(Bd *Board*, PBIBA *pBufArray*, BFU32 *CamIndex*, BFU32 *Mode*)

Description Sets a board's current camera to the camera with the given index. Depending on the *mode*, the board can also be initialized for this camera.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

CamIndex

Index of camera to become current. Index is set in SysReg.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but do not modify the board.
CiSysConfigure - initialize the board for this camera.

Returns

BI_OK	Function was successful.
BI_ERROR_CAM_SEL	If an error occurs selecting the camera index. This is usually caused by an invalid CamIndex.

Comments Each board has associated with it a list of configured cameras (set in the SysReg application) and a current camera. By default, the current camera is the first camera in the list of configured cameras. The current camera is important because it dictates the parameters used for acquisition. There must be a current camera set in order to use the acquisition functions. This function allows you to pick one of the configured cameras to be the current camera.

If *Mode* = CiSysConfigure, the board will be initialized for the given camera.

This function is useful for switching on-the-fly between multiple preconfigured camera types.

3.5 BiCamSetCur

Prototype BFRC BiCamSetCur(Bd *Board*, PBIBA *pBufArray*, PRVCAM *pCam*, BFU32 *Mode*)

Description Sets the current camera to the camera object *pCam* that is not necessarily one of the preconfigured cameras. The board can be optionally initialized to the camera.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

pCam

A camera object.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but does not modify the board.
CiSysConfigure - initialize the board for this camera.

Returns

BI_OK	Function was successful.
BI_ERROR_CAM_SET_CUR	If an error occurs setting the camera file.

Comments This function sets the current camera to a camera object that is not one of the cameras already configured for the board (via SysReg). The camera must already be opened successfully (see BiCamOpen).

This function allows you to handle your own camera management. You can select, open, configure and close cameras to suit your applications needs independently of the SDK's camera management.

If *Mode* = CiSysConfigure, the board will be initialized for the given camera.

3.6 BiCamGetCur

Prototype BFRC BiCamGetCur(Bd *Board*, PBIBA *pBufArray*, PBFCNF **pCam*)

Description Gets the current camera object the board is using.

Parameters

Board

Board to select.

pBufArray

A pointer to a structure that holds all acquisition information.

**pCam*

Pointer to the camera object.

Returns

BI_OK	If successful.
BI_ERROR_GET_CUR	If unable to retrieve the current camera object.

Comments

3.7 BiCamGetFileName

Prototype BFRC BiCamGetFileName (Bd *Board*, PBIBA *pBufArray*, BFU32 *Num*, PBFCHAR *CamName*, BFSIZET *CamNameStLen*)

Description Gets the file name of the attached camera(s).

Parameters

Board

Board to select.

pBufArray

A pointer to a structure that holds all acquisition information.

Num

Camera number to get the name of.

CamName

Contains the file name of the camera configuration.

CamNameStLen

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *CamName*.

Returns

BI_OK	If successful.
BI_ERROR_CAM_FILENAME	If unable to return the name of the camera file.

Comments This function can be used to get the file name for one of the attached camera configurations. These configurations are attached to the board in SysReg. The *Num* parameter corresponds to the number configuration in the list of attached cameras in SysReg.

BufIn Acquisition Functions

Chapter 4

4.1 Introduction

The acquisition functions are some of the most important in the BufIn SDK. While the initialization functions set up the board's registers for a particular camera, these functions do most of the work required to get the board reading to DMA the images to memory.

The concept here is that the setup functions are time and CPU intensive, so they should be called before any time critical processing has begun. In a sense, these are extensions of the initialization process. Once the setup functions are called for a particular buffer, they need not be called again.

The cleanup function frees up any resources allocated in the setup functions, and put the DMA engine in an idle mode.

For example, the basic flow of a program would be:

```
BiBrdOpen
BiSeqAqSetup or BiCircAqSetup
Loop

// Acquisitions and/or processing

BiSeqCleanUp or BiCircCleanUp
BiBrdClose
```

The bulk of the work is done in the setup functions. These functions create a scatter gather table based on the virtual memory address, called a relative QTab.

The relative QTab is passed to the kernel driver, where the destination buffer is locked down (so that it cannot be paged to disk) and the physical address are determined for each page of the buffer (Windows uses 4K byte pages). These physical addresses are used to build a physical QTab. This physical QTab is then written to the board in preparation scatter gather DMAing.

Finally, the DMA engine is initialized and started. Again, this function need be called only once, for a particular destination buffer.

4.2 BiSeqAqSetup

Prototype BIRC BiSeqAqSetup(Bd *Board*, PBIBA *pBufArray*, BFU32 *Options*)

Description Sets up the system for sequence acquisition. When the system is setup using this function it will do only one thing, acquire frames sequentially to the host buffers.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Options

Setup options for sequence capture are as follows:

BiAqEngJ - Use DMA engine J. Only the J engine can be used at this time.
AbortMissedFrame - If a frame is missed, acquisition will stop.
DisableAqErrorSig - The overflow and hardware exception signals will not be created.
UseHighResTimer - The high resolution timer is used to time stamp incoming image data.
InvertEvenFrames - The even frames will be DMAed from the bottom of the buffer to the top.
InvertOddFrames - The odd frames will be DMAed from the bottom of the buffer to the top.
OnlyOddLines - The board will DMA the incoming image to only the odd lines of the host buffer (2x DMA on the Karbon only).
OnlyEvenLines - The board will DMA the incoming image to only the even lines of the host buffer (2x DMA on the Karbon only).

Returns

BI_OK	If successful.
BI_ERROR_LUT_MASK	Error in masking LUT.
BI_ERROR_QTABARRAY_MEM	Lack of memory for QTab array allocation.
BI_ERROR_REL_QTAB	Error creating any/all relative QTABs.
BI_ERROR_PHYS_QTAB	Error creating any/all physical QTABs.
BI_ERROR_CREATE_CTAB_SIG	Unable to create signal for CTABs.
BI_ERROR_CREATE_THREAD	Unable to create worker thread.
BI_ERROR_SIGCREATE_EX	Unable to create signal for exceptions.
BI_ERROR_SIGCREATE_OVF	Unable to create signal for overflow.

BI_ERROR_SIGCREATE_QUAD	Unable to create signal for quad done.
BI_ERROR_SIGCREATE_ STARTSTOP	Unable to create signal for start/stop.
BI_ERROR_CREATE_ ERRORTHREAD	Unable to create error thread.
BI_ERROR_ENGINEK	DMA engine K is currently not supported.
BI_ERROR_P_QTABARRAY_MEM	Lack of memory for array of host QTab pointers.
BI_ERROR_CHAIN_LINK	Error linking physical qtabs together.
BI_ERROR_QTABHOST_SKIPFRAME	Skip frames is currently not supported for host qtabs.
BI_ERROR_HOST_MODE	Board is currently setup for qtabs on the board.
BI_ERROR_SEQINFO_MEM	Lack of memory for sequence buffer information.
BI_ERROR_SEQARRAY_MEM	Lack of memory for sequence buffer information array.

Comments

This function sets up the entire board's acquisition systems for acquisition to host. This function need be called only once, before acquisition begins. It does not need to be called again unless BiSeqCleanUp is called. BiSeqCleanUp should be called when done acquiring in order to free up resources used by this process. Once this function is called, the function BiSeqControl can be used to grab, freeze or abort acquisition.

Options can be OR together. For example, BiAqEngJ|UseHighResTimer is a legal and typical usage for the options parameter.

4.3 BiSeqAqSetupROI

Prototype

BIRC BiSeqAqSetupROI(Bd Board, PBIBA pBufArray, BFU32 *XOffset*, BFU32 *YOffset*, BFU32 *XSize*, BFU32 *YSize*, BFU32 Options)

Description

Sets up the system for sequence acquisition. When the system is setup using this function it will do only one thing, acquire frames sequentially to the host buffers. The board will be setup for the given sub-window (ROI).

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

XOffset

Number of pixels to offset horizontally the captured sub-window from the camera's output image.

YOffset

Number of lines to offset vertically the captured sub-window from the camera's output image.

XSize

Width in pixels of the captured sub-window.

YSize

Height in lines of the captured sub-window.

Options

Setup options for sequence capture are as follows:

BiAqEngJ - Use DMA engine J. Only the J engine can be used at this time.
AbortMissedFrame - If a frame is missed, acquisition will stop.
DisableAqErrorSig - The overflow and hardware exception signals will not be created.
UseHighResTimer - The high resolution timer is used to time stamp incoming image data.
InvertEvenFrames - The even frames will be DMAed from the bottom of the buffer to the top.
InvertOddFrames - The odd frames will be DMAed from the bottom of the buffer to the top.
OnlyOddLines - The board will DMA the incoming image to only the odd lines of the host buffer (2x DMA on the Karbon only).

OnlyEvenLines - The board will DMA the incoming image to only the even lines of the host buffer (2x DMA on the Karbon only).

Returns

BI_OK	If successful.
BI_ERROR_LUT_MASK	Error in masking LUT.
BI_ERROR_QTABARRAY_MEM	Lack of memory for QTab array allocation.
BI_ERROR_REL_QTAB	Error creating any/all relative QTABs.
BI_ERROR_PHYS_QTAB	Error creating any/all physical QTABs.
BI_ERROR_CREATE_CTAB_SIG	Unable to create signal for CTABs.
BI_ERROR_CREATE_THREAD	Unable to create worker thread.
BI_ERROR_SIGCREATE_EX	Unable to create signal for exceptions.
BI_ERROR_SIGCREATE_OVF	Unable to create signal for overflow.
BI_ERROR_SIGCREATE_QUAD	Unable to create signal for quad done.
BI_ERROR_SIGCREATE_ STARTSTOP	Unable to create signal for start/stop.
BI_ERROR_CREATE_ ERRORTHREAD	Unable to create error thread.
BI_ERROR_ENGINEK	DMA engine K is currently not supported.
BI_ERROR_P_QTABARRAY_MEM	Lack of memory for array of host QTab pointers.
BI_ERROR_CHAIN_LINK	Error linking physical qtabs together.
BI_ERROR_QTABHOST_SKIPFRAME	Skip frames is currently not supported for host qtabs.
BI_ERROR_HOST_MODE	Board is currently setup for qtabs on the board.
BI_ERROR_SEQINFO_MEM	Lack of memory for sequence buffer information.
BI_ERROR_SEQARRAY_MEM	Lack of memory for sequence buffer information array.
BI_ERROR_CIR_ROI_ERROR	The board can not be programmed to the requested ROI

Comments

This function sets up the entire board's acquisition systems for acquisition to host. This function need be called only once, before acquisition begins. It does not need to be called again unless BiSeqCleanUp is called. BiSeqCleanUp should be called when done acquiring in order to free up resources used by this process. Once this function is called, the function BiSeqControl can be used to grab, freeze or abort acquisition.

This function programs the board to capture a sub-windows out of the camera's full resolution output. The sub-window must be smaller than the resolution as set in the camera configuration file. There granularity of the parameters XOffset and XSize will vary depending on frame grabber model and tap format, however, it will generally be greater than four pixels. The granularity of the parameters YOffset and YSize is one in most cases.

Options can be OR together. For example, BiAqEngJ|UseHighResTimer is a legal and typical usage for the options parameter.

4.4 BiSeqAqSetupPitch

Prototype BIRC BiSeqAqSetupPitch(Bd *Board*, PBIBA *pBufArray*, BFU32 *Pitch*, BFU32 *Options*)

Description Sets up the system for sequence acquisition. When the system is setup using this function it will do only one thing, acquire frames sequentially to the host buffers. This function allows the user to override the default pitch use to write pixels in the host buffer. Changing to different pitch supports acquiring into a buffer that is "wider" than the line size.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Pitch

Number of bytes the DMA engine should use when calculating the memory offset from a pixel on one line to the same pixel on the line below.

Options

Setup options for sequence capture are as follows:

BiAqEngJ - Use DMA engine J. Only the J engine can be used at this time.
AbortMissedFrame - If a frame is missed, acquisition will stop.
DisableAqErrorSig - The overflow and hardware exception signals will not be created.
UseHighResTimer - The high resolution timer is used to time stamp incoming image data.
InvertEvenFrames - The even frames will be DMAed from the bottom of the buffer to the top.
InvertOddFrames - The odd frames will be DMAed from the bottom of the buffer to the top.
OnlyOddLines - The board will DMA the incoming image to only the odd lines of the host buffer (2x DMA on the Karbon only).
OnlyEvenLines - The board will DMA the incoming image to only the even lines of the host buffer (2x DMA on the Karbon only).

Returns

BI_OK	If successful.
BI_ERROR_LUT_MASK	Error in masking LUT.
BI_ERROR_QTABARRAY_MEM	Lack of memory for QTab array allocation.

BI_ERROR_REL_QTAB	Error creating any/all relative QTABs.
BI_ERROR_PHYS_QTAB	Error creating any/all physical QTABs.
BI_ERROR_CREATE_CTAB_SIG	Unable to create signal for CTABs.
BI_ERROR_CREATE_THREAD	Unable to create worker thread.
BI_ERROR_SIGCREATE_EX	Unable to create signal for exceptions.
BI_ERROR_SIGCREATE_OVF	Unable to create signal for overflow.
BI_ERROR_SIGCREATE_QUAD	Unable to create signal for quad done.
BI_ERROR_SIGCREATE_ STARTSTOP	Unable to create signal for start/stop.
BI_ERROR_CREATE_ ERRORTHREAD	Unable to create error thread.
BI_ERROR_ENGINEK	DMA engine K is currently not supported.
BI_ERROR_P_QTABARRAY_MEM	Lack of memory for array of host QTab pointers.
BI_ERROR_CHAIN_LINK	Error linking physical qtabs together.
BI_ERROR_QTABHOST_SKIPFRAME	Skip frames is currently not supported for host qtabs.
BI_ERROR_HOST_MODE	Board is currently setup for qtabs on the board.
BI_ERROR_SEQINFO_MEM	Lack of memory for sequence buffer information.
BI_ERROR_SEQARRAY_MEM	Lack of memory for sequence buffer information array.
BI_ERROR_CIR_ROI_ERROR	The board can not be programmed to the requested ROI

Comments

This function sets up the entire board's acquisition systems for acquisition to host. This function need be called only once, before acquisition begins. It does not need to be called again unless BiSeqCleanUp is called. BiSeqCleanUp should be called when done acquiring in order to free up resources used by this process. Once this function is called, the function BiSeqControl can be used to grab, freeze or abort acquisition.

This function programs the board to capture using a non-standard pitch. Pitch is the number of bytes between that address of the first pixel on one line to the first pixel on the line below it. By using a custom pitch, the board can be used to DMA an image into a buffer that is wider than the number of pixels in the line. This can be useful when stitching the output from two adjacent cameras into the same host buffer.

For example, if the two cameras were 1024 pixel each. You could allocated a host buffer that is 2048 pixels wide. The frame grabber could be set up to acquire from the first camera in the left half of the host buffer, using a pitch of 2048. The other camera could be set up to acquire into the right half of the host buffer also using a pitch of 2048.

Options can be OR together. For example, BiAqEngJ|UseHighResTimer is a legal and typical usage for the options parameter.

4.5 BiCircAqSetup

Prototype BIRC BiCircAqSetup(Bd *Board*, PBIBA *pBufArray*, BFU32 *ErrorMode*, BFU32 *Options*)

Description Sets up the system for acquisition to a circular set of buffers. For circular buffer acquisition the board will be acquiring into one buffer while the CPU processes a previous buffer.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

ErrorMode

If the system is filling buffers faster than the user is marking them available, eventually the system will run out of buffers. The user can control what to do in this situation with the following error modes:

CirErStop - Stop acquiring images if the buffers are full and unavailable.
CirErIgnore - Continue acquisition by overwriting the buffers regardless of the status.

Options

Setup options for circular capture are as follows:

BiAqEngJ - Use DMA engine J. Only the J engine can be used at this time.
AbortMissedFrame - If a frame is missed, acquisition will stop.
DisableAqErrorSig - The overflow and hardware exception signals will not be created.
UseHighResTimer - The high resolution timer is used to time stamp incoming image data.
InvertEvenFrames - The even frames will be DMAed from the bottom of the buffer to the top.
InvertOddFrames - The odd frames will be DMAed from the bottom of the buffer to the top.
OnlyOddLines - The board will DMA the incoming image to only the odd lines of the host buffer (2x DMA on the Karbon only).
OnlyEvenLines - The board will DMA the incoming image to only the even lines of the host buffer (2x DMA on the Karbon only).

Returns

BI_OK	If successful.

BI_ERROR_CIR_ENGINEK	DMA engine K is currently not supported.
BI_ERROR_CIR_FIRMWARE	Host QTab firmware is not present on the board.
BI_ERROR_CIR_LUT_MASK	Error in masking LUT.
BI_ERROR_CIR_QTABARRAY_MEM	Lack of memory for QTab array.
BI_ERROR_CIR_P_QTABARRAY_ MEM	Lack of memory for array of host QTab pointers.
BI_ERROR_CIR_REL_QTAB	Could not create all relative QTABs.
BI_ERROR_CIR_PHYS_QTAB	Could not create all physical QTABs.
BI_ERROR_CIR_CHAIN_LINK	Could not link qtabs.
BI_ERROR_CIR_CREATE_THREAD	Could not create circular worker thread.
BI_ERROR_CIR_SIG_OVF	Overflow signal not created.
BI_ERROR_CIR_SIG_EX	Exception signal not created.
BI_ERROR_CIR_SIG_QUAD	Quad done signal not created.
BI_ERROR_CIR_SIG_STARTSTOP	Start/Stop signal not created.
BI_ERROR_CIR_SIG_ERROR-THREAD	Internal error signal not created.
BI_ERROR_CIR_BUF_STAT	Lack of memory to create buffer status array.
BI_ERROR_CIR_BUF_QUEUE	Lack of memory to create circular buffer queue.

Comments

This function sets up the system for acquisition to a circular set of buffers. The concept of circular buffers is that the board will aqcquire to one buffer out of the set while the CPU processes another, different buffer out of the set. Each buffer in the set has its own status. The buffers will start out with the status of BIAVAILABLE. The status of the buffers will change from BIAVAILABlE to BIFRESH when the buffer is filled with new data from the board. With the BiCirWaitDoneFrame function, the user removes the buffer from the queue and the buffers status is marked BINEW. When the user is done processing the buffer, it is their responsibility to mark the buffer BIAVAILABLE with the BiCirStatusSet function. The user can also use the BiCirStatusSet function to give a buffer status of BIHOLD. If a buffer has a status of BIHOLD, the buffer will not be acquired into the next time it is up as a destination. The marked BIHOLD will be skipped indefinitely until the user marks the buffer as BIAVAILABLE.

Options can be OR together. For example, BiAqEngJ|UseHighResTimer is a legal and typical usage for the options parameter.

4.6 BiCircAqSetupROI

Prototype BIRC BiCircAqSetupROI(Bd *Board*, PBIBA *pBufArray*, BFU32 *XOffset*, BFU32 *YOffset*, BFU32 *XSize*, BFU32 *YSize*, BFU32 *ErrorMode*, BFU32 *Options*)

Description Sets up the system for acquisition to a circular set of buffers. For circular buffer acquisition the board will be acquiring into one buffer while the CPU can be processing a previous buffer. The board will be setup for the given sub-window (ROI).

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

XOffset

Number of pixels to offset horizontally the captured sub-window from the camera's output image.

YOffset

Number of lines to offset vertically the captured sub-window from the camera's output image.

XSize

Width in pixels of the captured sub-window.

YSize

Height in lines of the captured sub-window.

ErrorMode

If the system is filling buffers faster than the user is marking them available, eventually the system will run out of buffers. The user can control what to do in this situation with the following error modes:

CirErStop - Stop acquiring images if the buffers are full and unavailable.
CirErIgnore - Continue acquisition by overwriting the buffers regardless of the status.

Options

Setup options for circular capture are as follows:

BiAqEngJ - Use DMA engine J. Only the J engine can be used at this time.
AbortMissedFrame - If a frame is missed, acquisition will stop.

DisableAqErrorSig - The overflow and hardware exception signals will not be created.
UseHighResTimer - The high resolution timer is used to time stamp incoming image data.
InvertEvenFrames - The even frames will be DMAed from the bottom of the buffer to the top.
InvertOddFrames - The odd frames will be DMAed from the bottom of the buffer to the top.
OnlyOddLines - The board will DMA the incoming image to only the odd lines of the host buffer (2x DMA on the Karbon only).
OnlyEvenLines - The board will DMA the incoming image to only the even lines of the host buffer (2x DMA on the Karbon only).

Returns

BI_OK	If successful.
BI_ERROR_CIR_ENGINEK	DMA engine K is currently not supported.
BI_ERROR_CIR_FIRMWARE	Host QTab firmware is not download to board.
BI_ERROR_CIR_LUT_MASK	Error in masking LUT.
BI_ERROR_CIR_QTABARRAY_MEM	Lack of memory for QTab array.
BI_ERROR_CIR_P_QTABARRAY_MEM	Lack of memory for array of host QTab pointers.
BI_ERROR_CIR_REL_QTAB	Could not create all relative QTABs.
BI_ERROR_CIR_PHYS_QTAB	Could not create all physical QTABs.
BI_ERROR_CIR_CHAIN_LINK	Could not link qtabs.
BI_ERROR_CIR_CREATE_THREAD	Could not create circular worker thread.
BI_ERROR_CIR_SIG_OVF	Overflow signal not created.
BI_ERROR_CIR_SIG_EX	Exception signal not created.
BI_ERROR_CIR_SIG_QUAD	Quad done signal not created.
BI_ERROR_CIR_SIG_STARTSTOP	Start/Stop signal not created.
BI_ERROR_CIR_SIG_ERROR-THREAD	Internal error signal not created.
BI_ERROR_CIR_BUF_STAT	Lack of memory to create buffer status array.
BI_ERROR_CIR_BUF_QUEUE	Lack of memory to create circular buffer queue.
BI_ERROR_CIR_ROI_ERROR	The board can not be programmed to the requested ROI

Comments

This function sets up the system for acquisition to a circular set of buffers. The concept of circular buffers is that the board will be acquiring to one buffer out of the set, while the CPU can be processing another, different, buffer out of the set. Each buffer in the set has its own status. The buffers will start out with the status of BIAVAILABLE. The status of the buffers will change from BIAVAILABlE to BIFRESH when the buffer is filled with new data from the board. With the BiCirWaitDoneFrame function, the user removes the buffer from the queue and the buffers status is marked BINEW. When the user is done processing the a buffer, it is their responsibility to mark the buffer BIAVAILABLE with the BiCirStatusSet function. The user can also use the BiCirStatus-Set function to give a buffer status of BIHOLD. If a buffer has a status of BIHOLD, the buffer will not be acquired into the next time it is up as a destination. The marked BIHOLD will be skipped indefinitely until the user marks the buffer as BIAVAILABLE.

This function programs the board to capture a sub-windows out of the camera's full resolution output. The sub-window must be smaller than the resolution as set in the camera configuration file. There granularity of the parameters XOffset and XSize will vary depending on frame grabber model and tap format, however, it will generally be greater than four pixels. The granularity of the parameters YOffset and YSize is one in most cases.

Options can be OR together. For example, BiAqEngJ|UseHighResTimer is a legal and typical usage for the options parameter.

4.7 BiCircAqSetupPitch

Prototype BIRC BiCircAqSetupPitch(Bd *Board*, PBIBA *pBufArray*, BFU32 *Pitch*, BFU32 *Options*)

Description Sets up the system for acquisition to a circular set of buffers. For circular buffer acquisition the board will be acquiring into one buffer while the CPU can be processing a previous buffer. This function allows the user to override the default pitch use to write pixels in the host buffer. Changing to different pitch supports acquiring into a buffer that is "wider" than the line size.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Pitch

Number of bytes the DMA engine should use when calculating the memory offset from a pixel on one line to the same pixel on the line below.

ErrorMode

If the system is filling buffers faster than the user is marking them available, eventually the system will run out of buffers. The user can control what to do in this situation with the following error modes:

CirErStop - Stop acquiring images if the buffers are full and unavailable.
CirErIgnore - Continue acquisition by overwriting the buffers regardless of the status.

Options

Setup options for circular capture are as follows:

BiAqEngJ - Use DMA engine J. Only the J engine can be used at this time.
AbortMissedFrame - If a frame is missed, acquisition will stop.
DisableAqErrorSig - The overflow and hardware exception signals will not be created.
UseHighResTimer - The high resolution timer is used to time stamp incoming image data.
InvertEvenFrames - The even frames will be DMAed from the bottom of the buffer to the top.
InvertOddFrames - The odd frames will be DMAed from the bottom of the buffer to the top.
OnlyOddLines - The board will DMA the incoming image to only the odd lines of the host buffer (2x DMA on the Karbon only).
OnlyEvenLines - The board will DMA the incoming image to only the even

lines of the host buffer (2x DMA on the Karbon only).

Returns

BI_OK	If successful.
BI_ERROR_CIR_ENGINEK	DMA engine K is currently not supported.
BI_ERROR_CIR_FIRMWARE	Host QTab firmware is not download to board.
BI_ERROR_CIR_LUT_MASK	Error in masking LUT.
BI_ERROR_CIR_QTABARRAY_MEM	Lack of memory for QTab array.
BI_ERROR_CIR_P_QTABARRAY_MEM	Lack of memory for array of host QTab pointers.
BI_ERROR_CIR_REL_QTAB	Could not create all relative QTABs.
BI_ERROR_CIR_PHYS_QTAB	Could not create all physical QTABs.
BI_ERROR_CIR_CHAIN_LINK	Could not link qtabs.
BI_ERROR_CIR_CREATE_THREAD	Could not create circular worker thread.
BI_ERROR_CIR_SIG_OVF	Overflow signal not created.
BI_ERROR_CIR_SIG_EX	Exception signal not created.
BI_ERROR_CIR_SIG_QUAD	Quad done signal not created.
BI_ERROR_CIR_SIG_STARTSTOP	Start/Stop signal not created.
BI_ERROR_CIR_SIG_ERROR-THREAD	Internal error signal not created.
BI_ERROR_CIR_BUF_STAT	Lack of memory to create buffer status array.
BI_ERROR_CIR_BUF_QUEUE	Lack of memory to create circular buffer queue.
BI_ERROR_CIR_ROI_ERROR	The board can not be programmed to the requested ROI

Comments

This function sets up the system for acquisition to a circular set of buffers. The concept of circular buffers is that the board will be acquiring to one buffer out of the set, while the CPU can be processing another, different, buffer out of the set. Each buffer in the set has its own status. The buffers will start out with the status of BIAVAILABLE. The status of the buffers will change from BIAVAILABlE to BIFRESH when the buffer is filled with new data from the board. With the BiCirWaitDoneFrame function, the user removes the buffer from the queue and the buffers status is marked BINEW. When the user is done processing the a buffer, it is their responsibility to mark the buffer BIAVAILABLE with the BiCirStatusSet function. The user can also use the BiCirStatus-Set function to give a buffer status of BIHOLD. If a buffer has a status of BIHOLD, the buffer will not be acquired into the next time it is up as a destination. The marked BIHOLD will be skipped indefinitely until the user marks the buffer as BIAVAILABLE.

This function programs the board to capture using a non-standard pitch. Pitch is the number of bytes between that address of the first pixel on one line to the first pixel on the line below it. By using a custom pitch, the board can be used to DMA an image into a buffer that is wider than the number of pixels in the line. This can be useful when stitching the output from two adjacent cameras into the same host buffer.

For example, if the two cameras were 1024 pixel each. You could allocated a host buffer that is 2048 pixels wide. The frame grabber could be set up to acquire from the first camera in the left half of the host buffer, using a pitch of 2048. The other camera could be set up to acquire into the right half of the host buffer also using a pitch of 2048.

Options can be OR together. For example, BiAqEngJ|UseHighResTimer is a legal and typical usage for the options parameter.

4.8 BiSeqCleanUp

Prototype BIRC BiSeqCleanUp(Bd *Board*, PBIBA *pBufArray*)

Description Frees all resources used by the acquisition process. Makes sure the board is in a stable state.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	If successful.
BI_ERROR_SIGCANCEL_QUAD	Unable to cancel signal for quad done.
BI_ERROR_SIGFREE_QUAD	Unable to free signal for quad done.
BI_ERROR_SIGCANCEL_START-STOP	Unable to cancel signal for start/stop.
BI_ERROR_SIGFREE_STARTSTOP	Unable to free signal for start/stop.
BI_ERROR_SIGCANCEL_EX	Unable to cancel signal for exceptions.
BI_ERROR_SIGFREE_EX	Unable to free signal for exceptions.
BI_ERROR_SIGCANCEL_OVF	Unable to cancel signal for overflow.
BI_ERROR_SIGFREE_OVF	Unable to free signal for overflow.
BI_ERROR_PHYSQTAB_FREE	Unable to free physical qtabs.
BI_ERROR_RELQTAB_FREE	Unable to free relative qtabs.
BI_ERROR_AQ_CLEANUP	Failure with the CiSeqCleanUp function.

Comments This function frees all of the resources that were allocated in BiSeqAqSetup. Do not call this function unless you have already called BiSeqAqSetup and are finished acquiring into the current buffer.

4.9 BiCircCleanUp

Prototype BIRC BiCircCleanUp(Bd *Board*, PBIBA *pBufArray*)

Description Frees all resources used by the acquisition process. Makes sure the board is in a stable state.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	If successful.
BI_ERROR_CIR_SIGCANCEL_QUAD	Unable to cancel signal for quad done.
BI_ERROR_CIR_SIGFREE_QUAD	Unable to free signal for quad done.
BI_ERROR_CIR_SIGCANCEL_STARTSTOP	Unable to cancel signal for start/stop.
BI_ERROR_CIR_SIGFREE_STARTSTOP	Unable to free signal for start/stop.
BI_ERROR_CIR_SIGCANCEL_EX	Unable to cancel signal for exceptions.
BI_ERROR_CIR_SIGFREE_EX	Unable to free signal for exceptions.
BI_ERROR_CIR_SIGCANCEL_OVF	Unable to cancel signal for overflow.
BI_ERROR_CIR_SIGFREE_OVF	Unable to free signal for overflow.
BI_ERROR_CIR_PHYSQTAB_FREE	Unable to free physical qtabs.
BI_ERROR_CIR_RELQTAB_FREE	Unable to free relative qtabs.
BI_ERROR_CIR_AQ_CLEANUP	Failure with the CiCircCleanUp function.

Comments This function frees all of the resources that were allocated in BiCircAqSetup. Do not call this function unless you have already called BiCircAqSetup and are finished acquiring into the current buffer.

4.10 BiInternalTimeoutSet

Prototype BIRC BiInternalTimeoutSet(Bd *Board*, PBIBA *pBufArray*, BFU32 *TimeoutValue*)

Description Sets the timeout value for operations within the BufIn acquisition engine.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

TimeoutValue

The timeout value in mS.

Returns

BI_OK	If successful.
BI_ERROR_TIMEOUT_SET	Operation failed, the timeout value was not set.

Comments This function sets the timeout value for the internal operations within the Bufin acquisition engine. These operations include, but are not limited to, waiting for a frame, waiting for acquisition to start or abort, and waiting for active regions with a frame.

4.11 BiCallBackAdd

Prototype BIRC BiCallBackAdd(Bd *Board*, PBIBA *pBufArray*, BiCallBackFuncPtr *CallBackFunc*, PBFVOID *pUserData*)

Description Installs a Call Back function that is called whenever a new frame is acquired.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

CallBackFunc

A pointer to a Call Back function.

pUserData

A pointer to user allocated structure which can contain any context data that might be needed in the Call Back function when it is actually called. Can be NULL.

Returns

BI_OK	If successful.
BI_CB_NULL_POINTER	The parameter *CallBackFunc* is not a valid function pointer.
BI_CB_ALREADY_SET	A Call Back function has already been registered.
BI_CB_BAD_SEMAPHORE	Error creating the Windows objected needed for this Call Back.
BI_CB_THREAD_FAIL	Error creating the thread needed for this Call Back.
BI_CB_THREAD_GOING	The thread needed for this Call Back already exists.

Comments This function registers a function that will be called when ever a new frame has been acquired and is available for processing. The Call Back function must be removed prior to closing the board via the function BiCallBackRemove(). After this function is called, the registered Call Back function will no longer be called.

The Call Back function must have the following format:

```
void BiCallBackFuncPtr(Bd Board, PBIBA pBufArray, BiCirHandle
CirHandle, PBFVOID pUserData);
```

When the Call Back function is called, the *CirHandle* parameter contains the information about the newly acquired frame.

The Call Back function can be used in place of calling BiCircWaitFrameDone(). The same type of processing can either be done in the Call Back function or in a separate thread that calls BiCircWaitFrameDone(). Note that the processing in this Call Back function will take place at a priority set by the BufIn libraries, while calling BiCircWaitFrameDone() in your own processing thread lets you set the thread priority.

The pointer *pUserData* is designed so that the user can get context information inside of the call back function (when it is called). This pointer can point to anything (cast it inside the call back function). It must be allocated and de-allocated by the user. It can also be NULL.

4.12 BiCallBackRemove

Prototype BIRC BiCallBackRemove(Bd *Board*, PBIBA *pBufArray*)

Description Removes a Call Back function.

Parameters *Board*

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	If successful.

Comments This function removes a Call Back function that has been previously registered with the function BiCallBackAdd(). After this function is called, the registered Call Back function will no longer be called.

BufIn Memory Functions

Chapter 5

5.1 Introduction

This chapter contains functions that are used for memory management. The user has several options under the BufIn SDK. The user can specify the number of buffers they would like to acquire and let the BiBufferAllocCam function do the work of the proper allocation of memory, based on the current camera file. To free the memory allocated by BiBufferAllocCam, the user should use the BiBufferFree function. The other option for the user is to allocate memory on there own and assign that memory to the BufIn SDK. The assigning of user allocated memory to BufIn can be accomplished with the BiBufferAssign function. The user must use the BiBufferUnassign function to un-assign the user allocated memory that was assigned to BufIn. The user is also responsible for freeing the memory that was allocated.

The function BiBufferAlloc is intended to be used when reading from disk with the BiDiskBufRead. The BiBufferAllocCam is using the information in the camera file to allocate the proper size buffers. When reading a image from the disk, we can not use the camera file for information. BiBufferAlloc is provided to be able to allocate memory in BufIn without use of a camera file. The user must provide the appropriate information that would be found in the camera file.

The normal flow for an application using BiBufferAllocCam is as follows:

```
BiBrdOpen
BiBufferAllocCam
// processing and acquisition
BiBufferFree
BiBrdClose
```

The normal flow for an application using BiBufferAssign is as follows:

```
BiBrdOpen
// user allocated memory
BiBufferAssign
// processing and acquisition
BiBufferUnassign
// user frees memory
BiBrdClose
```

5.2 BiBufferAllocCam

Prototype BIRC BiBufferAllocCam(Bd *Board*, PBIBA *pBufArray*, BFU32 *NumBuffers*)

Description Allocates memory for the number of buffers specified by parameter NumBuffers. The buffers are the correct size for the camera currently attached to board.

Parameters

Board

Board to handle.

pBufArray

A returned pointer to a structure that holds all acquisition information.

NumBuffers

The number of buffers to be allocated.

Returns

BI_OK	If successful.
BI_ERROR_CAM_NO_MEM_ AVAIL	Lack of memory for buffer allocation. Reduce number of buffers.
BI_ERROR_CAM_FRAMESIZE	Error in inquiring frame size for memory allocation.
BI_ERROR_CAM_STACK_MEM	Lack of memory for error stack.
BI_ERROR_CAM_BUFFERS_ NUM	There must be at least two buffers to allocate.
BI_ERROR_CAM_PIXDEPTH	Pixel depth not supported.
BI_ERROR_CAM_UNKNOWN	Undefined error.

Comments This function allocates memory for *NumBuffers* number of buffers. The buffers are the correct size for the camera currently attached to board pointed to by handle *board*. An array of pointers to the buffers is returned in *pBufArray*. After a successful call to BiBufferAllocCam the returned *pBufArray* pointer will be used for all further accesses to the newly allocated memory.

BiBufferFree should be called to free up resources used by BiBufferAllocCam.

5.3 BiBufferAlloc

Prototype

BIRC BiBufferAlloc(Bd *Board*, PBIBA *pBufArray*, BFU32 *XSize*, BFU32 *YSize*, BFU32 *PixDepth*, BFU32 *NumBuffers*)

Description

Allocates memory for the number of buffers specified by parameter *NumBuffers*. The buffers size is determined by parameters *Xsize*, *YSize* and *PixDepth*.

Parameters

Board

Board to handle.

pBufArray

A returned pointer to a structure that holds all acquisition information.

XSize

Width of image in pixels.

YSize

Height of image in lines.

PixDepth

Depth of pixels in bits.

NumBuffers

The number of buffers to allocate.

Returns

BI_OK	If successful.
BI_ERROR_NO_MEM_AVAIL	Lack of memory for buffer allocation. Reduce number of buffers.
BI_ERROR_FRAMESIZE	Error in inquiring frame size for memory allocation.
BI_ERROR_STACK_MEM	Lack of memory for error stack.
BI_ERROR_BUFFERS_NUM	There must be at least two buffers to allocate.
BI_ERROR_PIXDEPTH	Pixel depth not supported.

Comments

This function provides a means of allocating memory for bufin when the camera file is not available and BiBufferAllocCam can not be used. This is the case when reading a saved image from disk. When reading a image from disk, the camera file is independent to this operation.

The following shows an example of how to use BiBufferAlloc:

```
BiBrdOpen// open board
BiDiskParamRead// Get parameters to pass to BiBufferAlloc
BiBufferAlloc// Allocate memory
BiDiskBufRead// Read in image to memory

// Display and/or process image

BiBufferFree// De-allocate memory
BiBrdClose// Close the board
```

After a successful call to BiBufferAlloc the returned *pBufArray* pointer will be used for all further accesses to the newly allocated memory.

BiBufferFree should be called to free up resources used by BiBufferAlloc.

5.4 BiBufferAssign

Prototype BIRC BiBufferAssign(Bd *Board*, PBIBA *pBufArray*, PBU32 **pMemArray*, BFU32 *NumBuffers*)

Description This function assigns memory allocated by the user to BufIn.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

**pMemArray*

A pointer to an array of pointers that points to each buffer that has been allocated by the user.

NumBuffers

The number of buffers that have been allocated by the user.

Returns

BI_OK	If successful.
BI_ERROR_MEM_SIZE	The frame size is larger than the size of the buffer allocated in memory.
BI_ERROR_ASSIGN_PIX-DEPTH	Pixel depth not supported.
BI_ERROR_NO_MEM_POINT	Memory could not be allocated for BufIn's array of pointers.

Comments For this function memory needs to be allocated by the user. With memory allocated, an array of pointers pointing to each frame will need to be created. The array of pointers will be passed to this function via the *pMemArray* parameter. This function will then assign the array of pointers to BufIn's array of pointers. The function will return BufIn's array of pointers via the *pBufArray* parameter. It is the users responsibility to allocate and set up the array of pointers correctly.

After a successful call to BiBufferAssign the returned *pBufArray* pointer will be used for all further accesses to the newly assigned memory.

BiBufferUnassign should be called to free up resources used by BiBufferAssign. The user will need to free the memory that they have allocated for the buffers and array of pointers.

As of BitFlow SDK version 4.5 BiBufferAssign supports allocation of non-sequential buffers in memory. Previous version of the SDK do not support non-sequential buffers in memory.

5.5 BiBufferFree

Prototype BIRC BiBufferFree(Bd *Board*, PBIBA *pBufArray*)

Description Frees the memory associated with *pBufArray*.

Parameters *Board*

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	In all cases.

Comments This function frees the resources created by BiBufferAllocCam and BiBufferAlloc.

5.6 BiBufferUnassign

Prototype BIRC BiBufferUnassign(Bd *Board*, PBIBA *pBufArray*)

Description Un-assigns resources created by BiBufferAssign.

Parameters *Board*

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	In all cases.

Comments This function frees the resources created by BiBufferAssign.

5.7 BiBufferArrayGet

Prototype BIRC BiBufferArrayGet(Bd *Board*, PBIBA *pBufArray*, PBFU32 ***BufferArray*)

Description Returns a pointer to the array of buffer pointers.

Parameters *Board*

Board to handle.

pBufArray

A returned pointer to a structure that holds all acquisition information.

****BufferArray***

A pointer to the array of buffer pointers.

Returns

BI_OK	In all cases.

Comments With the return of this function, the user can then access the buffers through the *BufferArray* parameter. *BufferArray* will point to an array of pointers, each one pointing to each of the buffers that have been allocated.

See example "BiSeqSimpleDisp.c" to see BiBufferArrayGet be used.

5.8 BiBufferClear

Prototype BIRC BiBufferClear(Bd *Board*, PBIBA *pBufArray*)

Description Clears all buffer's contents by writing zeros to all buffers.

Parameters *Board*

Board to handle.

pBufArray

A returned pointer to a structure that holds all acquisition information.

Returns

BI_OK	In all cases.

Comments This function clears all image data from every buffer allocated by filling each buffer with zero data. Buffers must be allocated or assinged with the BiBufferAlloc, BiBuffer-AllocCam or BiBufferAssign functions before calling this function.

5.9 BiBufferAllocAlignedCam

Prototype

BIRC BiBufferAllocAlignedCam(Bd *Board*, PBIBA *pBufArray*, BFU32 *NumBuffers*, BFSIZET *Alignment*)

Description

Allocates memory for the number of buffers specified by parameter *NumBuffers* on a specified alignment boundary. The buffers are the correct size for the camera currently attached to the board.

Parameters

Board

Board to handle.

pBufArray

A returned pointer to a structure that holds all acquisition information.

NumBuffers

The number of buffers to allocate.

Alignment

The alignment value, which must be an integer power of 2.

Returns

BI_OK	If successful.
BI_ERROR_CAM_NO_MEM_AVAIL	Lack of memory for buffer allocation. Reduce number of buffers.
BI_ERROR_CAM_FRAMESIZE	Error in inquiring frame size for memory allocation.
BI_ERROR_CAM_STACK_MEM	Lack of memory for error stack.
BI_ERROR_CAM_BUFFERS_NUM	There must be at least two buffers to allocate.
BI_ERROR_CAM_PIXDEPTH	Pixel depth not supported.
BI_ERROR_CAM_UNKNOWN	Undefined error.

Comments

This function allocates memory for *NumBuffers* number of buffers. The buffers are the correct size for the camera currently attached to board pointed to by handle *board*, and will be on a alignment boundary specified by *Alignment*. An array of pointers to the buffers is returned in *pBufArray*. After a successful call to BiBufferAllocCam the returned *pBufArray* pointer will be used for all further accesses to the newly allocated memory.

BiBufferFree should be called to free up resources used by BiBufferAllocAlignedCam.

5.10 BiBufferAllocAligned

Prototype BIRC BiBufferAllocAligned(Bd *Board*, PBIBA *pBufArray*, BFU32 *XSize*, BFU32 *YSize*, BFU32 *PixDepth*, BFU32 *NumBuffers*, BFSIZET *Alignment*)

Description Allocates memory for the number of buffers specified by parameter *NumBuffers* on a specified alignment boundary. The buffers size is determined by parameters *Xsize*, *YSize* and *PixDepth*.

Parameters

Board

Board to handle.

pBufArray

A returned pointer to a structure that holds all acquisition information.

XSize

Width of image in pixels.

YSize

Height of image in lines.

PixDepth

Depth of pixels in bits.

NumBuffers

The number of buffers to allocate.

Alignment

The alignment value, which must be an integer power of 2.

Returns

BI_OK	If successful.
BI_ERROR_NO_MEM_AVAIL	Lack of memory for buffer allocation. Reduce number of buffers.
BI_ERROR_FRAMESIZE	Error in inquiring frame size for memory allocation.
BI_ERROR_STACK_MEM	Lack of memory for error stack.
BI_ERROR_BUFFERS_NUM	There must be at least two buffers to allocate.
BI_ERROR_PIXDEPTH	Pixel depth not supported.

Comments

This function provides a means of allocating memory, on a alignment boundary, for bufin when the camera file is not available and BiBufferAllocAlignedCam can not be used. This is the case when reading a saved image from disk. When reading a image from disk, the camera file is independent to this operation.

The following shows an example of how to use BiBufferAllocAligned:

```
BiBrdOpen// open board
BiDiskParamRead// Get parameters for BiBufferAllocAligned
BiBufferAllocAligned// Allocate memory
BiDiskBufRead// Read in image to memory

// Display and/or process image

BiBufferFree// De-allocate memory
BiBrdClose// Close the board
```

After a successful call to BiBufferAllocAligned the returned *pBufArray* pointer will be used for all further accesses to the newly allocated memory.

BiBufferFree should be called to free up resources used by BiBufferAllocAligned.

BufIn Sequence Capture Management

Chapter 6

6.1 Introduction

This group of functions will provide the means to manage a sequence capture application. The functions in this chapter will allow the user to overwrite the default settings of BiSeqAqSetup, give the ability to wait for the sequence to complete, allow the user to issue commands, for instance stop acquisition, check error status, notify when a frame has been acquired and what frame is currently being acquired.

6.2 BiSeqParameters

Prototype BIRC BiSeqParameters(Bd *Board*, PBIBA *pBufArray*, BFU32 *StartFrame*, BFU32 *NumFrames*, BFU32 *SkipFrames*)

Description Modifies the default behavior of the sequence capture system.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

StartFrame

The first buffer to start acquiring into.

NumFrames

The number of frames to acquire.

SkipFrames

The number of frames to skip between captured images.

Returns

BI_OK	If successful.
BI_ERROR_START_INVALID	Start frame is greater than number of buffers allocated.
BI_ERROR_TOMANY_FRAMES	Number of frames to collect is greater then buffers allocated.
BI_ERROR_START_COMBO	Start frame plus the number of frames is greater than buffers allocated.
BI_ERROR_NUM_FRAMES	Must acquire at least 2 frames for sequence capture.

Comments This function is used to modify the default behavior of the sequence capture system. By default, after a call to BiSeqAqSetup, acquisition will being capturing into the first buffer and capture every frame until all the buffers are full. BiSeqParameters can specify which buffer is to be the first, how many frames are to be acquired, how many frames are to be skipped between captures.

6.3 BiSeqWaitDone

Prototype BIRC BiSeqWaitDone(Bd *Board*, PBIBA *pBufArray*, BFU32 *TimeOut*)

Description Does an efficient wait for the sequence to be completely captured.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

TimeOut

Number of milliseconds to wait for the sequence to be acquired before returning with a time-out error. Set to INFINITE to never time-out.

Returns

BI_OK	If successful.
BI_ERROR_WAIT_TIME-OUT	Timed out while waiting for sequence acquisition
BI_ERROR_WAIT_FAILED	WaitForSingleObject failed.
BI_AQ_ABORTED	Sequence acquisition was aborted.
BI_AQ_STOPPED	Sequence acquisition was stopped.

Comments This function efficiently waits for the sequence to be completed. If the sequence is not complete by the time specified by the *TimeOut* parameter, the function returns with BI_ERROR_WAIT_TIMEOUT. The *TimeOut* parameter is in milliseconds or can be INFINITE to never time-out. This function also returns if the sequence capture is killed, aborted or stopped by the BiSeqControl function.

6.4 BiSeqControl

Prototype BIRC BiSeqControl(Bd *Board*, PBIBA *pBufArray*, BFU32 *Command*, BFU32 *Options*)

Description Controls the sequence acquisition system.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Command

Acquisition command to initiate:

BISTART - Starts sequence acquisition.
BISTOP - Stops sequence acquisition after the current frame has been acquired.
BIPAUSE - Pauses sequence acquisition after the current frame has been acquired.
BIRESUME - Resumes sequence acquisition after a pause command.
BIABORT - Stops sequence acquisition immediately. Does not wait for the current frame to be acquired.

Options

Sequence control options for sequence capture are:

BiWait - wait for the current command to complete.
BiAsync - as soon as the command is issued return.

Returns

BI_OK	If successful.
BI_ERROR_CNTL_UNKNOWN	Unknown command parameter being passed to function.
BI_ERROR_START_TIMEOUT	Timed out waiting for acquisition to start.
BI_ERROR_STOP_TIMEOUT	Timed out waiting for acquisition to stop.
BI_ERROR_PAUSE_TIMEOUT	Timed out waiting for acquisition to pause.
BI_ERROR_RESUME_TIMEOUT	Timed out waiting for acquisition to resume.
BI_ERROR_ABORT_TIMEOUT	Timed out waiting for acquisition to abort.

BI_ERROR_CICONAQCMD_FREEZE	The CiConAqCommand failed trying to issue a freeze.
BI_ERROR_CICONAQSTATUS	Could not determine the acquisition status.
BI_ERROR_INVALIDAQSTATUS	The acquisition status was an invalid type.
BI_ERROR_INVALID_CMD_SEQ	A invalid sequence of control commands was used. For example, a stop was issued after a stop.
BI_ERROR_CICONAQCMD_ABORT	The CiConAqCommand failed trying to issue a abort.

Comments

This function can only be called after a successful call to BiSeqAqSetup.

BI_ERROR_INVALID_CMD_SEQ is a error that will be returned if there is a invalid sequence of commands. An example of that would be if the stop command was called twice in a row. The first stop command would stop acquisition and the second stop command would return this error because acquisition has already been stopped. The following are valid command sequences: A start then stop or abort. Two of the same commands issued one after the other, i.e. pause after a pause. A start after anything other than an stop or abort. A pause after anything but a start or resume.

It is recommended to call this function with an Options parameter of BiWait. BiAsync should only be used when BiWait cannot be used.

6.5 BiSeqErrorWait

Prototype BIRC BiSeqErrorWait(Bd *Board*, PBIBA *pBufArray*)

Description Efficiently waits for an error to occur. Returns immediately if one has occurred since the function was last called.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	If successful.
BI_CLEANUP	Warning that BiSeqCleanUp killed the BiSeqError-Wait function.
BI_ERROR_ACQUISITION	A acquisition error has occurred, check the error stack for the specific error.

Comments This function will return only when an acquisition error has occurred. This function is designed to be in a separate thread that needs to be woken up whenever an error occurs. Once woken up, the error thread can take the appropriate action. This function will let the user know that a acquisition error has occurred, the user will then have to check the error stack with the BiSeqErrorCheck and BiErrorShow functions to view the specific error. This function will end when the system is cleaned up through the calling of the BiSeqCleanUp function. When BiSeqCleanUp ends BiSeqErrorWait a BI_CLEANUP warning will be returned. Warnings can be ignored by comparing the returned value to BI_WARNINGS. If the returned value is greater than BI_WARNINGS the return is a warning. If the returned value is less than BI_WARNINGS the return is a error.

6.6 BiSeqErrorCheck

Prototype BIRC BiSeqErrorCheck(Bd *Board*, PBIBA *pBufArray*)

Description Checks the error stack for any errors that have occurred.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	If no errors have occurred.
The error on the error stack.	If an error has occurred.

Comments This function checks to see if any acquisition errors have occurred. It performs a similar function to BiSeqErrorWait except that it only checks for errors, it does not wait for an error. This function does not require the use of a separate error thread. For single threaded applications, this function can be called from time to time to check for error conditions. If no errors are on the error stack BI_OK will be returned, else the error will be returned. To view the error, pass the returned value to the BiErrorShow function as follows:

```
Error = BiSeqErrorCheck(Board, pBufArray);
if (Error != BI_OK) BiErrorShow(Board, Error);
```

6.7 BiSeqStatusGet

Prototype BIRC BiSeqStatusGet(Bd *Board*, PBIBA *pBufArray*, PBFU32 *Frame*)

Description Checks to see what frame is currently being acquired

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Frame

The number of the frame currently being acquired.

Returns

BI_OK	In all cases.

Comments This function checks to see what frame is currently being acquired and returns the frame number on parameter *Frame*.

6.8 BiSeqWaitDoneFrame

Prototype BIRC BiSeqWaitDoneFrame(Bd *Board*, PBIBA *pBufArray*, BFU32 *TimeOut*)

Description Efficiently waits for a frame to be acquired. Once a complete frame has been acquired the function returns.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

TimeOut

Number of milliseconds to wait for the sequence to be acquired before returning with a time-out error. Set to INFINITE to never time-out.

Returns

BI_OK	A frame has been acquired.
BI_ERROR_FRAME_TIME-OUT	Timed out waiting for a frame.
BI_ERROR_FRAME_FAILED	Error while waiting for frame.
BI_CANCEL_FRAME_ DONE	BiSeqCleanUp killed the BiSeqWaitDoneFrame function.
BI_FRAME_STOP	Sequence acquisition was stopped.
BI_FRAME_ABORT	Sequence acquisition was aborted.

Comments This function efficiently waits for frames to be acquired into memory. This function will wait until a frame has been acquired or BiSeqCleanUp has been called. BiSeq-CleanUp ends this function with a return of BI_CANCEL_FRAME_DONE warning. Warnings can be ignored by comparing the returned value to BI_WARNINGS. If the returned value is greater than BI_WARNINGS the return is a warning. If the returned value is less than BI_WARNINGS the return is a error.

6.9 BiSeqBufferStatus

Prototype BIRC BiSeqBufferStatus(Bd *Board*, PBIBA *pBufArray*, BFU32 *BufferNumber,* PBiSeqInfo *BufferInfo*)

Description Returns the frame count and time stamp for a specific buffer.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

BufferNumber

The buffer that information will be returned for.

BufferInfo

A structure that contains the information for the buffer BufferNumber. The structure contains the frame count and the time stamp for the last image acquired into the buffer.

Returns

BI_OK	If no errors have occurred.
BI_ERROR_BUF_STAT	BiSeqAqSetup needs to be called before this function can be called.

Comments This function returns information about a buffer specified by the parameter *BufferNumber*. BufferInfo is the return structure that contains the buffer information. The structure contains the frame count and time stamp of the buffer.

The frame count will be the number of frames acquired including a partial frame from a overflow or hardware exception. The frame count can be used to determine which frame was missed if an overflow or hardware exception occurred during acquisition. An example would be the acquisition of frame number 100, during acquisition for frame number 101 a overflow occurs. When the overflow occurs, the acquisition will be reset and the partial image will be overwritten with the new one. The frame count will also be incriminated, indicating that another frame has been acquired. The next count will be 102 for the newly acquired frame. The user will see a frame count of 100, 102, 103, etc. The gap will show that an overflow or hardware exception occurred on frame 101.

The time stamp includes the year, year day, month, day, hour, minute, second and milli-second of the time that the frame finishes DMA and is in memory.

6.10 BiSeqBufferStatusClear

Prototype BIRC BiSeqBufferStatusClear(Bd *Board*, PBIBA *pBufArray*)

Description Initialize the buffer information to zero.

Parameters *Board*

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	In all cases.

Comments This function initializes the frame count and time stamp information for all buffers to zero.

BufIn Circular Capture Management

Chapter 7

7.1 Introduction

This group of functions will provide the means to manage a circular capture application. The functions in this chapter will allow the user to overwrite the default settings of BiCircAqSetup, allow the user to issue commands, for instance stop acquisition, check error status, notify when a frame has been acquired and what frame is currently being acquired.

7.2 BiCirControl

Prototype BIRC BiCirControl(Bd *Board*, PBIBA *pBufArray*, BFU32 *Command*, BFU32 *Options*)

Description Controls the circular acquisition system.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Command

Acquisition command to initiate:

BISTART - Starts circular acquisition.
BISTOP - Stops circular acquisition after the current frame has been acquired.
BIPAUSE - Pauses circular acquisition after the current frame has been acquired.
BIRESUME - Resumes circular acquisition after a pause command.
BIABORT - Stops circular acquisition immediately. Does not wait for the current frame to be acquired.

Options

Circular control options for circular capture are:

BiWait - wait for the current command to complete.
BiAsync - as soon as the command is issued return.

Returns

BI_OK	If successful.
BI_ERROR_CIR_CNTL_UNKNOWN	Unknown command parameter being passed to function
BI_ERROR_CIR_RESUME_P	Tried to issue a resume after a start. Resume should be used after a pause to begin acquisition again.
BI_ERROR_CIR_PAUSE	Tried to pause while acquisition is stopped or aborted. Pause can only be used while acquiring.
BI_ERROR_CIR_RESUME	Tried to issue a resume after a stop or abort. Resume should be used after a pause to begin acquisition again.

BI_ERROR_CIR_PAUSE_START	Tried to issue a start from a pause. Use resume to continue acquisition after a pause.
BI_ERROR_CIR_START_TIMEOUT	Timed out waiting for acquisition to start.
BI_ERROR_CIR_STOP_TIMEOUT	Timed out waiting for acquisition to stop.
BI_ERROR_CIR_PAUSE_TIMEOUT	Timed out waiting for acquisition to pause.
BI_ERROR_CIR_RESUME_TIMEOUT	Timed out waiting for acquisition to resume.
BI_ERROR_CIR_ABORT_TIMEOUT	Timed out waiting for acquisition to abort.
BI_ERROR_CIR_CICONAQCMD_ FREEZE	The CiConAqCommand failed trying to issue a freeze.
BI_ERROR_CIR_CICONAQSTATUS	Could not determine the acquisition status.
BI_ERROR_CIR_INVALIDAQSTATUS	The acquisition status was an invalid type.
BI_ERROR_CIR_INVALID_CMD_SEQ	A invalid sequence of control commands was used. For example, a stop was issued after a stop.
BI_ERROR_CIR_CICONAQCMD_ ABORT	The CiConAqCommand failed trying to issue a abort.

Comments

This function can only be called after a successful call to BiCircAqSetup.

BI_ERROR_INVALID_CMD_SEQ is a error that will be returned if there is a invalid sequence of commands. An example of that would be if the stop command was called twice in a row. The first stop command would stop acquisition and the second stop command would return this error because acquisition has already been stopped. The following are valid command sequences: A start then stop or abort. Two of the same commands issued one after the other, i.e. pause after a pause. A start after anything other than an stop or abort. A pause after anything but a start or resume.

7.3 BiCirErrorWait

Prototype BIRC BiCirErrorWait(Bd *Board*, PBIBA *pBufArray*)

Description Efficiently waits for a Bi error to occur. Returns immediately if one has occurred since the function was last called.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	Error has occurred.
BI_CIR_CLEANUP	Warning that BiCircCleanup killed the BiCirError-Wait function.
BI_ERROR_CIR_ACQUISI-TION	A acquisition error has occurred, check the error stack for the specific error.

Comments This function will return only when a circular acquisition error has occurred. This function is designed to be in a separate thread that needs to be woken up whenever a error occurs. Once woken up, the error thread can take the appropriate action. This function will let the user know that a acquisition error has occurred, the user will then have to check the error stack with the BiCirErrorCheck and BiErrorShow functions to view the specific error. This function will end when the system is cleaned up through the BiCircCleanUp function. When BiCircCleanUp ends BiCirErrorWait a BI_CIR_CLEANUP warning will be returned. Warnings can be ignored by comparing the returned value to BI_WARNINGS. If the returned value is greater than BI_WARNINGS the return is a warning. If the returned value is less than BI_WARNINGS the return is a error.

7.4 BiCirErrorCheck

Prototype BIRC BiCirErrorCheck(Bd *Board*, PBIBA *pBufArray*)

Description Checks the Bufin error stack for any errors that have occurred.

Parameters *Board*

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Returns

BI_OK	If no errors have occurred.
The error on the error stack.	If an error has occurred.

Comments This function checks to see if any acquisition errors have occurred. It performs a similar function to BiCirErrorWait except that it only checks for errors, it does not wait for an error. This function does not require the use of a separate error thread. For single threaded applications, this function can be called from time to time to check for error conditions.

Calling this function will automatically remove an error from the Bufin error stack.

7.5 BiCirWaitDoneFrame

Prototype BIRC BiCirWaitDoneFrame(Bd *Board*, PBIBA *pBufArray*, BFU32 *Timeout*, PBiCirHandle *CirHandle*)

Description Waits until there is a newly filled buffer. Returns when the buffer is filled, stopped, aborted, error occurs or when the clean up function is called.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Timeout

Number of milliseconds to wait for the signal to occur before returning with a time-out error. Set to INFINITE to never time-out.

CirHandle

Returned pointer to the circular handle of the buffer that was just captured. The handle includes information about the frame that was captured including, a pointer to the image data buffer, the frame count, the time stamp and the buffer number.

Returns

BI_OK	If successful.
BI_CIR_STOPPED	Warning that circular acquisition has been stopped.
BI_CIR_ABORTED	Warning that circular acquisition has been aborted.
BI_ERROR_CIR_WAIT_TIMEOUT	Wait for multiple objects timed out.
BI_ERROR_CIR_WAIT_FAILED	Wait for multiple objects failed.
BI_ERROR_QEMPTY	The queue is empty.

Comments This function efficiently waits until there is a newly filled buffer. Every time a buffer is filled, it is added to the buffer queue. Whenever this function is called, the oldest buffer in the queue is removed. The only time this function does not return, is if the error mode is BiErStop and the system is out of buffers marked BIAVAILABLE. When a buffer is removed from the queue, its status changes from BIFRESH to BINEW. Once a buffer has BINEW status, it is the responsibility of the application to change the buffers status to BIAVAILABLE. There is no difference between BIFRESH and BINEW status except that the system acknowledges the buffer has been removed from the

queue. The *Timeout* parameter determines how long the function will wait before returning a time-out error. This parameter is given in milliseconds and can be set to INFINITE to never time-out.

The CirHandle structure provides information about the frame that was just acquired. The following entries make up the CirHandle structure:

pBufData - A pointer to the buffer in memory where the image has been acquired.

FrameCount - The number of frames that have been acquired. This number includes frames that have been missed due to overflows. If there is a gap in the frame count the number missing is the frame that is missing. An example would be a frame count of 1, 2, 4, 5,...n. In this case frame 3 overflowed and was overwritten by frame number 4.

TimeStamp - The time that the image finished acquisition into memory. This time is taken from the clock on the host computer. The time stamp is accurate to +/- 20mS.

HiResTimeStamp - A high-resolution time stamp of when the image finished acquisition into memory. This time stamp uses the CPU clock to determine time. Hence, the faster the CPU the more accurate the time stamp. Using a modern day CPU the time stamp should be accurate to at least +/- 1mS. It is recommended that the user benchmark the time stamp for their particular system.

BufferNumber - The buffer number that the image was acquired into.

pNode - A pointer to the node in the list of buffers.

7.6 BiCirStatusSet

Prototype BIRC BiCirStatusSet(Bd *Board*, PBIBA *pBufArray*, BiCirHandle *CirHandle*, BFU32 *Status*)

Description Sets the status for the *CirHandle* returned by BiCirWaitDoneFrame.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

CirHandle

Pointer to the circular handle of a buffer returned by BiCirWaitDoneFrame.

Status

The status to set the buffer too. The only options to the user are BIAVAILABLE and BIHOLD. All other statuses are marked by the circular system. If a buffer is marked BIAVAILABLE it can then be acquired into by the circular system. If a buffer is marked BIHOLD, it will not be acquired into until the user marks it as BIAVAILABLE.

Returns

BI_OK	If successful.
BI_ERROR_MARK_STAT	Tried to change to a status to something other than BIAVAILABLE and BIHOLD.
BI_ERROR_MARK_HOLD	Unable to change buffer status to hold. Current buffer being acquired into was to close to the hold buffer to safely change status.
BI_ERROR_CHAIN_DISABLE	An error occurred in the low level driver while trying to hold a buffer.
BI_ERROR_CHAIN_ENABLE	An error occurred in the low level driver while trying to make a buffer available.
BI_ERROR_AVAIL_QUEUE_FULL	The buffer could not be stored on the available queue because it was full.
BI_ERROR_ON_AVAIL_QUEUE	Unable to change buffer status, because the buffer is on the available queue.
BI_ERROR_STATUS_STATE	Couldn't determine the current status of the buffer.

BI_ERROR_BUFFER_RANGE	Too many buffers are being held. Could not determine a safe buffer range where the buffer can be held.
BI_ERROR_STATUS_MUTEX_TIMEOUT	Timed out waiting for the buffer status mutex.
BI_ERROR_STATUS_MUTEX_ABANDONED	The buffer status mutex was not released properly.
BI_ERROR_RAVEN_HOLD	Holding buffers with the Raven is not supported.

Comments

This function changes the status of the buffer designated by *CirHandle*. The status can be changed to BIAVAILABLE or BIHOLD only. An error will be returned if the status is anything but BIAVAILABLE or BIHOLD.

This function is thread safe, meaning that if this function is called from two different threads only one thread will be allowed into the function at a time. The first thread to call the function will fully execute the function before the call from the second thread is allow to enter the function. This is helpful for the user because there is no need for the user to provide thread synchronization (mutex, semaphore, ...) when using this function in two or more threads.

When holding a buffer or making a buffer available from a hold, the system compares the buffer number to change the status of, to the buffer the framegrabber is DMAing too. If the framegrabber is currently DMAing to the buffer the user wants to change the status of, or the two buffers behind the buffer the user want to change the status of, the BI_ERROR_MARK_HOLD error will be returned. Otherwise the buffer's status will be changed.

7.7 BiCirStatusGet

Prototype BIRC BiCirStatusGet(Bd *Board*, PBIBA *pBufArray*, BiCirHandle *CirHandle*, PBFU32 *Status*)

Description Returns the current status for the *CirHandle* returned by BiCirWaitDoneFrame.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

CirHandle

Pointer to the circular handle of a buffer returned by BiCirWaitDoneFrame.

Status

The returned status of the buffer referred to by *CirHandle*. The returned value can be one of BIFRESH, BINEW, BIAVAILABLE, BIQUEUED or BIHOLD. A BIFRESH buffer would be a buffer that was just filled and is fresh to the buffer queue. A BINEW buffer would be a buffer that has been removed from the buffer queue with the BiCirWait-DoneFrame function. A buffer that is BIAVAILABLE is a buffer that can be acquired into by the circular buffer system. A buffer that is marked BIHOLD is a buffer that can not be acquired into until the user marks it as BIAVAILABLE. A buffer marked BIQUEUED is on the available queue and will be made available by the circular buffer system.

Returns

BI_OK	In all cases.

Comments

7.8 BiCirBufferStatusSet

Prototype BIRC BiCirBufferStatusSet(Bd *Board*, PBIBA *pBufArray*, BFU32 *BufferNumber*, BFU32 *Status*)

Description Sets the status for the buffer specified by *BufferNumber*.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

BufferNumber

The number of the buffer to set status.

Status

The status to set the buffer too. The only options to the user are BIAVAILABLE and BIHOLD. All other statuses are marked by the circular system. If a buffer is marked BIAVAILABLE it can then be acquired into by the circular system. If a buffer is marked BIHOLD, it will not be acquired into until the user marks it as BIAVAILABLE.

Returns

BI_OK	If successful.
BI_ERROR_MARK_STAT	Tried to change to a status to something other than BIAVAILABLE and BIHOLD.
BI_ERROR_MARK_HOLD	Unable to change buffer status to hold. Current buffer being acquired into was to close to the hold buffer to safely change status.
BI_ERROR_CHAIN_DISABLE	An error occurred in the low level driver while trying to hold a buffer.
BI_ERROR_CHAIN_ENABLE	An error occurred in the low level driver while trying to make a buffer available.
BI_ERROR_AVAIL_QUEUE_FULL	The buffer could not be stored on the available queue because it was full.
BI_ERROR_ON_AVAIL_QUEUE	Unable to change buffer status, because the buffer is on the available queue.
BI_ERROR_STATUS_STATE	Couldn't determine the current status of the buffer.

BI_ERROR_BUFFER_RANGE	Too many buffers are being held. Could not determine a safe buffer range where the buffer can be held.
BI_ERROR_STATUS_MUTEX_TIMEOUT	Timed out waiting for the buffer status mutex.
BI_ERROR_STATUS_MUTEX_ABANDONED	The buffer status mutex was not released properly.
BI_ERROR_RAVEN_HOLD	Holding buffers with the Raven is not supported.
BI_ERROR_CIR_BAD_BUFFER_NUM	Trying to hold a buffer number that doesn't exist.

Comments

Please refer to the BiCirStatusSet function for comments. The BiCirStatusSet and BiCirBufferStatusSet functions are identical except BiCirStatusSet uses the circular handle and BiCirBufferStatusSet uses the buffer number to determine the buffer that will have its status change.

7.9 BiCirBufferStatusGet

Prototype BIRC BiCirBufferStatusGet(Bd *Board*, PBIBA *pBufArray*, BFU32 *BufferNumber*, PBFU32 *Status*)

Description Returns the current status for the buffer specified by *BufferNumber*.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

BufferNumber

The buffer to get the status of.

Status

The returned status of the buffer referred to by *BufferNumber*. The returned value can be one of BIFRESH, BINEW, BIAVAILABLE, BIQUEUED or BIHOLD. A BIFRESH buffer would be a buffer that was just filled and is fresh to the buffer queue. A BINEW buffer would be a buffer that has been removed from the buffer queue with the BiCirWaitDoneFrame function. A buffer that is BIAVAILABLE is a buffer that can be acquired into by the circular buffer system. A buffer that is marked BIHOLD is a buffer that can not be acquired into until the user marks it as BIAVAILABLE. A buffer marked BIQUEUED is on the available queue and will be made available by the circular buffer system.

Returns

BI_OK	In all cases.
BI_ERROR_CIR_BUFFER_NUM	Tried getting the status of a buffer that doesn't exist.

Comments

7.10 BiBufferQueueSize

Prototype BIRC BiBufferQueueSize(Bd Board, PBIBA pBufArray, PBFU32 NumFrames)

Description Returns the number of buffers stored on the buffer queue.

Parameters

Board

Board to handle.

pBufArray

A pointer to a structure that holds all acquisition information.

NumFrames

The number of buffers stored on the buffer queue.

Returns

BI_OK	If successful.

Comments This function returns the number of buffers stored on the buffer queue. As buffers are acquired, the buffer count increments. Everytime the user receives a buffer from BiCirWaitDoneFrame, the count decrements.

BufIn Trigger Functions

Chapter 8

8.1 Introduction

The trigger functions provide the ability to inquire the trigger mode of the current camera file, to change the trigger mode from software, and to force a software trigger.

8.2 BiTrigModeSet

Prototype BIRC BiTrigModeSet(Bd *Board*, BFU32 *TriggerMode*, BFU32 *TriggerPolarity*)

Description Sets the trigger mode and polarities for both acquisition engines.

Parameters *Board*

Handle to board.

TriggerMode

The trigger modes can be one of the following:

BiTrigFreeRun - no trigger is used, board free runs.
BiTrigContinuousData - for continuous data sources.
BiTrigOneShot - one shot mode for asynchronously resettable cameras. DMA engines J and K are run by the same trigger(s).
BiTrigOneShotJbyAandKbyB - one shot mode for asynchronously resettable cameras. DMA engine J is run by trigger A and engine K by trigger B.
BiTrigOneShotStartStop - start/stop mode for line scan cameras. DMA engines J and K are run by the same trigger(s).
BiTrigOnShotSSJbyAandKbyB - start/stop mode for line scan cameras. DMA engine J is run by trigger A and engine K by trigger B.
BiTrigOneShotStartAStopB - start/stop mode for line scan cameras. Trigger A resets DMA engine J and starts acquiring one frame. Trigger B will terminate acquisition of the frame.
BiTrigAqCmd - triggered acquired command mode for non-resettable cameras. DMA engines J and K are run by the same trigger(s).
BiTrigAqCmdJbyAandKbyB - triggered acquired command mode for non-resettable cameras. DMA engine J is run by trigger A and engine K by trigger B.
BiTrigAqCmdFreezeCmd - start/stop trigger for non-resettable cameras. DMA engines J and K are run by the same trigger(s).
BiTrigAqCmdFrzCmdJbyAandKFree - start/stop trigger for non-resettable cameras. DMA engine J is run by trigger A and engine K free runs.
BiTrigOneShotSelfTrig - self triggering one shot mode.

TriggerPolarity

Polarity for trigger A and B:

BiTrigAssertedHigh - Trigger A and B are asserted on rising edge.
BiTrigAssertedLow - Trigger A and B are asserted on falling edge.
BiTrigAHighBLow - Trigger A is asserted on rising edge, B on falling edge.
BiTrigALowBHigh - Trigger A is asserted on falling edge, B on rising edge.

Returns

BI_OK	If successful.
BI_ERROR_UNKNOWN_TRIG_SET	Unknown trigger mode or polarity parameter
BI_ERROR_BOARDPTR_TRIG_SET	Bad board pointe.
BI_ERROR_BAD_BOARD_PARA	Parameter for function valid but not for the specified board.

Comments

This function works in conjunction with the camera configurations files. It is important to understand that not all cameras support all triggering modes and not all trigger modes are supported by a particular board. Usually a particular camera will only support one or two triggering modes. Furthermore, a different camera configuration file is usually needed for each triggering mode. For example, a camera will almost always have a free running configuration file, useful for set up and offline testing. A camera may also have a one shot file, which would be used in time-critical applications. You cannot usually put the board, set up by the free running file, into one shot mode because the latter mode requires special triggering signals to be sent to the camera. However, you can put a board set up by a one shot file, into self triggering one shot mode. This is useful for camera set up and system debugging.

This function only controls how the board is vertically triggered. Vertical triggers cause the board to acquire a whole frame from a area camera or a number of lines from a line scan camera.

This function is only needed to change from one type of trigger mode to another on the fly. If the camera file is set up for one-shot mode, there is no need to set the trigger mode to one-shot. By default the Bi API will acquire from the camera based on what the camera file is set for. If the user wants to change the trigger mode from that of the camera file, this function can be used to do that.

See Table 8-1 and Table 8-2 for how the various triggering modes work on the Road Runner/R3 and R64 respectively.

Table 8-1 Trigger Modes Road Runner/R3

Trigger Modes	Trigger A asserts	Trigger B asserts
BiTrigFreeRun	No effect	No effect
BiTrigContinuousData	Engine J starts acquiring continuous data	N/A
BiTrigOneShot	Engine J resets its camera and acquires one frame	No effect
BiTrigOneShotJbyAandKbyB	N/A	N/A

Table 8-1 Trigger Modes Road Runner/R3

Trigger Modes	Trigger A asserts	Trigger B asserts
BiTrigOneShotStartStop	Engine J resets its camera and starts acquiring one frame. Frame is terminated on trigger deassertion edge.	No effect
BiTrigOneShotSSJbyAandKbyB	N/A	N/A
BiTrigOneShotStartAStopB	Engine J resets its camera and starts acquiring one frame	Engine J terminates acquisition of the frame
BiTrigAqCmd	Engine J command latches	No effect
BiTrigAqCmdJbyAandKbyB	N/A	N/A
BiTrigAqCmdFreezeCmd	Engine J command latches	Freeze Issued
BiTrigAqCmdFrzCmdJbyAandK-Free	N/A	N/A
BiTrigOneShotSelfTrig	N/A	N/A

Table 8-2 Trigger Modes R64

Trigger Modes	Trigger A asserts
BiTrigFreeRun	No effect
BiTrigContinuousData	Engine J starts acquiring continuous data
BiTrigOneShot	Engine J resets its camera and acquires one frame
BiTrigOneShotJbyAandKbyB	N/A
BiTrigOneShotStartStop	Engine J resets its camera and starts acquiring one frame. Frame is terminated on trigger deassertion edge.
BiTrigOneShotSSJbyAandKbyB	N/A
BiTrigOneShotStartAStopB	N/A
BiTrigAqCmd	Engine J command latches

Table 8-2 Trigger Modes R64

Trigger Modes	Trigger A asserts
BiTrigAqCmdJbyAandKbyB	N/A
BiTrigAqCmdFreezeCmd	N/A
BiTrigAqCmdFrzCmdJbyAandK-Free	N/A
BiTrigOneShotSelfTrig	No Effect

8.3 BiTrigModeGet

Prototype BIRC BiTrigModeGet(Bd *Board*, PBFU32 *TriggerMode*, PBFU32 *TriggerPolarity*)

Description Gets the current trigger mode and polarities.

Parameters *Board*

Handle to board.

TriggerMode

Returns the current trigger mode:

BiTrigFreeRun - no trigger is used, board free runs.
BiTrigContinuousData - for continuous data sources.
BiTrigOneShot - one shot mode for asynchronously resettable cameras. DMA engines J and K are run by the same trigger(s).
BiTrigOneShotJbyAandKbyB - one shot mode for asynchronously resettable cameras. DMA engine J is run by trigger A and engine K by trigger B.
BiTrigOneShotStartStop - start/stop mode for line scan cameras. DMA engines J and K are run by the same trigger(s).
BiTrigOnShotSSJbyAandKbyB - start/stop mode for line scan cameras. DMA engine J is run by trigger A and engine K by trigger B.
BiTrigOneShotStartAStopB - start/stop mode for line scan cameras. Trigger A resets DMA engine J and starts acquiring one frame. Trigger B will terminate acquisition of the frame.
BiTrigAqCmd - triggerd acquired command mode for non-resettable cameras. DMA engines J and K are run by the same trigger(s).
BiTrigAqCmdJbyAandKbyB - triggerd acquired command mode for non-resettable cameras. DMA engine J is run by trigger A and engine K by trigger B.
BiTrigAqCmdFreezeCmd - start/stop trigger for non-resettable cameras. DMA engines J and K are run by the same trigger(s).
BiTrigAqCmdFrzCmdJbyAandKFree - start/stop trigger for non-resettable cameras. DMA engine J is run by trigger A and engine K free runs.
BiTrigOneShotSelfTrig - self triggering one shot mode.

TriggerPolarity

Returns the current trigger polarity:

BiTrigAssertedHigh - Trigger A and B are asserted on rising edge.
BiTrigAssertedLow - Trigger A and B are asserted on falling edge.
BiTrigAHighBLow - Trigger A is asserted on rising edge, B on falling edge.
BiTrigALowBHigh - Trigger A is asserted on falling edge, B on rising edge.

Returns

BI_OK	If successful.
BI_ERROR_BOARDPTR_TRIG_GET	Unknown Board Type.

Comments

This function returns the current state of the trigger circuitry. See the function BiTrigModeSet for a complete description of the modes.

8.4 BiTrigForce

Prototype BIRC BiTrigForce(Bd *Board*, BFU32 *Mode*)

Description Performs a software trigger on the specified board.

Parameters *Board*

Handle to board.

Mode

How to assert the trigger. The following are the only valid means to assert a trigger:

BiTrigAssertTrigA - Trigger A is given a rising edge.
BiTrigAssertTrigB - Trigger B is given a rising edge.
BiTrigDeassertTrigA - Trigger A is given a falling edge.
BiTrigDeassertTrigB - Trigger B is given a falling edge.

Returns

BI_OK	If successful.
BI_ERROR_UNKNOWN_MODE	Unknown Trigger mode.
BI_ERROR_BRD_TRIG_FORCE	Unknown Board Type.

Comments This function triggers the board. The mode parameter can be either asserted or de-asserted for trigger A and trigger B. De-assertion of a trigger can only be used with modes that require a stop trigger such as start/stop.

BufIn Disk I/O Functions

Chapter 9

9.1 Introduction

The I/O functions provide the ability to save the images acquired by a sequence or circular application to disk. The write functions provide the ability to choose which images to save to disk from sequence or circular application. BufIn supports images being saved to disk in BMP, TIFF, AVI and RAW data formats.

BufIn provides the ability to read images from disk and into memory. The read function can only reliably read images saved to disk by BufIn's write function. BufIn also provides a helper function for reading images from disk, which will read the pertinent information from the header of the image. This information should be used in the read function.

9.2 BiDiskBufWrite

Prototype BIRC BiDiskBufWrite(Bd *Board*, PBIBA *pBufArray*, BFU32 *FileType*, BFU32 *FirstBuf-Number*, BFU32 *NumBufs*, PBFCHAR *FileName*, BFSIZET *FIleNameSize*, BFU32 *Options*)

Description Writes one or more buffers to files on the disk.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

FirstBufNumber

The first buffer to be written to disk. If multiple buffers are to be written to disk, this will specify the first buffer to be written in the series. If only one buffer is to be written, this will be the number of that buffer.

NumBufs

The number of buffers to be written. If this parameter is greater than one, the files will all have the same name with the buffer number appended to the name (e.g. “image0000.BMP”, “image0001.BMP”, “image0002.BMP”, etc.) The only exception will be with AVI files, in which the function will produce only one file with *NumBufs* number of buffers as a continuous movie.

FileName

The name chosen to save the buffer as. *FileName* should included the full path and the extension. The file name prefix will be appended with the file number. For example “image.bmp” will be saved as “image000000.bmp”.

The extension will determine the type of file create. Acceptable extensions are:

avi - Saves a Windows AVI video file (8 or 24-bit pixels only)
bmp - Saves Windows bitmap format file (8 or 24 bits pixels only)
raw - Saves a raw file (binary copy of in-memory image)
tiff - TIFF file format

FileNameSize

The size in bytes of the *FileName* parameter buffer.

Options

Additional options for saving the buffer to disk.

0 - No additional options used.
SwapRGB - Swap the RGB format to BGR.
Pack32to24Bit - Save 32 bit color XRGB data to 24 bit RGB.
OVERWRITEFILE - If the file already exists, it will be overwritten. Default behavior is to return an error and not overwrite an existing file.

Returns

BI_OK	If successful.
BI_ERROR_UNKNOWN_FILE-TYPE	The file type to save as is unknown.
BI_ERROR_FIRSTBUF_INVALID	The first buffer is larger that the number of buffers.
BI_ERROR_NUMBUFS_ZERO	The number of buffers must be at least 1.
BI_ERROR_TOO_MANY_BUF-FERS	Asking for more buffers to save then allocated.
BI_ERROR_OPEN_BMP_FILE	File for BMP could not be created.
BI_ERROR_BITMAP_HEADER	Error writing BMP header to file.
BI_ERROR_BMP_DATA_WRITE	Error writing BMP data to file.
BI_ERROR_OPEN_AVI_FILE	Error opening AVI File.
BI_ERROR_CREATE_STREAM	Error creating stream.
BI_ERROR_SAVE_OPTIONS	Error with dialog box save options.
BI_ERROR_COMPRESS_STREAM	Error with compressing the stream.
BI_ERROR_AVI_HEADER	Error putting AVI header in the stream.
BI_ERROR_WRITING_AVI_DATA	Error writing the AVI stream.
BI_ERROR_OPEN_RAW_FILE	Error opening the RAW file.
BI_ERROR_RAW_DATA_WRITE	Error writing RAW data to file.
BI_ERROR_OPEN_TIFF_FILE	Error opening TIFF file.
BI_ERROR_WRITING_TIFF_ HEADER	Error writing TIFF header.
BI_ERROR_WRITING_TIFF_DATA	Error writing TIFF data.
BI_ERROR_SWAPRGB	Can only use swap RGB option with color data.
BI_ERROR_MEM_SWAP_RGB	Could not allocate memory for RGB swap buffer.
BI_ERROR_PACK24BIT	Must begin with 32 bit data in order to pack to 24 bit.
BI_ERROR_BUF_POINTER	Invalid buffer pointer

BI_ERROR_FILE_XSIZE	Invalid XSize. The XSize must be greater than zero.
BI_ERROR_FILE_YSIZE	Invalid YSize. The YSize must be greater than zero.
BI_ERROR_WRITE_BITDEPTH	Unknown bit depth. The bit depth must be 8, 10, 12, 14, 16, 24, 32, 36, 42 or 48.
BI_ERROR_WRITE_BMP_BIT-DEPTH	Invalid bit depth for BMP. Bmp supports 8, 24 and 32 bit pixel depths.
BI_ERROR_WRITE_TIF_BITDEPTH	Invalid bit depth for tif. Tif supports 8, 10, 12, 14, 16, 24, 32, 36, 42 and 48 bit pixel depths
BI_ERROR_WRITE_AVI_BIT-DEPTH	Invalid bit depth for AVI. Avi supports 8, 24 and 32 bit pixel depths.
BI_ERROR_WRITE_LOW_MEM	Failed allocating memory.
BI_ERROR_OPEN_TEXTFILE	Failed opening text file to write raw image information.
BI_ERROR_RAW_TEXT_WRITE	Failed writing raw image information to text file.
BI_ERROR_UNKNOWN_WRITE	Unknown error returned from BFIOWrite function.

Comments

This function writes one or more buffers to files on the disk. The file type is specified by the *FileType* parameter. The file types supported are BMP, TIFF, AVI and RAW data. The first buffer to be written is specified by the *FirstBufNumber* parameter. The number of buffers to be written is specified by the *NumBufs* parameter. If *NumBufs* is greater than one, the files will all have the same name with the buffer number appended to the name (e.g. “image0000.BMP”, “image0001.BMP” etc.). The only exception will be with AVI files, which will produce only one file with all the buffers saved as a continuous movie.

The options parameter is provided to manage saving RGB color data. RGB color data will be presented to the frame grabber in all different formats based on the camera and cabling between the camera and framegrabber. To deal with the different formatting of color data there are two options, SwapRGB and Pack32to24Bit.

When using SwapRGB as an option, data will be swapped from RGB to BGR or BGR to RGB, depending on how the data is coming into the framegrabber. This option can be used if the red and blue colors have been switched not displaying colors correctly.

The Pack32to24Bit option provides a means of saving off 32-bit XRGB data packed as 24 bit RGB data. The majority of color data packed by the framegrabber is packed at 32 bit data with the upper 8 bits zeroed out. This option packs this data to 24 bits. This can be useful because some applications will not recognize 32-bit color data with the upper 8 bits being zero. This option provides a means around this problem.

The SwapRGB and Pack32to24Bit options can be ORed together to pack 32-bit data to 24-bit data and swap red and blue. If neither of the options are to be used, the options parameter should be 0.

Currently BiDiskBufWrite only supports single image TIFF format.

9.3 BiDiskBufRead

Prototype BIRC BiDiskBufRead(Bd *Board*, PBIBA *pBufArray*, BFU32 *FirstBufNumber*, BFU32 *NumBufs*, PBFCHAR *FileName*)

Description Reads images from disk into a buffer array.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

FirstBufNumber

Indicates the first buffer, of the buffer array *pBufArray*, that the image on the disk will be read into. If *NumBufs* is greater than one, images will continue to be read off disk and into subsequent buffers in *pBufArray*. In order for multiple images to be read off the disk, they must have the format, "prefixXXXX.ext", where XXXX is a sequential number series.

NumBufs

Specifies the number of buffers to be read into.

FileName

The name of the first image read. Subsequent reads will increment the XXXX portions of the file name until the number of images, specified by *NumBufs*, are read.

Returns

BI_OK	If successful.
BI_ERROR_NUMBUFS	Bad value of NumBufs, must be greater than zero.
BI_ERROR_BUFFER_SIZE_SMALL	Image size is larger that the buffer size being copied to.
BI_ERROR_FILE_OPEN_READ	Error opening file for reading.
BI_ERROR_FILE_READ	Error reading file.
BI_ERROR_FILETYPE_READ_IO	Unknown File type. File type must be raw, BMP, tif, or AVI.
BI_ERROR_MEM_TEMP_BUF_READ_IO	Error allocating memory for temporary BMP buffer.

BI_ERROR_INVALID_NAME	The file name given is invalid, to few characters.
BI_ERROR_PIXDEPTH_READ	Pixel depth not supported.
BI_ERROR_DISK_PARAM_READ	Error returned by BiDiskParamRead function.
BI_ERROR_NUMBER_OF_FRAMES	Number of frames in AVI file exceeds the number of buffers.
BI_ERROR_NO_DECOMPRESS	Decompressing image to dimensions outside of maximum limit.
BI_ERROR_READ_BUF_POINTER	Invalid buffer pointer.
BI_ERROR_READ_XSIZE	Invalid XSize. The XSize must be greater than zero.
BI_ERROR_READ_YSIZE	Invalid YSize. The YSize must be greater than zero.
BI_ERROR_RAW_READ	Reading a raw image file is not supported.
BI_ERROR_UNKNOWN_READ	Unknown error returned from BFIORead function.

Comments

This function reads images from disk into a buffer array. This function is intended to read images that have been saved to disk using BiDiskBufWrite. The user must create the buffers that the images will be read into by allocating memory using the BiBufferAlloc function. The allocation of memory must be done before calling BiDiskBufRead.

This function supports the reading of BMP, AVI, and TIFF. Reading of raw data is not supported by this function.

9.4 BiDiskParamRead

Prototype BIRC BiDiskParamRead(Bd *Board*, PBFCHAR *FileName*, PBFU32 *XSize*, PBFU32 *YSize*, PBFU32 *PixDepth*, PBFU32 *NumFrames*)

Description Reads the format and file information from a file on disk.

Parameters

Board

Handle to board.

FileName

The name of the file to get information from. The file name should have the format "Name.ext".

XSize

Returns the width of the image, specified by *FileName*, in pixels.

YSize

Returns the height of the image, specified by *FileName*, in lines.

PixDepth

Returns the depth of a pixel in bits of the image specified by *FileName*.

NumFrames

Returns the number of frames in the file. This parameter only has meaning for file formats that support multiple images. Currently this parameter will always return 1 unless used with a AVI file.

Returns

BI_OK	If successful.
BI_ERROR_FILETYPE_PARAM	Unknown file type. File type must be raw, BMP, tif, or AVI.
BI_ERROR_MEM_TEMP_BUF_PARAM	Error creating temporary buffer.
BI_ERROR_FILE_OPEN_PARAM	Error opening file. Check file name.
BI_ERROR_READFILE_PARAM	Error reading in file data.
BI_ERROR_RAW_READ_PARAMS	Reading raw file parameters is not supported.
BI_ERROR_UNKNOWN_PARAM	Unknown error returned from BFIOReadParam.

Comments

This function reads the format information from a file on disk. The file to inquire about must be in either BMP, TIFF, or AVI format. Reading parameters from a raw file is not supported. This function is intended to be called before BiDiskBufRead. This function will retrieve all the information needed for BiDiskBufRead.

BufIn Status Functions

Chapter 10

10.1 Introduction

This set of functions provides the ability to inquire acquisition status and DLL version. The functions in this section provide the ability to find out if acquisition has been started, stopped, aborted, paused or if clean up has been called. There is also a function to determine how many frames have been captured and missed during an acquisition session.

The version checking function checks the version of the BufIn DLL that is on the runtime machine with the version of the SDK BufIn was built with. The variables BF_SDK_VERSION_MAJOR and BF_SDK_VERSION_MINOR are defined in the header files and should match the values returned from this function at runtime. If there is a miss match, usually the DLL on the runtime machine is not compatible with the application.

The functions in this section can be used with both sequence and circular functions.

10.2 BiControlStatusGet

Prototype BIRC BiControlStatusGet(Bd *Board*, PBIBA *pBufArray*, PBFBOOL *Start*, PBFBOOL *Stop*, PBFBOOL *Abort*, PBFBOOL *Pause*, PBFBOOL *Cleanup*)

Description Outputs the values of the acquisition status flags.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Start

Returns the start status of acquisition. If *Start* is TRUE, the acquisition of image data to buffers has started. If *Start* is FALSE, acquisition of image data has either been stopped, never started, or aborted.

Stop

Returns the stop status of acquisition. If *Stop* is TRUE, acquisition of image data to buffers has been stopped, aborted, or never started. When acquisition is stopped, the last frame is fully acquired, then acquisition is stopped. If *Stop* is FALSE, image data is being acquired.

Abort

Returns the abort status of acquisition. If *Abort* is TRUE, acquisition of image data to buffers has been aborted. When acquisition is aborted, acquisition of data is stopped immediately, not waiting for the last frame to be completely acquired. If *Abort* is FALSE, acquisition has not been aborted.

Pause

Returns the pauses status of acquisition. If *Pause* is TRUE, acquisition of image data to buffers has been paused. If FALSE, acquisition has not been paused.

Cleanup

Returns the clean up status. If *Cleanup* is TRUE, BiSeqCleanUp or BiCircCleanUp has been called. If FALSE, BiSeqCleanUp or BiCircCleanUp has not been called.

Returns

BI_OK	In all cases.

Comments This function returns the acquisition status flags.

10.3 BiCaptureStatusGet

Prototype

BIRC BiCaptureStatusGet(Bd *Board*, PBIBA *pBufArray*, PBFU32 *Captured*, PBFU32 *Missed*)

Description

Outputs the number of frames that have been captured and missed at the moment the function is called.

Parameters

Board

Handle to board.

pBufArray

A pointer to a structure that holds all acquisition information.

Captured

Returns the number of frames that have been captured.

Missed

Returns the number of frames that have been missed.

Returns

BI_OK	In all cases.

10.4 BiDVersion

Prototype BIRC BiDVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)

Description Returns the current version of their corresponding DLL.

Parameters *pMajorVersion*

When the function returns, it contains the major version number. If the highest order bit is set, the DLL is a debug version.

pMinorVersion

When the function returns, it contains the minor version number.

Returns

BI_OK	In all cases.

Comments This function returns the version of the BiD DLL. This function can be used to make sure applications are working with the correct DLLs. The variables BF_SDK_VERSION_MAJOR and BF_SDK_VERSION_MINOR are always defined in the header files of the SDK. These variables can be used in an application to determine what version of the SDK the application was built with. These variables can also be compared to the values returned from the above functions to maintain version consistency between the application and the current DLLs.

The highest order bit (bit 31) will be set in the pMajorVersion parameter upon return if the current DLL is a debug version.

BufIn Error Functions

Chapter 11

11.1 Introduction

The error function provides a means to view error messages returned from function. The error message will be displayed in a dialog box. The message will include the function that failed and a description of the failure.

Note: These functions only work the errors produced by the BufIn functions. There is a separate set of functions for BF level errors.

Note: There are also special error functions that are for use when performing circular acquisition. See Section 6.5, Section 6.6, Section 7.3 and Section 7.4.

11.2 BiErrorShow

Prototype BIRC BiErrorShow(Bd *Board*, BIRC *ErrorNum*)

Description Pops up a dialog displaying a text description of the error given by the *ErrorNum* parameter.

Parameters

Board

Handle to board.

ErrorNum

The error number usually returned by a BI function.

Returns

BI_OK	In all cases.

Comments This function pops up a dialog box displaying the text describing the error passed by the *ErrorNum* parameter. This function also displays the name of the function that failed. This function can optionally be called by an application to visually display error text. The *ErrorNum* parameter will usually be the value returned from a function.

11.3 BiErrorTextGet

Prototype

BIRC BiErrorTextGet(Bd *Board*, BIRC *ErrorNum*, PBFCHAR *ErrorText*, PBFU32 *ErrorTextSize*)

Description

Returns a text description of the error given by the *ErrorNum* parameter.

Parameters

Board

Handle to board.

ErrorNum

The error number to get the error text for.

ErrorText

A user allocated buffer to hold the returned error text.

ErrorTextSize

The size of the *ErrorText* buffer the user has allocated in bytes.

Returns

BI_OK	If successful.
BI_ERROR_BUFFER_TOO_ SMALL	The returned error text will not fit into the user allocated buffer. Allocate more memory for *ErrorText*.
BI_ERROR_ERR_NOT_FOUND	A *ErrorNum* was passed into the function that dose not exist.
BI_ERROR_NULL_TEXT_PTR	The pointer to the *ErrorText* buffer was null.

Comments

This function returns a text description for the error given by the *ErrorNum* parameter. To use this function the user must allocate memory to hold the returned error text, *ErrorText*. The user passes in the size of the allocated buffer with the *ErrorTextSize* parameter. When the function returns, the *ErrorTextSize* parameter will be overwritten with the number of bytes written into the *ErrorText* buffer when the function is successful. If the buffer is to small to accommodate the error text, the *ErrorTextSize* parameter returns the size that the buffer needs to be to hold the complete message. The function will also copy as much of the error message as possible into the buffer when the buffer is to small. For any other errors, the *ErrorText* buffer and *ErrorTextSize* remain unchanged.

11.4 BiErrorList

Prototype BIRC BiErrorList()

Description Generates a file that lists all the error numbers and descriptions for the BI API.

Parameters None.

Returns

BI_OK	If successful.
A non-zero value.	This function will return a non-zero value if it fails.

Comments The function will generate a text file called "BiErrorList.txt" in the current directory. The text file will contain all the error numbers and descriptions of the errors. This list of errors can be helpful when debugging.

Camera Interface (Ci) Introduction

Chapter 12

12.1 Overview

The main purpose of the "Ci" function is to provide a board model agnostic interface to setting up and controlling any BitFlow frame grabber. These function hide the model specific functionality that is seen in the "R2", R64" and "Gn2" APIs. The "CI" functions provide everything needed to set up a board for acquisition and control all of the board's features. However, using these function to set the board up to acquire into more than a single host buffer can be quite complicate at the "Ci" level. We there for recommend that you use the "Bi" functions for anything beyond setting up a single host buffer. The "Bi" functions provide all the major board setup and control functionality in a very simple to use interface.

That said "Ci" API still provides some much needed functionality when use a board in a non-standard way or modifying the boards settings on-the-fly from your application. In this sense the "Ci" layer occupies the middle of the BitFlow API stack. Chances are you will need some of these functions, but probably not very many.

For the most part is fine to mix "Bi" and "Ci" functions. However, the "Bi" functions should be use for opening the board, setting up buffers for acquisition and controlling acquisition. The "Ci" function can then be used to further customize the board.

Ci System Open and Initialization

Chapter 13

13.1 Introduction

The functions described in this chapter are quite simple; the idea is to find the board or boards that you want to work with, then open and optionally initialize them. When you are finished, close the system up, thus cleaning up all resources allocated in the open function.

A normal program would uses these functions, in this order:

```
CiSysBrdFind(CISYS_TYPE_ANY, 0, &Entry);
CiBrdOpen(Entry, hBoard, BFSysInitialize);
// acquisition and processing
CiBrdClose(hBoard);
```

If you want to open two boards, the flow would be as follows:

```
CiSysBrdFind(CISYS_TYPE_ANY,0,&Entry0);// find board 0
CiBrdOpen(Entry0, hBoard0, BFSysInitialize);// open board 0
CiSysBrdFind(CISYS_TYPE_ANY,1,&Entry1);// find board 1
CiBrdOpen(Entry1,hBoard1,BFSysInitialize);// open board 1

// acquisition and processing

CiBrdClose(hBoard0);// close board 0
CiBrdClose(hBoard1);// close board 1
```

The board find functions are used to make sure that you are opening the correct board in a multi-board system. If you have only one board, then the call is trivial.

There is currently two board find functions. The first is CiSysBrdFind, which is used to find boards by board family and board number. The second is CiSysBoardFindSWConnector, which is used to find a board based on its switch settings and connector number (for boards with more than one Virtual Frame Grabber).

The handle return by the function CiBrdOpen is used in all subsequent function calls. If you are using two or more boards, open each board and store each handle in a separate variable. Whenever you want to talk to board X, pass the handle for board X to the function.

There is no need to call CiBrdOpen more than once per process per board. Because this function takes a fair amount of CPU time and allocated resources, we discourage users from repeatedly calling CiBrdOpen and the CiBrdClose in a loop. We recommend opening the board once, when the application starts, and closing it once when the application exits. If you are using a program that has multiple threads, open the board once in the first

main thread and then pass the board handle to every thread that is subsequently created. You must call CiBrdClose for every board that is opened with CiBrdOpen. You should also call CiBrdClose in the same thread the CiBrdOpen was called in.

13.2 Specifying Camera Configuration Files

A camera configuration file is used to initialize a board to work with a specific camera in a specific mode. When a board is opened and initialized, a camera configuration file must be specified. There area a few different methods to specify a camera configuration file. For more information on these methods see section 1.3.

13.3 CiSysBrdEnum

Prototype BFRC CiSysBrdEnum(BFU32 *Type*, BFU32 *Number*)

Description This function can be used to find the number of specific boards in a system. This function is unique to the Ci API.

Parameters *Type*

Board to select:

CISYS_TYPE_R2 – Find all RoadRunners.
CISYS_TYPE_R64 – Find all R64s.
CISYS_TYPE_GN2 - Find Aons, Axions, Cytons and Claxons.
CISYS_TYPE_ANY – Find all model boards installed.

Number

Pointer to the number of boards found in the system.

Returns

CI_OK	If successful.
CISYS_ERROR_UNKNOWN_TYPE	An invalid board handle was passed to the function.

Comments

13.4 CiSysBrdFind

Prototype BFRC CiSysBrdFind(BFU32 *Type,* BFU32 *Number*, PCiENTRY *pEntry*)

Description Finds a the board specified by *Type* on the PCI bus with a given number.

Parameters

Type

The type of board to find:

CISYS_TYPE_R2 – Search for a RoadRunner/R3.
CISYS_TYPE_R64 – Search for a R64/R64e/Karbon/Neon/Alta.
CISYS_TYPE_GN2 - Find Aons, Axions, Cytons and Claxons.
CISYS_TYPE_ANY – Search for a board by number, ignoring the board type.

Number

The number of the board to find. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots, as long as the number of boards remains the same and are in the same PCI slot.

pEntry

A pointer to an empty CiENTRY structure, used to tell the CiBrdOpen function which board to open.

Returns

CI_OK	The board was successfully found.
CISYS_ERROR_NOTFOUND	There is no board with this number.
CISYS_ERROR_UNKNOWN_TYPE	The board type is unknown.
CISYS_ERROR_SYSTEM	Error reading registry.

Comments If you have only one board in your system, set *Number* = 0 and only call this function once. This function can be used to enumerate all of the boards in a system. It can be called repeatedly, incrementing *Number* each time, until the function returns CISYS_ERROR_NOTFOUND.

There is no standard way to correlate the *Number* parameter of this function to the PCI slot number. Every motherboard and BIOS manufacturer has a different scheme. You can use the system configuration utility, SysReg, to determine the relationship between slot number and board *Number*, by setting the board ID switches different for each board in your system and walking through all the installed boards.

When using CISYS_TYPE_ANY for *Type*, the board is found only using the board number. For instance, if board 0 is an R3, board 1 is a Neon and board 2 is an Axion and the following function calls are made with the following results:

```
CiSysBrdFind(CISYS_TYPE_ANY, 0, pEntry); // Returns the R3
CiSysBrdFind(CISYS_TYPE_ANY, 1, pEntry); // Returns the Neon
CiSysBrdFind(CISYS_TYPE_ANY, 2, pEntry); // Returns the Axion
```

13.5 CiSysBoardFindSWConnector

Prototype

BFRC CiSysBoardFindSWConnector(BFU32 *Type,* BFU32 *Switch*, BFU32 *Connector*, PCiENTRY *pEntry*)

Description

Finds a the board/VFG of the given *Type* with the switch set to *Switch* and using the connector number *Connector*.

Parameters

Type

The type of board to find:

CISYS_TYPE_R2 – Search for a RoadRunner/R3.
CISYS_TYPE_R64 – Search for a R64/R64e/Karbon/Neon/Alta.
CISYS_TYPE_GN2 - Find Aons, Axions, Cytons and Claxons.
CISYS_TYPE_ANY – Search for any type of board.

Switch

This is an integer value (usually between 0 and 3 inclusive) that matches the physical switch setting on the board to be found.

Connector

This is an integer value (usually between 1 and 4 inclusive) that matches the connector number on the board to be found. This is only useful for boards with more than one virtual frame grabber and more than one connector. For boards with only one connector, this should be set to 0.

pEntry

A pointer to an empty CiENTRY structure, used to tell the CiBrdOpen function which board to open.

Returns

CI_OK	The board was successfully found.
CISYS_ERROR_NOTFOUND	There is no board with this number.
CISYS_ERROR_UNKNOWN_TYPE	The board type is unknown.
CISYS_ERROR_SYSTEM	Error reading registry.

Comments

This function is usefull for find a particular board in a multi board system. There are two types of mult-board systems. You might have one physical board in your system, but it might have more than one virutal frame grabber on it. For example, the Karbon-4 looks to software like four frame grabbers. These are usually called virtual frame grabbers or VFGs. Each VFG on a Karbon 4 has its own connetor. So this function can be used to find the VFG associated with a particular connector number. For these situ-

ations, set the *Connector* variable to that of the desire VFG (camera) that you would like to open. In these cases, you will still need to set the *Switch* variable so that it matches that of the board, the default on all boards is 0.

For situations where more than one board is installed, you can set each board's phsycial switch to a different value. Then call this function and set the *Switch* parameter to match that of the board you want to open. In these situations, set the *Connector* parameter to 0.

In some situations, for example when two Karbon-4 boards are installed in the same system, you will have to set both the *Connector* and the *Switch* parameter to find the correct board/VFG to open.

If you only have one, single VFG, board installed in your system, it might be easier to use CiSysBrdFind function.

13.6 CiBrdOpen

Prototype BFRC CiBrdOpen(PCiENTRY *pEntry*, Bd **pBoard*, BFU32 *Mode*)

Description Opens a board for access. This function must return successfully before any other BitFlow SDK functions are called (with the exception of CiSysBrdFind and CiSysBrdEnum functions).

Parameters

pEntry

A pointer to a filled out CiENTRY structure. This structure describes which board is to be opened. The structure is filled out by a call to the CiSysBrdFind function.

**pBoard*

A pointer to a board handle. This handle is used for all further accesses to the newly opened board. This function takes a pointer to a handle where as all other functions just take a handle.

Mode

This parameter allows for different modes of opening the board, one or more of these parameters can be ORed together:

0 - board will open normally but not initialized. Board registers are not changed.
BFSysInitialize - initialize the board.
BFSysExclusive - open only if no other process has, and do not allow any subsequent process to open the board.
BFSysNoIntThread - do not start interrupt IRP thread.
BFSysNoCameraOpen - do not open any configured cameras.
BFSysNoOpenErrorMess - suppress all dialogs in open function
BFSysNoPoCLChange - This flag forces the system to leave the PoCL system as is (does not change its state).
BFSysPoCLUpOnly - This flag will cause the board to power up PoCL if it is off, but won't turn it off, if it is on.
BFSysPoCLCycle - This flag will cause the board to power down PoCL if it is on, the power PoCL back up.
BFSysSerialPortOpen - used when opening the serial port, included some of the above flags
BFSysNoCXPInit - Don't initialize the CXP subsystem
BFSysNoGenTLInit - Don't use GenTL camera control during board initialization.
BFSysNoIOReset - Do not reset I/O outputs before setting them as per configuration file

Returns

CI_OK	Function was successful.

CISYS_ERROR_OPENING	Error opening board.
CISYS_ERROR_BAD_ ENTRY	Invalid entry passed to function. Be sure that the entry returned from CiSysBrdFind is passed this function.

Comments

This function opens the board for all accesses. Call the CiSysBrdFind function to find the board you wish to open, then call this function to open to board. The board must be opened before any other functions can be called. When you are finished accessing the board you must call CiBrdClose, before exiting your process. Failure to call CiBrdClose will result in incorrect board open counts used by the driver.

If this function fails, you cannot access the board. Also, you do not need to call CiBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board.

You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call CiBrdClose first.

Calling this function with *Mode* = BFSysInitialize initializes the board and sets it up for the first camera that is configured for this board. If another process has already opened the board using this flag, the board will not be re-initialized, but you will have access to the board in the state that it is.

The *Mode* = BFSysExclusive is designed to guarantee that only one process can have the board open at a time. If the board has already been opened with this flag you will not be able to open it again, regardless of the *Mode* parameter that you use. If in CiSysExclusive mode, you will not be able to open the board if any other process has already opened the board, regardless of the mode the other process used to open the board. Finally, if you do succeed in opening the board in this mode, no other processes will be allowed to open the board.

13.7 CiBrdOpenCam

Prototype BFRC CiBrdOpen(PCiENTRY *pEntry*, Bd **pBoard*, BFU32 *Mode*, PBFCHAR *ForceCamFile*)

Description Opens a board for access. This function must return successfully before any other BitFlow SDK functions are called (with the exception of CiSysBrdFind and CiSysBrdEnum functions).

Parameters

pEntry

A pointer to a filled out CiENTRY structure. This structure describes which board is to be opened. The structure is filled out by a call to the CiSysBrdFind function.

pBoard

A pointer to a board handle. This handle is used for all further accesses to the newly opened board. This function takes a pointer to a handle where as all other functions just take a handle.

Mode

This parameter allows for different modes of opening the board, one or more of these parameters can be ORed together:

0 - board will open normally but not initialized. Board registers are not changed.
BFSysInitialize - initialize the board.
BFSysExclusive - open only if no other process has, and do not allow any subsequent process to open the board.
BFSysNoIntThread - do not start interrupt IRP thread.
BFSysNoCameraOpen - do not open any configured cameras.
BFSysNoAlreadyOpenMess – suppress board already open message.
BFSysNoPoCLChange - This flag forces the system to leave the PoCL system as is (don't change its state).
BFSysPoCLUpOnly - This flag will power up PoCL if it is off, but won't turn it off, if it is on.
BFSysSerialPortOpen - used when opening the serial port, included some of the above flags
BFSysNoCXPInit - Don't initialize the CXP subsystem
BFSysNoGenTLInit - Don't use GenTL camera control during board initialization.
BFSysNoIOReset - Do not reset I/O outputs before setting them as per configuration file

ForceCamFile

The camera file to open. The camera file should include the name and the file extension. If only the file name and extension are given, the camera configuration path is searched for the camera file. (The camera configuration path by default is the Config folder under the SDK root.) If the full path is given, the camera file will try and be opened from that location.

Note: For Gen2 boards, The format needs to be "mode@camfile.ext", where "mode" is one of the modes from the BFML file. If just "camfile.ext" is used, mode="default" will be opened.

Returns

CI_OK	Function was successful.
CISYS_ERROR_OPENING	Error opening board.
CISYS_ERROR_BAD_ ENTRY	Invalid entry passed to function. Be sure that the entry returned from CiSysBrdFind is passed this function.

Comments

This function opens the board for all accesses. Call the CiSysBrdFind function to find the board you wish to open, then call this function to open to board. The board must be opened before any other functions can be called. When you are finished accessing the board you must call CiBrdClose, before exiting your process. Failure to call CiBrdClose will result in incorrect board open counts used by the driver.

If this function fails, you cannot access the board. Also, you do not need to call CiBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board.

You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call CiBrdClose first.

Calling this function with *Mode* = BFSysInitialize initializes the board and sets it up for the first camera that is configured for this board. If another process has already opened the board using this flag, the board will not be re-initialized, but you will have access to the board in the state that it is.

The *Mode* = BFSysExclusive is designed to guarantee that only one process can have the board open at a time. If the board has already been opened with this flag you will not be able to open it again, regardless of the *Mode* parameter that you use. If in CiSysExclusive mode, you will not be able to open the board if any other process has already opened the board, regardless of the mode the other process used to open the board. Finally, if you do succeed in opening the board in this mode, no other processes will be allowed to open the board.

13.8 CiBrdCamSel

Prototype BFRC CiBrdCamSel(Bd *Board*, BFU32 *CamIndex*, BFU32 *Mode*)

Description Sets a board's current camera to the camera with the given index. Depending on the *mode*, the board can also be initialized for this camera.

Parameters

Board

Handle to board.

CamIndex

Index of camera to become current. Index is set in SysReg.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but do not modify the board.
CiSysConfigure - initialize the board for this camera.

Returns

CI_OK	Function was successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
R2_INCOMP	RoadRunner camera file is incompatible with this board, or camera file is incompatible with this version of the SDK.
R2_BAD_CNFG	An error occurred initializing the RoadRunner for this camera file.
BF_BAD_CAM_INDEX	An invalid CamIndex was passed to the function.

Comments Each board has associated with it a list of configured cameras (set in the SysReg application) and a current camera. By default, the current camera is the first camera in the list of configured cameras. The current camera is important because it dictates the parameters used for acquisition. There must be a current camera set in order to use the acquisition functions. This function allows you to pick one of the configured cameras to be the current camera.

If *Mode* = CiSysConfigure, the board will be initialized for the given camera.

This function is useful for switching on-the-fly between multiple preconfigured camera types.

13.9 CiBrdCamSetCur

Prototype BFRC CiBrdCamSetCur(Bd *Board*, PRVCAM *pCam*, BFU32 *Mode*)

Description Sets the current camera to the camera object *pCam* that is not necessarily one of the preconfigured cameras. The board can be optionally initialized to the camera.

Parameters

Board

Handle to board.

pCam

A camera object. If *pCam* = NULL, the function will set the current camera to the default camera as configured in SysReg.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but does not modify the board.
CiSysConfigure - initialize the board for this camera.

Returns

CI_OK	Function was successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
R2_INCOMP	RoadRunner camera file is incompatible with this board, or camera file is incompatible with this version of the SDK.
R2_BAD_CNFG	An error occurred initializing the RoadRunner for this camera file.

Comments This function sets the current camera to a camera object that is not one of the cameras already configured for the board (via SysReg). The camera must already be opened successfully (see CiCamOpen).

This function allows you to handle your own camera management. You can select, open, configure and close cameras to suit your applications needs independently of the SDK's camera management.

If *Mode* = CiSysConfigure, the board will be initialized for the given camera.

You must reset the current camera to another camera before you close your manually managed camera files. This simplest way to do this is call this function if the parameter *pCam* set to NULL.

13.10 CiBrdInquire

Prototype BFRC CiBrdInquire(Bd *Board*, BFU32 *Member*, PBFU32 *pVal*)

Description Returns parameters about the current board.

Parameters *Board*

Handle to board.

Member

Parameter to inquire about:

CiBrdInqModel - returns the board model. The parameter *pVal* will point to one of:

BFBrdValModel11 - Model RUN-PCI-11-xx
BFBrdValModel12- Model RUN-PCI-12-xx
BFBrdValModel13 - Model R3-PCI-CL-13-xx
BFBrdValModel14 - Model RUN-PCI-14-xx
BFBrdValModel23 - Model R3-PCI-CL-23-xx
BFBrdValModel24 - Model RUN-PCI-24-xx or R3-PCI-DIF
BFBrdValModel44 - Model RUN-PCI-44-xx
BFBrdValModel010 - Model RAV-PCI-010-xxx
BFBrdValModel110 - Model RAV-PCI-110-xxx
BFBrdValModel220 - Model RAV-PCI-220-xxx
BFBrdValModel440 - Model RAV-PCI-440-xxx
BFBrdValModelR64Cl - Model R64-PCI-CL-xx.
BFBrdValModelR64Dif - Model R64-PCI-DIF-xx.
BFBrdInqIDReg - returns the settings of the small switch on the top of the board.
BFBrdValUnkown - Model number is not known (this is a valid return, all of the possible models may be known at the time of the software release).
CiCamXXXX - inquiry parameter is sent the function CiCamInquire using the current camera configuration. See the function CiCamInquire for parameters. The result *pVal* is pass back through this function unmodified.

CiBrdInqModel - returns the board model. The parameter *pVal* will point to one of:

BFBrdValSpeed40MHz
BFBrdValSpeedNormal

CiBrdInqLUT - the type of LUT mounted on this board. The parameter *pVal* will be one of:

BFBrdValLUTNone
BFBrdValLUT16
BFBrdValLUT8And12

CiBrdInqScanType – returns board scan type. The parameter *pVal* will point to one of the following:

BFBrdValStandard – board will only work with standard scan cameras.
BFBrdValVariable – board will work with variable scan cameras and standard scan cameras.

CiBrdInqColorDecoder – indicates if board has NTSC/PAL decoder.The parameter *pVal* will point to one of the following:

BFBrdValDecoderMounted – color decoder mounted.
BFBrdValNoDecoder – no color decoder mounted.

CiBrdInqAnalogType – returns type of analog video input board is setup for.The parameter *pVal* will point to one of:

BFBrdValDifferential – board has differential video input.
BFBrdValSingle – board has single ended video input.
BFBrdInqNumCams – returns the number of cameras the board is configured for.

pVal

Pointer returned containing the requested value.

Returns

CI_OK	Function was successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R2_BAD_INQ_PARAM	The *Member* parameter is unknown.
RV_BAD_INQ_PARAM	The *Member* parameter is unknown.

Comments

This function is used to inquire of the system characteristics of the board. This function can also be called with CiCamInquire *Members*, which are then passed to that function using the board's current camera.

13.11 CiBrdClose

Prototype BFRC CiBrdClose(Bd *Board*)

Description Closes the board and frees all associated resources.

Parameters *Board*

Handle to board.

Returns

CI_OK	In all cases.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function closes the board and releases associated resources. This function must be called whenever a process exits regardless of the reason the process is exiting. The only time that this function does not have to be called is if CiSysBrdOpen fails. This function decrements the internal counters that are used to keep track of the number of processes that have opened the board.

13.12 CiBrdAqTimeoutSet

Prototype BFRC CiBrdAqTimeoutSet(Bd *Board*, BFU32 *Timeout*)

Description Sets the timeout value for this board's current camera.

Parameters

Board

Board to select the camera for.

Timeout

New value for timeout, in milliseconds. Set this value to the define INFINITE or 0xffffffff to have the functions never timeout.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
Non-zero	On error.

Comments This function sets the timeout value for this board's current camera. By default, this value is set from the camera configuration file.

13.13 CiBrdCamGetCur

Prototype BFRC CiBrdCamGetCur(Bd *Board*, PBFCNF **pCam*)

Description Gets the current acquire signal to a signal record provided by the caller.

Parameters

Board

Board to select.

**pCam*

Pointer to caller's signal record.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function gets the current acquire signal to a signal record provided by the caller. For more information about signals, refer to the Signal Functions section.

13.14 CiBrdType

Prototype BFRC CiBrdType (Bd *Board*)

Description Returns the type of board.

Parameters *Board*

Board to select.

Returns

CISYS_TYPE_R2	If the board is a RoadRunner or RoadRunnerCL.
CISYS_TYPE_R64	If the board is a R64.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function. The board is not recognized.

Comments Use this function to determine what board is being used.

13.15 CiBrdAqSigSetCur

Prototype BFRC CiBrdAqSigSetCur (Bd *Board*, PBFVOID *pAqSig,* BFU32 *AqEngine*)

Description Sets the current acquire signal to a signal record provided by the caller.

Parameters

Board

Board to select.

pAqSig

Pointer to caller's signal record.

AqEngine

AqEngJ – Acquisition engine J.
AqEngK – Acquisition engine K.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function. The board is not recognized.

Comments This function sets the current acquire signal to a signal record provided by the caller. For more information about signals, refer to the Signal Functions section.

The *AqEngine* parameter only used by the Raven. *AqEngine* is used by the Raven to chose between the two DMA engines on the board, engine J and engine K.

13.16 CiBrdAqSigGetCur

Prototype BFRC CiBrdAqSigGetCur (Bd *Board*, PBFVOID *pAqSig,* BFU32 *AqEngine*)

Description Gets the current acquire signal.

Parameters

Board

Board to select.

pAqSig

Pointer to storage for acquire signal.

AqEngine

AqEngJ – Acquisition engine J.
AqEngK – Acquisition engine K.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function. The board is not recognized.

Comments This function gets the current acquire signal. See the section on signal to understand what a signal is.

The *AqEngine* parameter only used by the Raven. *AqEngine* is used by the Raven to chose between the two DMA engines on the board, engine J and engine K.

13.17 CiBrdCamGetFileName

Prototype BFRC CiBrdCamGetFileName (Bd *Board*, BFU32 *Num*, PBFCHAR *CamName,* BFSIZET *CamNameStLen*)

Description Gets the file name of the attached camera(s).

Parameters

Board

Board to select.

Num

Camera number to get the name of.

CamName

Contains the file name of the camera configuration.

CamNameStLen

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *CamName*.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function. The board is not recognized.

Comments This function can be used to get the file name for one of the attached camera configurations. These configurations are attached to the board in SysReg. The *Num* parameter corresponds to the number configuration in the list of attached cameras in SysReg.

13.18 CiBrdCamGetFileNameWithPath

Prototype BFRC CiBrdCamGetFileName (Bd *Board*, BFU32 *Num*, PBFCHAR *CamNameWithPath,* BFSIZET *CamNameWithPathStLen*)

Description Gets the file name (including full path) of the attached camera(s).

Parameters

Board

Board to select.

Num

Camera number to get the name of.

CamNameWithPath

Contains the file name (including full path) of the camera configuration.

CamNamWithPateStLen

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *CamNameWithPath*.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function. The board is not recognized.

Comments This function can be used to get the file name for one of the attached camera configurations. The returned string includes the full path of the file. These configurations are attached to the board in SysReg. The *Num* parameter corresponds to the number configuration in the list of attached cameras in SysReg.

13.19 CiBrdCamGetMMM

Prototype BFRC CiBrdCamGetMMM(Bd *Board*, PBFCHAR *Make*, BFU32 *MakeStrSize*, PBFCHAR *Model*, BFU32 *ModelStrSize*, PBFCHAR *Mode*, BFU32 *ModeStrSize*)

Description Returns the make, model and mode of the current camera configuration.

Parameters

Board

Board to select.

Make

Returns the make string from the camera configuration file.

MakeStrSize

This size of the string pointed to by the *Make* parameter.

Model

Returns the model string from the camera configuration file.

ModelStrSize

This size of the string pointed to by the *Model* parameter.

Mode

Returns the mode string from the camera configuration file.

ModeStrSize

This size of the string pointed to by the *Mode* parameter.

Returns

CI_OK	If successful.
R64_BAD_CNF_FILE	The current camera configuration file is invalid.
R2_BAD_CNF_FILE	The current camera configuration file is invalid.
GN2_BAD_CNF_FILE	The current camera configuration file is invalid.

Comments This function can be used to get the file name for one of the attached camera configurations. The returned string includes the full path of the file. These configurations are attached to the board in SysReg. The *Num* parameter corresponds to the number configuration in the list of attached cameras in SysReg.

13.20 CiMMMIterate

Prototype

BFRC CiMMMIterate(BFU32 *Index*, PBFCHAR *Make*, BFU32 *MakeStrSize*, PBFCHAR *Model*, BFU32 *ModelStrSize*, PBFCHAR *Mode*, BFU32 *ModeStrSize,* PBFBOOL *pCLModel*)

Description

Iterates through all the configured boards in the system, returns the make, model and mode of the current camera configuration for each board.

Parameters

Index

Board index to select.

Make

Returns the make string from the camera configuration file.

MakeStrSize

This size of the string pointed to by the *Make* parameter.

Model

Returns the model string from the camera configuration file.

ModelStrSize

This size of the string pointed to by the *Model* parameter.

Mode

Returns the mode string from the camera configuration file.

ModeStrSize

This size of the string pointed to by the *Mode* parameter.

pCLModel

This parameter is set to TRUE if the board at the given index is a Camera Link board.

Returns

CI_OK	If successful.
CI_BAD_INDEX	There is no board with the index
CI_BAD_CONFIG	Error opening the board number *Index*

Comments

This function can be used to iterate through all the boards in a system and return each board's configuration's make, model and mode.

This function can be called in a loop, incrementing Index each time, in order to get the information on each board. Continue iterating until the function returns CI_BAD_INDEX.

Ci Camera Configuration

Chapter 14

14.1 Introduction

One of the most powerful features of BitFlow's interface boards, is the ability for the board to interface to an almost infinite variety of cameras. The knowledge behind these interfaces is stored in the camera configuration files.

The normal way a BitFlow application works is that the board is initialized to interface to the camera currently attached to the board. The currently attached camera is selected in the SysReg utility program. Normally an application is written so that it will work with whatever camera is attached. The board is initialized for the currently attached camera when CiBrdOpen is called. If an application is written this way there is no need to call any of the functions in this chapter. However, some users may want to manage what cameras are attached and how the user switches between them using their own software. For this reason, these camera configuration functions are provided.

The normal flow for an application that wants to manage its own camera files is as follows:

```
CiBrdOpen
CiCamOpen
CiBrCamSetCur
// processing and acquisition
CiCamClose
CiBrdClose
```

If using more than one camera:

```
CiBrdOpen
CiCamOpen            // open camera 0
CiCamOpen            // open camera 1
CiBrdCamSetCur       // configure for camera 0

// processing and acquisition
CiBrdCamSetCur       // configure for camera 1

// processing and acquisition
CiCamClose           // close camera 0
CiCamClose           // close camera 1
CiBrdClose
```

14.2 CiCamOpen

Prototype BFRC CiCamOpen(Bd *Board*, PCHAR *CamName*, PBFCNF **pCam*)

Description Allocates a camera configuration object, opens a camera configuration file, and loads the file into the object.

Parameters

Board

Handle to board.

CamName

The name of the camera file to open. Do not include the path. The camera file must be in the configuration directory (see the SysReg application). For example: "GenRS170-PLL.rvc".

**pCam*

A pointer to a camera object. The memory to hold the object is allocated in this function.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
R64_NO_CNFDIR_REG_KEY	The configuration directory entry is missing in the register (run SysReg).
R64_BAD_PATH	Error building the path to the camera file.
R64_BAD_STRUCT	Error calculating the size of the camera structure.
R64_BAD_ALLOC	Cannot allocate memory to perform open.
R64_BAD_CNF_FILE	Error opening or reading configuration file.
R64_BAD_HEADER	Error in configuration file header. This could include an error in one or more of the following items: signature (RoadRunner configuration), endian test (endian model is unknown), revision (camera revision is incompatible), size (size of file is not the same as written) and CRC (byte error in file).
R64_BAD_BINR	Error reading configuration item from file.
R64_BAD_CNFA	Error inserting configuration item into camera object.
ERRV_BAD_CNF_FILE	Error opening or reading configuration file.

Comments

This function allocates memory to hold a camera configuration object, locates the given camera configuration file in the configuration directory, checks the file for errors, then loads the camera configuration parameters into the camera object. The camera object is used to tell the system how to set up the board to acquire from a particular camera. Use the program CamVert to edit camera configuration files.

The resulting camera object can be passed to other functions such as CiBrdCamSetCur.

The resources allocated by the function must be freed by calling CiCamClose.

14.3 CiCamInquire

Prototype BFRC CiCamInquire(Bd *Board*, PBFCNF *pCam*, BFU32 *Member*, PBFU32 *pVal*)

Description Returns information about the given camera.

Parameters

Board

Handle to board.

pCam

Camera whose characteristics are requested.

Member

Characteristic to find the value of. The member must be one of:

CiCamInqXSize - width of image in pixels.
CiCamInqYSize0 - Camera 0 height of image in lines.
CiCamInqYSize1 - Camera 1 height of image in lines.
CiCamInqYSize2 - Camera 2 height of image in lines.
CiCamInqYSize3 - Camera 3 height of image in lines.
CiCamInqFormat - image format.
CiCamInqPixBitDepth - depth of pixel in bits, as acquired to host.
CiCamInqBytesPerPix - depth of pixel in bytes, as acquired to host.
CiCamInqFrameSize0 - camera 0 total size of image in bytes, as acquired to host.
CiCamInqFrameSize1 - camera 1 total size of image in bytes, as acquired to host.
CiCamInqFrameSize2 - camera 2 total size of image in bytes, as acquired to host.
CiCamInqFrameSize3 - camera 3 total size of image in bytes, as acquired to host.
CiCamInqFrameWidth - width of image in bytes, as acquired to host.
CiCamInqAqTimeout - number of milliseconds to wait before acquisition command times out.
CiCamInqBytesPerPixDisplay - depth of pixel in bytes, as acquired to display.
CiCamInqPixBitDepthDisplay - depth of pixel in bits, as acquired to display.
CiCamInqBitsPerSequence - depth of multi-channel pixel in bits, as acquired to host.
CiCamInqBitsPerSequenceDisplay - depth of multi-channel pixel in bits, as acquired to display.
CiCamInqDisplayFrameSize0 - total size of image in bytes, as acquired to display.
CiCamInqDisplayFrameWidth - width of image in bytes, as acquired to display.
CiCamInqCamType - camera type.

CiCamInqControlType - type of camera control accessible through API.

pVal

Pointer to value of the characteristic.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R64_BAD_INQ_PARAM	Unknown *Member* parameter.
RV_BAD_INQ_PARAM	Unknown *Member* parameter.
Non-zero	On error.

Comments

This function is used to inquire about characteristics of a camera. For 8-bit cameras, the parameter CiCamInqHostFrameSize is equal to CiCamInqDisplayFrameSize. The parameter only differs for pixel depths greater than eight.

Table 14-1 Shows the Ci parameters and equivalent for the RoadRunner parameters. If there is no equivalent parameter, an unknown parameter error will be returned.

Table 14-1 Correlation of Ci Parameters to RoadRunner Parameters

Ci Parameter	RoadRunner Parameter
CiCamInqXSize	R2CamInqXSize
CiCamInqYSize0	R2CamInqYSize
CiCamInqYSize1	Error
CiCamInqYSize2	Error
CiCamInqYSize3	Error
CiCamInqFormat	R2CamInqFormat
CiCamInqPixBitDepth	R2CamInqPixBitDepth
CiCamInqBytesPerPix	R2CamInqBytesPerPix
CiCamInqFrameSize0	R2CamInqHostFrameSize
CiCamInqFrameSize1	Error
CiCamInqFrameSize2	Error
CiCamInqFrameSize3	Error

Table 14-1 Correlation of Ci Parameters to RoadRunner Parameters

CiCamInqFrameWidth	R2CamInqHostFrameWidth
CiCamInqAqTimeout	R2CamInqAqTimeout
CiCamInqBytesPerPixDisplay	R2CamInqBytesPerPixDisplay
CiCamInqPixBitDepthDisplay	R2CamInqPixBitDepthDisplay
CiCamInqBitsPerSequence	R2CamInqBitsPerSequence
CiCamInqBitsPerSequenceDis-play	R2CamInqBitsPerSequenceDisplay
CiCamInqDisplayFrameSize0	R2CamInqDisplayFrameSize0
CiCamInqDisplayFrameWidth	R2CamInqDisplayFrameWidth
CiCamInqCamType	R2CamInqCamType
CiCamInqControlType	R2CamInqControlType

14.4 CiCamClose

Prototype BFRC CiCamClose(Bd *Board*, PBFCNF *pCam*)

Description Frees resources used by a camera object.

Parameters *Board*

Handle to board.

pCam

Camera object.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.

Comments This function frees all resources used by a camera object.

14.5 CiCamAqTimeoutSet

Prototype BFRC CiCamAqTimeoutSet(Bd *Board*, PBFCNF *pCam*, BFU32 *Timeout*)

Description Sets the acquisition timeout variable in the given camera configuration.

Parameters

Board

Handle to board.

pCam

Camera whose characteristics are requested.

Timeout

New value for timeout, in milliseconds.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
Non-zero	On error.

Comments This functions sets the timeout value for the configuration. The timeout value is how long the system will wait for an acquisition command to complete before returning an error.

14.6 CiCamModeSet

Prototype BFRC CiCamModeSet(Bd *Board*, PBFCNF *pCam*, PBFCHAR *ModeName*)

Description Sets the board and the camera in the given mode. Only works for models that support multi-mode camera configuration files.

Parameters

Board

Handle to board.

pCam

Pointer to the current camera configuration.

ModeName

Mode to set the board and camera to.

Returns

CI_OK	If successful.
GN2_BAD_CNF_FILE	The camera configuration is invalid

Comments This functions sets both the board and the camera into the given mode. This function only works on models that support multi-mode camera configuration files. Multi-mode camera files can program both the frame grabber's settings (e.g. I/O, ROI, Timing Sequencer, etc) and registers as well as programming the camera's registers (for cameras that support standard register access.

If the mode *ModeName* does not exist, no action is taken and the function will still return BF_OK.

The mode should not be changed when the board is set up for acquisition. The mode should only be change when acquisition has been cleaned up (or not yet set up).

For a complete list of features that can be changed by changing modes, see the documentation for the camera configuration files.

14.7 CiCamModeGet

Prototype BFRC CiCamModeSet(Bd *Board*, PBFCHAR *ModeName*, BFSIZET *ModeNameSize*)

Description Gets the name of the current mode. Only works for models that support multi-mode camera configuration files.

Parameters

Board

Handle to board.

ModeName

String that will contain the name of the current mode.

ModeNameSize

Size of the buffer pointed to by *ModeName*.

Returns

CI_OK	In all cases

Comments This functions returns the name (as a string) of the current mode the board and camera are configured for. This function only works for models that support mulit-mode camera configurations.

The current mode can be set via SysReg or by calling CiCamModeSet

14.8 CiCamModesEnum

Prototype

BFRC CiCamModesEnum(Bd Board, PBFCNF pCam, BFU32 Index, PBFCHAR ModeName, BFU32 ModeNameSize, PBFCHAR ModeComment, BFU32 ModeCommentSize)

Description

Enumerates all of the modes supported by the current camera configuration file. Only works for models that support multi-mode camera configuration files.

Parameters

Board

Handle to board.

pCam

Pointer to the current camera configuration.

Index

The index used to retrieve the mode information.

ModeName

String that will contain the name of the mode for the given index. If there is no mode corresponding to the given *Index*, the string is returned empty, *ModeName*[0] = 0.

ModeNameSize

Size of the buffer pointed to by *ModeName*.

ModeComment

String that will contain the mode comment for the given index.

ModeCommentSize

Size of the buffer pointed to by *ModeComment*.

Returns

CI_OK	In all cases
GN2_BAD_CND_FILE	The camera configuration file is invalid

Comments

This function is used to enumerate all of the modes in the current camera configuration. The function only works for multi-mode camera configurations. This function can be used, for example, to provide a list of modes that the user can choose.

To enumerate all of the modes, call this function in a loop and increment Index each time the function is called. Each time the function returns a new ModeName and ModeComment will be returned. Once all of the modes have been enumerate, the parameter ModeName will be return such that ModeName[0] = 0.

Once a choice has been made, through, for example, a user dialog, the chosen *ModeName* can the be passed to CiCamModeSet in order to set both the frame grabber and the camera in the chosen mode.

14.9 CiCamUpdateParams

Prototype BFRC CiCamUpdateParams(Bd *Board*)

Description Update the drivers internal version of the connected CoaXPress camera's ROI parameters.

Parameters *Board*

Handle to board.

pCam

Camera object.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function is used to get the current camera's acquisition parameters. Currently this only works on CXP camera that are CXP revision 1.1 or later. The update takes place internally. The caller should then use CiBrdInquire() to get the new values. Note this auto configure functionality only works for parameter in the BFML file whose values are set to "Default".

Note: This this function only works on Gen 2 boards.

Ci Signal Functions

Chapter 15

15.1 Introduction

The purpose of the Signal Functions is to make hardware interrupts available to user-level applications in a simple and efficient manner. In fact, under Windows, there is no way for a user-level application to get direct notification of a hardware interrupt. Only kernel-level drivers can contain interrupt service routines (ISR). Most customers do not want to deal with the complications of writing ISRs anyway, so BitFlow has come up with the signaling system.

Basically, a signal is a wrapper around a Windows semaphore object. The signal has a state and a queue. Every time an interrupt occurs, the signal's state changes. The nice thing about signals is that you can wait for their state to change, without using any CPU cycles. This is what makes them so efficient. This means that you can have one thread processing images while another is waiting for the next image to be completely DMAed. The thread that is waiting for the signal consumes very little CPU time, thus making most of the CPU available for processing.

Start by creating a signal with the CiSignalCreate. There are a number of different interrupts that the signal can wait for, and it is in this function that you specify the one you want. Once the signal is created, your application waits for the interrupt with either the CiSignalWait or the CiSignalWaitNext function. The difference being that the CiSignalWait function uses a signals queue. If an interrupt has occurred before this function is called, then this function will return immediately. It will continue to return immediately until there are no more interrupts in the queue. The CiSignalWaitNext function always waits for the next interrupt after being called, regardless of how many have occurred since it was last called.

Signals can be used in a single threaded application, but whenever one of the wait functions is called, execution will be blocked until the interrupt occurs. Because this situation can potentially hang a process, a time-out parameter is provided for all of the wait functions. If you need an application to process data while waiting on an image to be captured, create a separate thread to call the wait function. Meanwhile, another thread can be processing with most of the CPUs cycles. A thread waiting on a signal can be cancelled with the function CiThreadCancel. This causes the waiting thread to return from the wait function with an error code indicating that it has been cancelled.

The following is an example of how these functions can be called:

```
Int ImageIn = 0
main ()
{
   CiBrdOpen// open board
   CiSignalCreate// create the signal for EOF
   CreateThread(EOFThread)// create a thread
   while (KeepProcessing)// main processing loop
```

```
   {
         // here we loop until we have an image
         while (ImageIn !=1)
         {
               // secondary processing
         }

         // now we have an image so process it

         ImageIn = 0           // reset variable

         // primary image processing
   }
   // Clean up
   CiSignalCancel              // cancel signal kill thread
   CiSignalFree                // free signal resources
   CiBrdClose                  // close board
}

// thread to watch for end of frame
EOFThread()
{
   loop
   {
         rv = CiSignalWait       // wait for signal
         if(rv == CANCELED)      // was is cancel?
               exit loop         // yes, kill this thread else
         else
               ImageIn = 1       // no, set new image flag
   }
}
```

15.2 CiSignalCreate

Prototype BFRC CiSignalCreate(Bd *Board*, BFU32 *Type*, PCiSIGNAL *pSignal*)

Description Creates a signal that will allow user level thread to be notified of hardware interrupts.

Parameters

Board

Handle to board.

Type

Type of interrupt signal to create. See Table 15-1 below a complete list options for this parameter.

pSignal

Pointer to CiSIGNAL structure.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
BF_BAD_ARGS	Unknown interrupt type.
BF_BAD_ALLOC	Could not allocate memory for signal.
BF_BAD_SEMAPHORE	Could not get semaphore object from operating system.

Comments This function creates a signal object that is used to receive interrupt notifications from the board. The CiSignalWaitXXXX function takes a signal as a parameter. These functions efficiently wait for an interrupt of the given type to occur. The best way to use a signal is to create a separate thread that calls one of the CiSignalWaitXXXX functions. This thread will consume minimal CPU cycles until the interrupt occurs. When the interrupt occurs, the signal is notified and the CiSignalWaitXXXX functions will return. The thread can then take appropriate action, calling whatever functions are necessary and/or send messages to the main application thread.

This signaling system is the only way to handle board interrupts at the user application level.

More than one signal can be created for the same interrupt on the same board. Also, more than one process and/or thread can wait for the same interrupt. When the interrupt occurs, all of the signals will be notified in the order they were created. The signal created by this function receives interrupt notification only from the board passed to this function in the *Board* parameter.

The most frequently used signal is *Type* = CiIntTypeEOD. The CiAqSetup function automatically sets the interrupt bit in the last quad in the QTab of the current image. This signal will be notified when the last pixel of the image has been DMAed into memory, and the current acquisition is done in the case of a snap or freeze. The net result is the application will receive this interrupt at the end of every frame.

The signal created by this function must be cleaned up by calling CiSignalFree.

Note: The signal Type=CiIntTypeEOD was refered to in previous releases as CiIntTypeQuadDone.

Note:

Interrupts

Table 15-1 shows all of the options for the parameter *Type*. Not all interrupt types are available for all models.

Note: In the table below, all signal types start with "BF", some of these types have an equivalent "Ci" type, this is shown in Table 15-2.

*Note: Interrupt Types marked with an * are not used by any current boards supported by the current SDK, however, they are still valid for code that might be built with older SDK versions.*

Table 15-1 Interrupt Types

Type	R2/R3	Karbon, Neon, Alta	Aon, Cyton	Axion
BFIntTypeHW	X	X		
BFIntTypeFIFO	X	X		
BFIntTypeDMADone	X			
BFIntTypeEOD	X	X	X	X
BFIntTypeCTab	X	X		
BFIntTypeCCUJ *				
BFIntTypeCCUK *				
BFIntTypeDMAOnly	X			
BFIntTypeEOFDMAJ *				
BFIntTypeEOFDMAK *				
BFIntTypeTrig	X	X	X	X
BFIntTypeSerial	X	X		X
BFIntTypeQL *				
BFIntTypeEOF	X	X		
BFIntTypeCXP			X	
BFIntTypeEncB			X	X

Table 15-1 Interrupt Types

Type	R2/R3	Karbon, Neon, Alta	Aon, Cyton	Axion
BFIntTypeEncA			X	X
BFIntTypeBMError			X	X
BFIntTypeAELossOfSync			X	X
BFIntTypePCIEPkt-Dropped			X	X
BFIntTypeYAq			X	X
BFIntTypeZAq			X	X
BFIntTypeVStart			X	X
BFIntTypeYStart			X	X
BFIntTypeZStart			X	X
BF0UnderCurrent			X	
BF0OverCurrent			X	
BF0TrigAckRcvd			X	
BF0GpioAckRcvd			X	
BF0CtlAckRcvd			X	
BF0GpioRcvd			X	
BF0TrigRcvd			X	
BF0CtlRspFifoOvf			X	
BF0CtlReqFifoOvf			X	
BF0GpioNomatch			X	
BF0TrigNomatch			X	
BF0IoackUnknownType			X	
BF0IoackNomatch			X	
BF0IoackUnexpectedInt			X	
BF0IoackNomatch2			X	
BF0StrmPktDrop			X	
BF0StrmNotEnoughDat			X	
BF0StrmTooMuchDat			X	
BF0StrmBadCrc			X	
BF0StrmOverflow			X	

Table 15-1 Interrupt Types

Type	R2/R3	Karbon, Neon, Alta	Aon, Cyton	Axion
BF0StrmCorner			X	
BF0SerdesLostAlign			X	
BF1UnderCurrent			X	
BF1OverCurrent			X	
BF1TrigAckRcvd			X	
BF1GpioAckRcvd			X	
BF1CtlAckRcvd			X	
BF1GpioRcvd			X	
BF1TrigRcvd			X	
BF1CtlRspFifoOvf			X	
BF1CtlReqFifoOvf			X	
BF1GpioNomatch			X	
BF1TrigNomatch			X	
BF1IoackUnknownType			X	
BF1IoackNomatch			X	
BF1IoackUnexpectedInt			X	
BF1IoackNomatch2			X	
BF1StrmPktDrop			X	
BF1StrmNotEnoughDat			X	
BF1StrmTooMuchDat			X	
BF1StrmBadCrc			X	
BF1StrmOverflow			X	
BF1StrmCorner			X	
BF1SerdesLostAlign			X	
BF2UnderCurrent			X	
BF2OverCurrent			X	
BF2TrigAckRcvd			X	
BF2GpioAckRcvd			X	
BF2CtlAckRcvd			X	

Table 15-1 Interrupt Types

Type	R2/R3	Karbon, Neon, Alta	Aon, Cyton	Axion
BF2GpioRcvd			X	
BF2TrigRcvd			X	
BF2CtlRspFifoOvf			X	
BF2CtlReqFifoOvf			X	
BF2GpioNomatch			X	
BF2TrigNomatch			X	
BF2IoackUnknownType			X	
BF2IoackNomatch			X	
BF2IoackUnexpectedInt			X	
BF2IoackNomatch2			X	
BF2StrmPktDrop			X	
BF2StrmNotEnoughDat			X	
BF2StrmTooMuchDat			X	
BF2StrmBadCrc			X	
BF2StrmOverflow			X	
BF2StrmCorner			X	
BF2SerdesLostAlign			X	
BF3UnderCurrent			X	
BF3OverCurrent			X	
BF3TrigAckRcvd			X	
BF3GpioAckRcvd			X	
BF3CtlAckRcvd			X	
BF3GpioRcvd			X	
BF3TrigRcvd			X	
BF3CtlRspFifoOvf			X	
BF3CtlReqFifoOvf			X	
BF3GpioNomatch			X	
BF3TrigNomatch			X	
BF3IoackUnknownType			X	

Table 15-1 Interrupt Types

Type	R2/R3	Karbon, Neon, Alta	Aon, Cyton	Axion
BF3IoackNomatch			X	
BF3IoackUnexpectedInt			X	
BF3IoackNomatch2			X	
BF3StrmPktDrop			X	
BF3StrmNotEnoughDat			X	
BF3StrmTooMuchDat			X	
BF3StrmBadCrc			X	
BF3StrmOverflow			X	
BF3StrmCorner			X	
BF3SerdesLostAlign			X	

The following table shows the "Ci" and "Bf" name equivalents. Either name can be used passed in to this function for the *Type* parameter.

Table 15-2 Ci Equivalent Types

"Ci" Type	"BF" Type
CiIntTypeHW	BFIntTypeHW
CiIntTypeFIFO	BFIntTypeFIFO
CiIntTypeDMADone	BFIntTypeDMADone
CiIntTypeEOD	BFIntTypeEOD
CiIntTypeCTab	BFIntTypeCTab
CiIntTypeDMAOnly	BFIntTypeDMAOnly
CiIntTypeCCUJ	BFIntTypeCCUJ
CiIntTypeCCUK	BFIntTypeCCUK
CiIntTypeEOFDMAJ	BFIntTypeEOFDMAJ
CiIntTypeEOFDMAK	BFIntTypeEOFDMAK
CiIntTypeTrig	BFIntTypeTrig
CiIntTypeSerial	BFIntTypeSerial

15.3 CiSignalWait

Prototype BFRC CiSignalWait(Bd *Board*, PCiSIGNAL *pSignal*, BFU32 *TimeOut*, PBFU32 *pNumInts*)

Description Efficiently waits for an interrupt to occur. Returns immediately if one has occurred since the function was last called.

Parameters

Board

Handle to board.

pSignal

Pointer to CiSIGNAL previously created by CiSignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout.

pNumInts

Pointer to a BFU32. When the function returns, it will contain the number of interrupts (the interrupt queue) that have occurred since this function was last called.

Returns

CI_OK	Interrupt has occurred.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
BF_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
BF_SIGNAL_CANCEL	Signal was canceled by another thread (see CiSignalCancel).
BF_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
BF_WAIT_FAILED	Operating system killed the signal.

Comments This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function will return immediately if the interrupt has occurred since the function was last called with this signal. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

When this function returns, the *pNumInts* parameter will contain the number of interrupts that have occurred since this function was last called. This is essentially an interrupt queue. Normally this will be one. However, if one or more interrupts have occurred, the function will return immediately and this variable will indicate the num-

ber that has occurred. This parameter is useful in determining if frames were missed. This function will continue to return immediately, reducing the number of interrupts in the queue each time until every interrupt that has occurred has been acknowledged, and the queue is empty.

To wait for the next interrupt and ignore any previous interrupts, use CiSignalWait-Next.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

15.4 CiSignalWaitEx

Prototype

BFRC CiSignalWaitEx(Bd *Board*, PCiSIGNAL *pSignal*, BFU32 *TimeOut*, PBFU32 *pNumInts*, BFSignalTimeInfoPtr *pTimeInfo*)

Description

Efficiently waits for an interrupt to occur. Returns immediately if one has occurred since the function was last called. Returns a high accuracy time stamp.

Parameters

Board

Handle to board.

pSignal

Pointer to CiSIGNAL previously created by CiSignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout.

pNumInts

Pointer to a BFU32. When the function returns, it will contain the number of interrupts (the interrupt queue) that have occurred since this function was last called.

pTimeInfo

A pointer to a BFSignalTimeInfoRec structure. This structure will be filled out with time stamp information from when the interrupt occurred.

Returns

CI_OK	Interrupt has occurred.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
BF_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
BF_SIGNAL_CANCEL	Signal was canceled by another thread (see CiSignalCancel).
BF_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
BF_WAIT_FAILED	Operating system killed the signal.

Comments

This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function will return immediately if the interrupt has occurred since the function was last called with this signal. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

When this function returns, the *pNumInts* parameter will contain the number of interrupts that have occurred since this function was last called. This is essentially an interrupt queue. Normally this will be one. However, if one or more interrupts have occurred, the function will return immediately and this variable will indicate the number that has occurred. his parameter is useful in determining if frames were missed. This function will continue to return immediately, reducing the number of interrupts in the queue each time until every interrupt that has occurred has been acknowledged, and the queue is empty.

To wait for the next interrupt and ignore any previous interrupts, use CiSignalWaitNext.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

The pTimeInfo points to a BFSignalTimeInfoRec structure that is filled out when this function returns. The time stamp information is extremely accurate as it is captured in the kernel level ISR when the interrupt actually occurs. The pTimeInfo->TimeStamp member is the actual time stamp, this is uses the CPU clock. This value is the raw number of CPU clocks that have occurred since the system booted. You can use the function BFFine to the current time stamp at the start of a process, the subtract the signals TimeStamp to the delta between the start and when an interrupt occurred. You can convert to seconds use the value returned from BFFineRate.

See the SDK example “inttime.c” for an illustration of these functions.

15.5 CiSignalNextWait

Prototype BFRC CiSignalNextWait(Bd *Board*, PCiSIGNAL *pSignal*, BFU32 *TimeOut*)

Description Like CiSignalWait, this function waits efficiently for an interrupt. However, this version always ignores any interrupts that might have occurred since it was called last, and just waits for the next interrupt.

Parameters

Board

Handle to board.

pSignal

Pointer to CiSIGNAL previously created by CiSignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout

Returns

CI_OK	Interrupt has occurred.
BF_BAD_SIGNAL	An invalid board handle was passed to the function.
BF_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
BF_SIGNAL_CANCEL	Signal was canceled by another thread (see CiSignalCancel).
BF_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
BF_WAIT_FAILED	Operating system killed the signal.

Comments This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function waits for the next interrupt, regardless of the number of interrupts in the signal's queue. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

Use CiSignalWait if you need a function that will return immediately if an interrupt has already occurred.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

15.6 CiSignalCancel

Prototype BFRC CiSignalCancel(Bd *Board*, PCiSIGNAL *pSignal*)

Description Cancels a signal, any CiSignalWaitXXX function will return with a value of BF_SIGNAL_CANCEL.

Parameters

Board

Handle to board.

pSignal

Pointer to CiSIGNAL to cancel.

Returns

CI_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.

Comments This function will cancel a signal. It is primarily used by multi threaded applications where one thread is waiting (with one of the CiSignalWaitXXXX functions) for a signal. Another thread can cancel the signal with this function, thereby waking up the waiting thread. When the waiting thread wakes up by CiSignalWaitXXXX function returning, the return value can be examined. If the return value is BF_SIGNAL_CANCEL, the thread knows that the signal it was waiting for was canceled, and it can take appropriate action.

This function is usually used as a clean way for the main application thread to tell waiting threads to kill themselves.

Canceling a signal with this function will interfere with its internal interrupt counts. Therefore, this function should only be called when synchronization with the interrupt is no longer important and/or the signal is going to be destroyed.

15.7 CiSignalQueueSize

Prototype BFRC CiSignalQueueSize(Bd *Board*, PCiSIGNAL *pSignal*, PBFU32 *pNumInts*)

Description Reports the current number of interrupts in a signal's queue.

Parameters

Board

Handle to board.

pSignal

Pointer to CISIGNAL whose queue is to be investigated.

pNumInts

When the function returns *pNumInts*, it will contain the number of interrupts in the signal's queue.

Returns

CI_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.

Comments This function returns the number of interrupts in a signal's queue. This function is useful for testing to see if any interrupts have come in for a given signal, when you do not want to call one of the CiSignalWaitXXX functions. This function can be called any time.

15.8 CiSignalQueueClear

Prototype BFRC CiSignalQueueClear(Bd *Board*, PCiSIGNAL *pSignal*)

Description Clears interrupts from a single queue.

Parameters

Board

Handle to board.

pSignal

Pointer to CiSIGNAL whose queue is to be investigated.

Returns

CI_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.
BF_WAIT_FAILED	Error clearing queue.

Comments This function clears all of the interrupts for a given signal's queue. This allows a thread to wait for the next interrupt to occur. This function is usually only used to resynchronize a signal to the current state of acquisition (i.e., ignore any interrupts that have occurred in the past) before calling CiSignalWait. To always wait for the next interrupt, call CiSignalWaitNext.

15.9 CiSignalFree

Prototype BFRC CiSignalFree(Bd *Board*, PCiSIGNAL *pSignal*)

Description Frees all resources used by a signal.

Parameters

Board

Handle to board.

pSignal

Pointer to CiSIGNAL whose queue is to be investigated.

Returns

CI_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.

Comments This function frees the resources used by a signal and removes it from the list of signals that get interrupt notification.

15.10 CiCallBackAdd

Prototype BFCAPI CiCallBackAdd(Bd *Board*, BFU32 *SignalType*, BFCallBackFuncPtr *CallBackFunc*, PBFVOID *pUserData*)

Description Adds a call back function to the list of call back functions.

Parameters

Board

Board ID.

SignalType

Type of interrupt signal used to initiate calling of the call back function. The list of signals is the same as used in the CiSignalCreate() function. These are also listed in the header file "CiDef.h".

Note: You can OR the SignalType paramter with the flag BFCBModeGrabOnly, which will force the system to only call your call back function when the board is actively grabbing.

CallBackFunc

This is a pointer to the call back function. The function must have the following format:

```
void CallBackFunc(Bd Board, BFU32 Num, PBFVOID pUserData);
```

This function will be called whenever the interrupt of type *SignalType* occurs. The *Num* parameter is the size of the signal queue (i.e. the number of un-handled interrupts left after this one is handled).

pUserData

A pointer to user allocated structure which can contain any context data that might be needed in the Call Back function when it is actually called. Can be NULL.

Returns

BF_OK	Success
BF_NULL_POINTER	The parameter *CallBackFunc* is NULL
BF_BAD_SIGNAL	The parameter *SignalType* is not a valid signal type.
BF_CB_ALREAD_SET	There is already a call back function for this signal type.
THREAD_FAIL	The thread required to run the call back system could not be created

Comments

A call back function is a way for a user's code to be notified of an event (usually hardware) has occurred on the board. For example, the call back function can be called every time a new frame has been acquired.

Call back functions can be used instead of the signalling functions (See CiSignalCreate() and associated functions. The advantage of call back functions is that a separate thread does not need to be created (like the Signal functions). The disadvantage is that call back functions will execute in a thread whose relative priority is determined by the call back system, not the user. This means the that user has no control over the priority of the processing that happens in the call back function. In general, call back functions are best used in simple applications where thread priority is not critical.

The call back function will be called whenever the interrupt of type *SignalType* occurs. The *Num* parameter is the size of the signal queue (i.e. the number of un-handled interrupts left after this one is handled). This behavior is similar to the Signalling System, the call back function will be called repeatedly until there are no more interrupts in the queue.

When a call back function is added using this function, the interrupt associated with the *SignalType* parameter is automatically enabled on the board.

When the user no longer wants the call back function to be called, or is finished with the resource, the function CiCallBackRemove() should be called with the same *SignalType* as was use to add it.

If you do not wish your call back function to be called when the board is not grabbing, OR the *SignalType* parameter with the flag BFCBModeGrabOnly (e.g. BFIntTypeEOD | BFCBModeGrabOnly). When this flag is used, your callback function will only be called when the board is actively grabbing.

The pointer *pUserData* is designed so that the user can get context information inside of the call back function (when it is called). This pointer can point to anything (cast it inside the call back function). It must be allocated and de-allocated by the user. It can also be NULL.

15.11 CiCallBackRemove

Prototype BFCAPI CiCallBackRemove(Bd *Board*, BFU32 *SignalType*)

Description Removes a call back function from the list of call back functions.

Parameters *Board*

Board ID.

SignalType

Type of interrupt signal used to initiate calling of the call back function. The list of signals is the same as used in the CiSignalCreate() function. These are also listed in the header file "CiDef.h".

Returns

BF_OK	Success

Comments This function removes a call back function from the list of call back functions. Once a call back function is removed, it will never be called again. A call back function can be added again using CiCallBackAdd().

For more information on call back function see CiCallBackAdd().

15.12 CiSignalNameGet

Prototype BFRC CiSignalNameGet(Bd *Board*, BFU32 *Type*, PBFCHAR *SignalName*, BFU32 *SignalNameSize*)

Description Gets the name of a signal given its integer type.

Parameters

Board

Handle to board.

Type

The signal to retrieve the name for.

SignalName

A pointer to a string, when the function returns it will contain the name of the give signal.

SignalNameSize

The size of the buffer pointed to by *SignalName*.

Returns

CI_OK	Success.
BF_BAD_ARGS	The parameter *Type* is not a valid signal type.

Comments This function can be use to fetch the actual human readable name of a signal type. See the function CiSignalCreate for all possible values.

Ci LUTs

Chapter 16

16.1 Introduction

These functions allow an application full control over the Look Up Tables (LUTs) on the Board. The LutPeek and LutPoke functions are fairly inefficient and should only be used in the case of modifying a small number of entries. For accessing a larger number of entries or the entire LUT, create an array on the host and use the CiLutWrite and CiLutRead functions. To create a "ramp" function in the LUTs, use the CiLutRamp function.

Note: Not all BitFlow board's have LUTs.

16.2 CiLutPeek

Prototype BFU32 CiLutPeek(Bd *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*)

Description Reads a single LUT value.

Parameters

Board

Bd Board ID.

Mode

LUT Mode:

CiLut8Bit - peek an 8-bit value out of an 8-bit LUT.
CiLut12Bit - peek an 16-bit value out of an 12-bit LUT.
CiLut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

CiLutBank0 - peek LUT bank 0
CiLutBank1 - peek LUT bank 1
CiLutBank2 - peek LUT bank 2
CiLutBank3 - peek LUT bank 3

Lane

One or more LUT lanes ORed together:

CiLutLane0 - peek LUT lane 0
CiLutLane1- peek LUT lane 1
CiLutLane2 - peek LUT lane 2
CiLutLane3 - peek LUT lane 3

Addr

LUT address.

Returns

The LUT value.	If successful.
CISYS_ERROR_NOTSUP-PORTED	This function not support for this board type.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.

Comments LUT definitions are declared in CiDef.h.

16.3 CiLutPoke

Prototype BFRC CiLutPoke(Bd *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *Value*)

Description Writes a single LUT value to one or more LUT lanes.

Parameters

Board

Bd Board ID.

Mode

LUT Mode:

CiLut8Bit - peek an 8-bit value out of an 8-bit LUT.
CiLut12Bit - peek an 16-bit value out of an 12-bit LUT.
CiLut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

CiLutBank0 - poke LUT bank 0
CiLutBank1 - poke LUT bank 1
CiLutBank2 - poke LUT bank 2
CiLutBank3 - poke LUT bank 3

Lane

One or more LUT lanes ORed together:

CiLutLane0 - poke LUT lane 0
CiLutLane1- poke LUT lane 1
CiLutLane2 - poke LUT lane 2
CiLutLane3 - poke LUT lane 3

Addr

LUT address.

Value

LUT write value.

Returns

CI_OK	Function successful.

CISYS_ERROR_NOTSUP-PORTED	This function not support for this board type.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_NO_BIG_LUTS	RoadRunner board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_POKE_ERR	LUT poke failed.
RV_BAD_BANK	Illegal LUT bank.
RV_BAD_LUT_ADDR	Illegal LUT address.
RV_LUT_POKE_ERR	LUT poke failed.

Comments

LUT definitions are declared in CiDef.h.

16.4 CiLutRead

Prototype BFRC CiLutRead(Bd *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *NumEntries*, PBFVOID *pDest*)

Description Reads a LUT.

Parameters

Board

Bd Board ID.

Mode

LUT Mode:

CiLut8Bit - peek an 8-bit value out of an 8-bit LUT.
CiLut12Bit - peek an 16-bit value out of an 12-bit LUT.
CiLut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

CiLutBank0 - read LUT bank 0
CiLutBank1 - read LUT bank 1
CiLutBank2 - read LUT bank 2
CiLutBank3 - read LUT bank 3

Lane

One or more LUT lanes ORed together:

CiLutLane0 - read LUT lane 0
CiLutLane1 - read LUT lane 1
CiLutLane2 - read LUT lane 2
CiLutLane3 - read LUT lane 3

Addr

LUT address.

NumEntries

Number of LUT entries to read.

pDest

Storage for LUT entries. The size of the destination is based on the LUT *mode* being used and the *NunEntries*. If CiLut8Bit LUT mode is being used, memory should be allocated for *NumEntries* of the BFU8 data type (a byte). Both CiLut12Bit and CiLut16Bit modes should use *NumEntries* of the BFU16 data type (a word). A example of the usage would be:

```
BFU8 LUT8[256];  // CiLut8Bit LUT mode.
BFU16 LUT16[4096]; // CiLut12Bit and CiLut16Bit LUT modes
```

Returns

CI_OK	Function successful.
CISYS_ERROR_BAD_ BOARDPTR	This function not support for this board type.
CISYS_ERROR_NOTSUP-PORTED	The R64 dose not support this function.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_NO_BIG_LUTS	RoadRunner board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_TOO_MANY_LANES	Only one lane may be read at a time.
R2_LUT_READ_ERR	LUT read failed.
RV_BAD_BANK	Illegal LUT bank
RV_BAD_LUT_ADDR	Illegal LUT address.
RV_TOO_MANY_LANES	Only one lane may be read at a time.
RV_LUT_READ_ERR	LUT read failed.

Comments

LUT definitions are declared in CiDef.h.

16.5 CiLutWrite

Prototype BFRC CiLutWrite(Bd *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *NumEntries*, PBFVOID *pSource*)

Description Writes a LUT.

Parameters

Board

Bd Board ID.

Mode

LUT Mode:

CiLut8Bit - peek an 8-bit value out of an 8-bit LUT.
CiLut12Bit - peek an 16-bit value out of an 12-bit LUT.
CiLut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

CiLutBank0 - write LUT bank 0
CiLutBank1 - write LUT bank 1
CiLutBank2 - write LUT bank 2
CiLutBank3 - write LUT bank 3

Lane

One or more LUT lanes ORed together:

CiLutLane0 - write LUT lane 0
CiLutLane1- write LUT lane 1
CiLutLane2 - write LUT lane 2
CiLutLane3 - write LUT lane 3

Addr

LUT address.

NumEntries

Number of LUT entries to write.

pSource

Storage LUT data. The size of the source is based on the LUT *mode* being used and the *NunEntries*. If CiLut8Bit LUT mode is being used, memory should be allocated for *NumEntries* of the BFU8 data type (a byte). Both CiLut12Bit and CiLut16Bit modes should use *NumEntries* of the BFU16 data type (a word). A example of the usage would be:

```
BFU8 LUT8[256];  // CiLut8Bit LUT mode.
BFU16 LUT16[4096]; // CiLut12Bit and CiLut16Bit LUT modes.
```

Returns

CI_OK	Function successful.
CISYS_ERROR_NOTSUP-PORTED	This function not support for this board type.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_NO_BIG_LUTS	RoadRunner board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_WRITE_ERR	LUT write failed.
RV_BAD_BANK	Illegal LUT bank.
RV_BAD_LUT_ADDR	Illegal LUT address.
RV_LUT_WRITE_ERR	LUT write failed.

Comments

LUT definitions are declared in CiDef.h.

16.6 CiLutFill

Prototype BFRC CiLutFill(Bd *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *NumEntries*, BFU32 *Val*)

Description Fills a LUT with a constant.

Parameters

Board

Bd Board ID.

Mode

LUT Mode:

CiLut8Bit - peek an 8-bit value out of an 8-bit LUT.
CiLut12Bit - peek an 16-bit value out of an 12-bit LUT.
CiLut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

CiLutBank0 - fill LUT bank 0
CiLutBank1 - fill LUT bank 1
CiLutBank2 - fill LUT bank 2
CiLutBank3 - fill LUT bank 3

Lane

One or more LUT lanes ORed together:

CiLutLane0 - fill LUT lane 0
CiLutLane1- fill LUT lane 1
CiLutLane2 - fill LUT lane 2
CiLutLane3 - fill LUT lane 3

Addr

LUT address.

NumEntries

Number of LUT entries to fill.

Val

Fill value.

Returns

CI_OK	Function successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUPPORTED	This function not support for this board type.
CISYS_ERROR_UNKNOWN_PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_NO_BIG_LUTS	RoadRunner board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_FILL_ERR	LUT fill failed.
RV_BAD_BANK	Illegal LUT bank.
RV_BAD_LUT_ADDR	Illegal LUT address.
RV_LUT_FILL_ERR	LUT fill failed.

Comments

LUT definitions are declared in CiDef.h.

16.7 CiLutRamp

Prototype BFRC CiLutRamp(Bd *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *StartAddr*, BFU32 *EndAddr*, BFU32 *StartVal*, BFU32 *EndVal*)

Description Fills a LUT with a ramp.

Parameters

Board

Bd Board ID.

Mode

LUT Mode:

CiLut8Bit - peek an 8-bit value out of an 8-bit LUT.
CiLut12Bit - peek an 16-bit value out of an 12-bit LUT.
CiLut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

CiLutBank0 - ramp LUT bank 0
CiLutBank1 - ramp LUT bank 1
CiLutBank2 - ramp LUT bank 2
CiLutBank3 - ramp LUT bank 3

Lane

One or more LUT lanes ORed together:

CiLutLane0 - ramp LUT lane 0
CiLutLane1- ramp LUT lane 1
CiLutLane2 - ramp LUT lane 2
CiLutLane3 - ramp LUT lane 3

StartAddr

LUT start address.

EndAddr

LUT end address.

StartVal

LUT start value.

EndVal

LUT end value.

Returns

CI_OK	Function successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	This function not support for this board type.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_NO_BIG_LUTS	RoadRunner board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_FILL_ERR	LUT fill failed.
RV_BAD_BANK	Illegal LUT bank.
RV_BAD_LUT_ADDR	Illegal LUT address.
RV_LUT_RAMP_ERR	LUT ramp failed.

Comments

LUT definitions are declared in CiDef.h.

Ci Acquisition

Chapter 17

17.1 Introduction

The Acquisition Functions are some of the most important in the BitFlow SDK. While the initialization functions set up the board's registers for a particular camera, these functions do most of the work required to get the board ready to DMA the images to memory.

The functions are organized into three groups:

Setup functions
Command function
Clean up functions

The concept here is that the setup functions are time and CPU intensive, so they should be called before any time critical processing has begun. In a sense, these are extensions of the initialization process. Once the setup functions are called for a particular buffer, they need not be called again.

The command function is designed to be used during time critical operations, and require minimal CPU time. They can be told to return immediately so that other operations can be performed simultaneously with acquisition. The command function can be called over and over, as many times as needed, to acquire into the buffers locked down in the setup functions.

The cleanup functions free up any resources allocated in the setup functions, and put the DMA engine in an idle mode. If the clean up functions are not called, then it is possible that large amounts of memory will not be freed up.

For example, the basic flow of a program would be:

```
CiBrdOpen
CiAqSetup

Loop
{
   CiAqCommand
}

CiAqCleanup
CiBrdClose
```

The bulk of the work is done in the CiAqSetup functions. These functions create a scatter gather table based on the virtual memory address, called a relative QTab.

The relative QTab is passed to the kernel driver, where the destination buffer is locked down (so that it cannot be paged to disk) and the physical address are determined for each page of the buffer (NT usually uses 4096 byte pages). These physical addresses are used to build a physical QTab. This physical QTab is then written to the board in preparation scatter gather DMAing.

Finally, the DMA engine is initialized and started. Again, this function need be called only once, for a particular destination buffer.

This function also supports setting up acquisition to a set of up to four buffers. In this case, the setup functions are called multiple times, once for each buffer in the set. Whenever an application is finished acquiring to a buffer or a set of buffers, one of the CiAqCleanUp functions must be called. You cannot call CiAqSetup for a different buffer or set of buffers before calling CiAqCleanUp for the previous buffer or set of buffers. When using buffers sets, CiAqCleanUp need only be called once to cleanup from any number of calls to CiAqSetup.

The CiAqCommand can be called either synchronously or asynchronously. In the synchronous case, the function does not return until the command has completed. In the asynchronous case, the function returns as soon as the command has been issued to the board. If you need to synchronize your process with the acquisition, you can use the CiAqWaitDone function or you can use the signaling system. Signaling is the best way to synchronize to the end of frame signals as they do not take any CPU cycles.

17.2 CiAqSetup

Prototype

BFRC CiAqSetup(Bd *Board*, PBFVOID *pDest*, BFU32 *DestSize*, BFS32 *Stride*, BFU32 *DestType*,BFU32 *LutBank*,BFU32 *LutMode*,BFU8 *QuadBank*, BFBOOL *FirstBank,* BFU32 *QTabMode,* BFU32 *AqEngine*)

Description

Sets up the board for acquisition to a host buffer. This function must be called before any acquisition command is issued.

Parameters

Board

Handle to board.

pDest

A void pointer to the destination buffer (already allocated).

DestSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in bytes, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

DestType

Note: RoadRunner specific.

Type of destination memory:

CiDMADataMem - host memory
CiDMABitmap - display memory

LutBank

The LUT bank to pass the image through:

CiLutBank0 - LUT bank 0
CiLutBank1 - LUT bank 1
CiLutBank2 - LUT bank 2
CiLutBank3 - LUT bank 3
CiLutBypass - bypass LUTs

LutMode

The mode of the LUT to use:

CiLut8Bit - LUT bank 0
CiLut12Bit - LUT bank 1
CiLut16Bit - LUT bank 2

QuadBank

The Quad bank used to store the QTABs build by this function:

CiQTabBank0 - Quad bank 0
CiQTabBank1 - Quad bank 1
CiQTabBank2 - Quad bank 2
CiQTabBank3 - Quad bank 3

FirstBank

For acquisition to single buffer, set to TRUE.

For acquisition using two or more ping-pong buffers, this parameter is used to indicate which buffer will be acquired into first:

TRUE - for first bank to acquire into.
FALSE - for subsequent banks.

QtabMode

The QTab mode:

CiQTabModeOneBank - the entire quad table is one bank.
CiQTabModeTwoBanks - the quad table is divided into two banks.
CiQTabModeFourBanks - the quad table is divided into four banks.

AqEngine

The acquisition engine to set up:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_ UNKNOWN_PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.

R64_BAD_ALLOC	Resources required for this operation could not be allocated.
R64_CON_QTAB_BANK_ERR	The QuadBank parameter is invalid.
R64_AQSETUP_FAIL	Other failure.
RV_BAD_ALLOC	Resources required for this operation could not be allocated.
RV_CON_QTAB_BANK_ERR	The QuadBank parameter is invalid.
RV_AQSETUP_FAIL	Other failure.

Comments

This function sets up the board's acquisition systems for acquisition to host. It will set up QTABs (relative and physical) and write them to the board. The QTABs are based on the current camera pointer in the board structure. This function only needs to be called once, before acquisition begins. It does not need to be called again unless CiAqCleanUp is called. CiAqCleanUp should be called when done acquiring in order to free up resources used by this process. The only reason to call this function again is to acquire into a different host buffer or acquire with a different type of camera. Once this function is called, the function CiAqCommand is used to snap, grab, freeze or abort acquisition.

The two acquisition engines on the Raven are completely independent. Each engine can be in a different acquisition state (snap, grab, abort, or freeze). It is recommend that both engines be set up before issuing a command to either one. Typically, each engine works with its own camera (or set of cameras). In other words, if only one camera is connected to the board, only one acquisition engine is needed. The connections between camera ports and acquisition engines is established in the camera configuration file and/or the registers. In general, the camera file will dictate whether or not one or two acquisition engines are available or needed.

If you are setting up acquisition for N buffers (where N can be up to 4), call this function N times, once for each buffer. For the first call to this function, you must set *FirstBank* = TRUE. In all subsequent calls (for the same set of buffers) this parameter must set to FALSE. If you are using two buffers, then *QTabMode* should equal CiQTabModeTwoBanks, and for three or four buffers, *QTabMode* should equal CiQTabModeFourBanks. Use the function CiAqNextBankSet to tell the board which buffer of the set should be the next buffer acquired into. When you are setting up a set of buffers, you will call this function two to four times, but you only need to call CiAqCleanUp once per acquisition engine to clean up the resources for the whole set of buffers.

When using board QTABs with the Raven, there is one block of memory that can be divided into one, two or four memory banks. The two acquisition engines on the Raven share this block of memory. When using both acquisition engines, at a minimum, the QTab bank mode must use two banks where one engine is using one memory bank and the other engine is using the other bank. When using both acquisition engines with ping-ponging between two memory banks, the QTab bank mode must use four banks, where each engine is using two memory banks and ping-pongs between the two.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

17.3 CiAqSetup2Brds

Prototype

BFRC CiAqSetup2Brds(Bd *Board*1, Bd *Board2*, PBFVOID *pDest*, BFU32 *DestSize*, BFS32 *Stride*, BFU32 *DestType*,BFU32 *LutBank*,BFU32 *LutMode*,BFU8 *QuadBank*, BFBOOL *FirstBank,* BFU32 *QTabMode,* BFU32 *AqEngine*)

Description

Sets up two boards for acquisition to the same host buffer. This function must be called before any acquisition command is issued. The function is for use with special "2x" firmware.

Parameters

Board1

Handle to board one.

Board2

Handle to board two.

pDest

A void pointer to the destination buffer (already allocated).

DestSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in bytes, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

DestType

Note: RoadRunner specific.

Type of destination memory:

CiDMADataMem - host memory
CiDMABitmap - display memory

LutBank

The LUT bank to pass the image through:

CiLutBank0 - LUT bank 0
CiLutBank1 - LUT bank 1
CiLutBank2 - LUT bank 2

CiLutBank3 - LUT bank 3
CiLutBypass - bypass LUTs

LutMode

The mode of the LUT to use:

CiLut8Bit - LUT bank 0
CiLut12Bit - LUT bank 1
CiLut16Bit - LUT bank 2

QuadBank

The Quad bank used to store the QTABs build by this function:

CiQTabBank0 - Quad bank 0
CiQTabBank1 - Quad bank 1
CiQTabBank2 - Quad bank 2
CiQTabBank3 - Quad bank 3

FirstBank

For acquisition to single buffer, set to TRUE.

For acquisition using two or more ping-pong buffers, this parameter is used to indicate which buffer will be acquired into first:

TRUE - for first bank to acquire into.
FALSE - for subsequent banks.

QtabMode

The QTab mode:

CiQTabModeOneBank - the entire quad table is one bank.
CiQTabModeTwoBanks - the quad table is divided into two banks.
CiQTabModeFourBanks - the quad table is divided into four banks.

AqEngine

The acquisition engine to set up:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.

CISYS_ERROR_ UNKNOWN_PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R64_BAD_ALLOC	Resources required for this operation could not be allocated.
R64_CON_QTAB_BANK_ ERR	The QuadBank parameter is invalid.
R64_AQSETUP_FAIL	Other failure.
RV_BAD_ALLOC	Resources required for this operation could not be allocated.
RV_CON_QTAB_BANK_ ERR	The QuadBank parameter is invalid.
RV_AQSETUP_FAIL	Other failure.

Comments

This function is for use with special "2x" firmware. This firmware sets the board up to use two DMA engines with one camera. This is for cameras that are too fast for one DMA engine. Both *Board1* and *Board2* must be opened an initialized separately. Both boards must be set up with the same basic camera configuration file.

Both acquistions engines are programed to DMA into the same host buffers as indicated by the *pDest* parameter. Generally, one board DMAs the even lines to the host buffer and the other board DMAs the odd lines.

Once acquisition is setup using this function, only the master board needs to be controlled using the acquisition command functions. When the 2x firmware is used, the slave acquisition commands are control by the master.

When you are finished acquiring to this buffer, you must call CiAqCleanUp2Brds to release the resources allocated for this board.

For more information on the other parameters use by this functoin, see the function CiAqSetup.

17.4 CiAqCommand

Prototype BFRC CiAqCommand(Bd *Board*, BFU32 *Command*, BFU32 *Mode*, BFU8 *Quad-Bank,* BFU32 *AqEngine*)

Description Once the board is set up for acquisition with CiAqSetup, this function issues the actual acquisition command.

Parameters

Board

Handle to board.

Command

Acquisition command to initiate:

CiConGrab - starting at the beginning of the next frame, acquire every frame.
CiConSnap - starting at the beginning of the next frame, acquire one frame.
CiConFreeze - stop acquiring at the end of the current frame. If in between frames, do not acquire any more frames.
CiConAbort - stop acquiring immediately. If in the middle of the frame, the rest of the frame will not be acquired.
CiConReset - reset conditions after an abort or overflow. The board is set up as it was when CiAqSetup was called.

Mode

This function can operate in two modes:

CiConAsync - as soon as the command is issued return.
CiConWait - wait for the current command to complete. For a snap, the function will return when the entire frame has been acquired into memory. For a grab, the function will wait until the first frame has begun to be acquired. For a freeze, the function waits for the current frame to end. All other commands return immediately.

QuadBank

QTab bank to start acquisition with:

CiQTabBank0 - Quad bank 0
CiQTabBank1 - Quad bank 1
CiQTabBank2 - Quad bank 2
CiQTabBank3 - Quad bank 3

AqEngine

The acquisition engine to issue the command to:

AqEngJ - the J engine.
AqEngK - the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R64_AQ_NOT_SETUP	CiAqSetup has not yet been called and the board is not ready for an acquisition command.
R64_BAD_AQ_CMD	A snap or grab command has already been issued and the board is already acquiring.
R64_BAD_STOP	The function was unable to reset the board.
R64_CON_QTAB_BANK_ERR	The *QuadBank* parameter is incorrect.
R64_BAD_DMA_SETUP	The board has not been set up properly for DMA.
RV_AQSTRT_TIMEOUT	A time-out occurred waiting for acquisition to begin.
RV_AQEND_TIMEOUT	A time-out occurred waiting for acquisition to end.
RV_AQ_NOT_SETUP	CiAqSetup has not yet been called and the board is not ready for an acquisition command.
RV_BAD_AQ_CMD	A snap or grab command has already been issued and the board is already acquiring.
RV_BAD_STOP	The function was unable to reset the board.
RV_CON_QTAB_BANK_ERR	The *QuadBank* parameter is incorrect.
RV_BAD_DMA_SETUP	The board has not been set up properly for DMA.
RV_AQSTRT_TIMEOUT	A time-out occurred waiting for acquisition to begin.
RV_AQEND_TIMEOUT	A time-out occurred waiting for acquisition to end.

Comments

This function can only be called after CiAqSetup is called. CiAqSetup need only be called once for any number and combination of calls to CiAqCommand. Basically, you call CiAqSetup once for a given host buffer, then call CiAqCommand as many times as you need to get data into that buffer. Call CiAqCleanUp when you are done acquiring into that buffer. Then the procedure starts over again for the next buffer.

The CiAqXXXX commands handle both DMA and camera acquisition. No other commands are needed to handle the process of acquiring into memory.

If you call this function with *Mode* = CiConWait, it will wait for the acquisition to complete, in the case of a snap or freeze command, or wait for the acquisition to begin, in the case of a grab command. This is an efficient wait that consumes minimal CPU cycles. The function will return when the last pixel has been DMAed into memory. Alternatively, you can call the function with *Mode* = CiConAsync, and the function will return as soon as the command has been issued. You can find out how much data has been DMAed by calling CiAqProgress. You can also just wait for the end of acquisition by calling CiAqWaitDone.

The functions mentioned above use the SDK's signaling system to efficiently wait for events. If you wish to have a higher level of control you can call the CiSignalXXXX functions yourself. These functions use a signaling system that allow processes to be notified of board events and interrupts. For acquisition, wait for the CiIntTypeEOD signal. This signal occurs at the end of every frame (or field for interlaced cameras), in both grab and snap mode. This signal occurs when the last pixel is DMAed into memory.

Calling this function with *Command* = CiConAbort will stop acquisition immediately. The acquisition process can be left anywhere in the frame. You must call this function with *Command* = CiConReset before any more acquire commands can be issued. Alternatively, you can call CiAqCleanUp and start over with CiAqSetup.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

17.5 CiAqCleanUp

Prototype BFRC CiAqCleanUp(Bd *Board,* BFU32 *AqEngine*)

Description Frees all resources used by the acquisition process. Makes sure the board is in a stable state.

Parameters

Board

Handle to board.

AqEngine

The acquisition engine to clean up:

AqEngJ - the J engine.
AqEngK - the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function frees all of the resources that were allocated in CiAqSetup for a particular acquisition engine. If you called CiAqSetup for a set of buffers, you only need to call this function once to clean up resource for the entire set. Do not call this function unless you have already called CiAqSetup, and unless you are finished acquiring into the current buffer.

This function does not free the destination buffer passed to CiAqSetup in the *pDest* parameter.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

17.6 CiAqCleanUp2Brds

Prototype BFRC CiAqCleanUp2Brds(Bd *Board1*, B2 *Board2*, BFU32 *AqEngine*)

Description Frees all resources used by the acquisition process when CiAqSetup2Brds was called. The function is for use with special "2x" firmware.

Parameters

Board1

Handle to board one.

Board2

Handle to board two.

AqEngine

The acquisition engine to clean up:

AqEngJ - the J engine.
AqEngK - the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function frees all of the resources that were allocated in CiAqSetup2Brds. Do not call this function unless you have already called CiAqSetup2Brds, and unless you are finished acquiring into the current buffer.

This function does not free the destination buffer passed to CiAqSetup2Brds in the *pDest* parameter.

For a detailed description of the parameters used by this function, see CiAqCleanup.

17.7 CiAqWaitDone

Prototype BFRC CiAqWaitDone(Bd *Board,* BFU32 *AqEngine*)

Description Waits for the current acquisition to complete

Parameters *Board*

Handle to board.

AqEngine

The acquisition engine to clean up:

AqEngJ - the J engine.
AqEngK - the K engine.

Returns

CI_OK	The current acquisition has completed.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
R64_AQ_NOT_SETUP	The acquisition process has not been set up yet.
R64_BAD_WAIT	The board is currently in grab mode and acquisition will not end, or there is another acquisition command pending after this one is completed.
R64_AQEND_TIMEOUT	The acquisition time-out expired before the acquisition command completed.
RV_AQ_NOT_SETUP	The acquisition process has not been set up yet.
RV_BAD_WAIT	The board is currently in grab mode and acquisition will not end, or there is another acquisition command pending after this one is completed.
RV_AQEND_TIMEOUT	The acquisition time-out expired before the acquisition command completed.

Comments This function efficiently waits for the current acquisition of the given *AqEngine* to complete. The completion is signaled by the last pixel being DMAed into memory. The function will return with a time-out error if the acquisition has not been completed by the designated acquisition time-out amount. This time is normally set in the camera configuration file, but can be changed in software as well, see CiCamSetTimeout. This function will return immediately if the acquisition has already completed. This function will also return immediately (with an error code), if the board is in a state where acquisition will not complete without further acquisition commands, such as when the board is in grab mode.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

17.8 CiAqNextBankSet

Prototype BFRC CiAqNextBankSet(Bd *Board*, BFU8 *QuadBank,* BFU32 *AqEngine*)

Description For double buffered acquisition, this function sets up the board for the next QTab bank to use for acquisition.

Parameters

Board

Handle to board.

QuadBank

Next QTab bank to use for acquire - must be 0 or 1:

CiQTabBank0 - Quad bank 0
CiQTabBank1 - Quad bank 1
CiQTabBank2 - Quad bank 2
CiQTabBank3 - Quad bank 3

AqEngine

The acquisition engine to check to progress on:

AqEngJ - the J engine.
AqEngK - the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	Function no supported on the installed frame grabber model.
CISYS_ERROR_UNKNOWN_ PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.

Comments This function sets the next QTab bank to use for acquisition. The RoadRunner has two QTab banks that can be loaded for two destinations that can be loaded for multiple destinations by calling CiAqSetup more than once. By default, the board will use the same bank continuously. This function can be called any time during the current frame, the bank switch will only happen at the beginning of the next frame.

You must call CiAqSetup more than once (each time with the parameter *QuadBank* set to a different bank), before using this function. Switching to a QTab bank that has not been loaded will cause unpredictable behavior.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

Note: This function should only be used with the R3 and RoadRunner families.

17.9 CiAqFrameSize

Prototype BFRC CiAqFrameSize(Bd *Board*, BFU32 *XSize*, BFU32 *YSize*, BFU32 *AqEngine*)

Description This function provides the ability to change the image height and image width.

Parameters

Board

Handle to board.

XSize

The image width in pixels.

YSize

The image height in lines.

AqEngine

The Acquisition Engine to Set up:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_	An invalid parameter was specified.
PARAMETER	
R2_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R2_CAM_SUPPORT	Cam file being used is not supported by this function.
R2_HCTAB_X16	Pixel clock divided by 16 is not supported.
R2_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R2_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/ or HStop.
BF_BAD_ALLOC	Couldn't allocate memory.
R64_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.

R64_CAM_SUPPORT	Cam file being used is not supported by this function.
R64_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R64_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
R64_BAD_CNF_FILE	Could not determine the pixels per clock from the camera file.

Comments

This function gives the ability to change the size of the image being acquired on the fly, from software. With this function the size of the frame can be changed on the fly, without the use of camera files. This function is limited to use with only free run camera files, and may not work with sophisticated camera files.

This function can be called before CiAqSetup and the new size will overwrite the size specified by the camera file. To change the size after CiAqSetup has been called CiAqCleanup must be called then CiAqFrameSize and CiAqSetup. The following is an example of the order needed to change the size of the frame after CiAqSetup has been called:

```
// Stop acquisition
CiAqCleanUp
CiAqFrameSize
CiAqSetup
// Begin acquisition
```

For a complete example on how to use the CiAqFrameSize function, see the CiChangeSize example included in the SDK.

For the R2, the minimum XSize is 4 and a minimum YSize of 2. The maximum YSize is 32768 and the maximum XSize is 8192. This function will return a R2_BAD_FRM_SIZE error for any of these problems. Another precaution to take is that the XSize needs to be a multiple of 4. Any XSize value that is not a multiple of 4 will give a R2_BAD_FRM_SIZE error. The R2 only supports a pixel clock divided by 4. If the pixel clock divided by 16 is being used, error R2_HCTAB_X16 will be returned.

For the R64, the minimum XSize is 8 pixels and a minimum YSize of 1 line. The maximum YSize and XSize is 131,072 lines and pixels. This function will return a R64_BAD_FRM_SIZE error for any problems with the size of the frame. Another precaution to take is that the XSize needs to be a multiple of the pixels per clock. Any XSize value that is not a multiple of the pixels per clock will give a R64_BAD_FRM_SIZE error.

It is left up to the user not to exceed the sensor size of the camera. For example if the user is using a area scan camera with a sensor size of 640x480 and tries to make the frame size 800x600, this function will try to acquire the 800x600 frame size even though the camera can not provide it. The user will end up with a scrambled or unstable image if this occurs.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

17.10 CiAqLastLine

Prototype BFRC CiAqLastLine(Bd *Board*, PBFU32 *pCurLine*, BFU32 *AqEngine*)

Description Returns the number of lines that have been acquired.

Parameters

Board

Handle to board.

pCurLine

Pointer to the last line number.

AqEngine

The Acquisition Engine to Set up:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successfully.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The model does not support this function.
CISYS_ERROR_UNKNOWN_ PARAMETER	An invalid parameter was specified.

Comments This functions returns the line number of the last line in the frame. The returned value is actually the Vertical CTAB counter value for the last line. If the camera being used is a line scan cameras then this value will be equivalent to the line number. However, for area scan camera the start of the vertical active region will have to be subtracted from the returned value (usually the vertical active region starts at 0x1000).

This function is most useful when acquiring variable sized images and thus the frame size is unknown. This function will return the value from the last frame up until the end of the following frame. In other words, the value of the last line stays constant for the entire duration of the next frame. Once the next frame ends, then the last line is the value for that frame.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

17.11 CiAqReengage

Prototype BFRC CiAqReengage(Bd *Board*, BFU8 *QuadBank*, BFU32 *AqEngine*)

Description Engages the physical QTab for the given bank.

Parameters

Board

Handle to board.

QuadBank

The next physical QTab bank to use.

AqEngine

The Acquisition Engine to Set up:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_UNKNOWN_	An invalid parameter was specified.
PARAMETER	
R64_BAD_CON_PARAM	QuadBank is not equal to R64QTabBank0 or R64QTabBank1
R64_AQ_NOT_SETUP	R64AqSetup has not yet been called and the board is not ready for an acquisition command.
BF_NULL_POINTER	*ChainArray* is NULL.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board.
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments

This function is used to engage the physical QTab for the bank specified by the Quad-Bank parameter. This function only needs to be used if the acquisition or the DMA is aborted in the middle of the frame (for example, when using start-stop triggering). This function is intended to be used with qtabs on the host. However, calling it with board qtabs will not cause any problems.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

Currently CiAqReengage is not supported by the Raven.

17.12 CiAqROISet

Prototype BFRC CiAqROISet(Bd *Board*, BFU32 *XOffset*, BFU32 *YOffset*, BFU32 *XSize*, BFU32 *YSize*, BFU32 *AqEngine*)

Description This function provides the ability to change the image height and image width.

Parameters

Board

Handle to board.

XOffset

The number of pixels to offset in the x-axis.

YOffset

The number of pixels to offset in the y-axis.

XSize

The value to change the XSize too.

YSize

The value to change the YSize too.

AqEngine

The Acquisition Engine to use:

AqEngJ - use the J engine.
AqEngK - use the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_	An invalid parameter was specified.
PARAMETER	
R2_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R2_CAM_SUPPORT	Cam file being used is not supported by this function.
R2_HCTAB_X16	Pixel clock divided by 16 is not supported.

R2_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R2_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
BF_BAD_ALLOC	Couldn't allocate memory.
R64_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R64_CAM_SUPPORT	Cam file being used is not supported by this function.
R64_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R64_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
R64_BAD_CNF_FILE	Could not determine the pixels per clock from the camera file.
RV_BAD_FRM_SIZE	Invalid frame size. The frame can be to big or small, or the XSize is not a multiple of 4.
RV_CAM_SUPPORT	Cam file being used is not supported by this function.

Comments

This function gives the ability to change the size of the image being acquired on the fly, from software. With this function the size of the frame can be changed on the fly, without the use of camera files. This function is limited to use with only free run camera files, and may not work with sophisticated camera files.

This function can be called before CiAqSetup and the new size will overwrite the size specified by the camera file. To change the size after CiAqSetup has been called CiAqCleanup must be called then CiAqFrameSize and CiAqSetup. The following is an example of the order needed to change the size of the frame after CiAqSetup has been called:

```
// Stop acquisition
CiAqCleanUp
CiAqFrameSize
CiAqSetup
// Begin acquisition
```

For the R2, the minimum XSize is 4 and a minimum YSize of 2. The maximum YSize is 32768 and the maximum XSize is 8192. This function will return a R2_BAD_FRM_SIZE error for any of these problems. Another precaution to take is that the XSize needs to be a multiple of 4. Any XSize value that is not a multiple of 4 will give a R2_BAD_FRM_SIZE error. The R2 only supports a pixel clock divided by 4. If the pixel clock divided by 16 is being used, error R2_HCTAB_X16 will be returned.

For the R64, the minimum XSize is 8 pixels and a minimum YSize of 1 line. The maximum YSize and XSize is 131,072 lines and pixels. This function will return a R64_BAD_FRM_SIZE error for any problems with the size of the frame. Another precaution to take is that the XSize needs to be a multiple of the pixels per clock. Any XSize value that is not a multiple of the pixels per clock will give a R64_BAD_FRM_SIZE error.

It is left up to the user not to exceed the sensor size of the camera. For example if the user is using a area scan camera with a sensor size of 640x480 and tries to make the frame size 800x600, this function will try to acquire the 800x600 frame size even though the camera can not provide it. The user will end up with a scrambled or unstable image if this occurs.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

Ci Mid-Level Control Functions

Chapter 18

18.1 Introduction

These functions are used to control the board at a lower level than the high level functions, for example, CiAqCommand. In general, an application should not need to use these functions unless special circumstances exist. These functions talk directly to the hardware and make no assumptions about how the rest of the board is set up. Generally, it is a bad idea to mix high-level functions and these mid-level functions.

18.2 CiConAqCommand

Prototype BFRC CiConAqCommand(Bd *Board*, BFU32 *Command,* BFU32 *AqEngine*)

Description Sends an acquisition command to the board.

Parameters

Board

Handle to board.

Command

Command send to board:

CiConSnap - snap one frame.
CiConGrab - start continuous acquisition.
CiConFreeze - stop continuous acquisition at the end of the current frame.
CiConAbort - stop acquisition immediately.

AqEngine

The acquisition engine to send command to:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Unknown *Command* parameter.
R64_BAD_CON_PARAM	Unknown *Command* parameter.
GN2_BAD_CON_PARAM	Unknown *Command* parameter.

Comments This function sends an acquisition command directly to the hardware. This is a low-level function and makes no assumptions about the state of the rest of the board.

This command returns immediately.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

18.3 CiConAqStatus

Prototype BFRC CiConAqStatus(Bd *Board*, PBFU32 *pStatus* BFU32 *AqEngine*)

Description Gets the current acquisition state of the board.

Parameters

Board

Handle to board.

PStatus

When this function returns it contains the status of the board. The status will be one of the following:

CiConFreeze - the board is not acquiring.
CiConSnap - the board is currently acquiring one frame.
CiConGrab - the board is currently in continuous acquisition mode.

AqEngine

The acquisition engine get the status of:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.

Comments This function returns the current acquisition status of the board.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

18.4 CiConInt

Prototype BFRC CiConInt(Bd *Board*, BFU32 *IntType*, BFU32 *Action*)

Description Disables or enables individual hardware interrupts.

Parameters

Board

Handle to board.

IntType

Type of interrupt. See Table 15-1 for a complete list of interrupt types.

Action

Indicates whether to enable or disable the interrupt:

CiConEnable - enable the interrupt.
CiConDisable - disable the interrupt.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
R2_BAD_CON_PARAM	Either the parameter *IntType* or *Action* is unknown.
R64_BAD_CON_PARAM	Either the parameter *IntType* or *Action* is unknown.
GN2_BAD_CON_PARAM	Either the parameter *IntType* or *Action* is unknown.

Comments This function enables or disables the specified hardware interrupt for being invoked on the PCI bus. The driver always has an interrupt service (ISR) routine ready to handle any interrupts that come in. The driver's ISR will automatically reset the appropriate interrupt bits on the board when an interrupt occurs.

To receive notification of interrupts at the user application level, use the signaling system (see the CiSignalXXXX functions). These functions automatically enable the appropriate interrupt when the signal is created, so you do not have to call this function to use an interrupt with the signaling system. However, you can use this function to enable and disable interrupts, based on your application needs, without creating and destroying signals. As a general rule, you should disable any interrupts that you are not using. Every interrupt uses a certain amount of CPU time, even if no application is waiting for it.

When the board is initialized, by default, all interrupts are turned off.

Note: Not all board families have all of the listed interrupts. Please check the appropriate hardware manuals for complete details.

18.5 CiConVTrigModeSet

Prototype BFRC CiConVTrigModeSet(Bd *Board*, BFU32 *TrigMode*, BFU32 *TrigAssignments*, BFU32 *TrigAPolarity*, BFU32 *TrigBPolarity*)

Description Sets the trigger mode and polarities for the acquisition engine(s).

Parameters *Board*

Handle to board.

TrigMode

Trigger mode to put the board into:

CITrigIgnore - no changes to the trigger mode will be made.
CiTrigFreeRun - no trigger is used, board free runs.
CiTrigOneShotSelfTriggered - self triggering one shot mode.
CiTrigOneShot - one shot mode, for asynchronously resettable cameras.
CiTrigAqCmd - triggered acquire command mode, non-resettable cameras.
CiTrigSnapQualified - designed for use with cameras that cannot be asynchronously reset. When the board is in this mode, it will snap one frame every time the trigger asserts.
CiTrigContinuousData - for continuous data sources.
CiTrigContinuousDataQualified - This is similar to CiTrigContinuousData. However the data is qualified with another incoming signal.
CiTrigOneShotStartAStopA - one shot mode, where frame starts with the assertion of trigger A and ends de-assertion of trigger A.
CiTrigOneShotStartAStopALevel - exactly like CiTrigOneShotStartAStopA except that if the trigger is still high after the current frame has been acquired, the board will acquire the next frame.
CiTrigNTGOneShot- Trigger triggers NTG.
BFTrigTriggeredGrab - one shot mode, where board goes into grab modes with the assertion of trigger A and freezes acquisition with the de-assertion of trigger A.

TrigAssignments

Note: This parameter is only used with the Raven. For all boards this parameter should be set to zero.

Controls which triggers go to which acquisition engine:

CiTrigJandKSameSource - both J and K are run by same trigger(s).
CiTrigJbyAandKbyB - J is run by TRIGGERA and K by TRIGGERB.
CiTrigJbyAandKFreerun - J is trigger and K free runs.

TrigAPolarity

Polarity for trigger A:

CiTrigAssertedHigh - TRIGGERA is asserted on rising edge.
CiTrigAssertedLow - TRIGGERA is asserted on falling edge.

TrigBPolarity

Note: This parameter is only used with the Raven. For all boards this parameter should be set to zero.

Polarity for trigger B:

CiTrigAssertedHigh - TRIGGERB is asserted on rising edge.
CiTrigAssertedLow - TRIGGERB is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments

Some board families have two trigger inputs, others have only one. The effect each trigger has on the board depends on the trigger mode, which is controlled by this function. Each trigger can be caused by an external hardware signal (on TRIGGERA and TRIGGERB pins respectively) or an internal software trigger (see the function CiConSwTrig). Both types of triggers effect the board in the same way. However, the parameters: *TrigAPolarity* and *TrigBPolarity* only have meaning with respect to external hardware triggers. The effect of each trigger for the various modes is explained in the following tables.

This function works in conjunction with the camera configuration files. It is important to understand that not all cameras support all triggering modes. Usually a particular camera will only support one or two triggering modes. Furthermore, a different camera configuration file is usually needed for each triggering mode. For example, a camera will almost always have a free running configuration file, useful for set up and offline testing. A camera may also have a one shot file, which would be used in time-

critical applications. You cannot usually put the board, set up by the free running file, into one shot mode because the latter mode requires special triggering signals to be sent to the camera. However, you can put the board, set up by a one shot file, into self triggering one shot mode. This is useful for camera set up and system debugging.

The exception to the paragraph above is the triggered acquire command mode, which will work with all cameras. This mode is really no different than just issuing an acquisition command at a specific point in time in the future. When the board is in this mode, an acquisition command is written by the host but not latched. Basically, the board is armed but does not acquire any data. When the trigger is asserted the command latches. Once the command is latched, it acts as it normally does, that is, the board starts acquiring data at the start of the next frame from the camera. The only acquisition commands that are affected are snap and grab. The freeze and abort commands work normally, and do not need a trigger to be latched. The disadvantage of this mode is that it can add up to a frame time of latency to any trigger, because the camera's timing is not being reset.

If you want to find out what mode the board is in, call the function CiConVTrigModeGet.

Note: This function only controls how the board is vertically triggered. Vertical triggers cause the board to acquire a whole frame from an area camera or a number of lines from a line scan camera.

Note: You must enable the connection of the external trigger with the acquisition engines with the function CiConExTrigConnect. The software triggers are always available.

Note: Not all combinations of TrigMode and TrigAssignments are possible

Note: In the mode, CiTrigOneShotSelfTriggered, the acquisition engine(s) continuously self trigger(s) at the maximum possible rate

Table 18-1 shows which modes are available on which board families.

Table 18-1 Trigger Mode Availability

Mode	R2/R3	Karbon/ Alta/Neon	Aon/Axion/ Cyton
CiTrigFreeRun	Yes	Yes	Yes
CiTrigAqCmd	Yes	Yes	No
CiTrigAqCmdStartStop	Yes	No	No
CiTrigOneShot	Yes	Yes	Yes
CiTrigOneShotSelfTriggered	Yes	Yes	Yes
CiTrigOneShotStartAStopA	Yes	Yes	Yes

Table 18-1 Trigger Mode Availability

Mode	R2/R3	Karbon/ Alta/Neon	Aon/Axion/ Cyton
CiTrigOneShotStartAStopALevel	No	Yes	Yes
CiTriggerSnap	No	Yes	No
CiTrigContinuousData	Yes	Yes	No
CiTrigContinuousDataQualified	No	Yes	No

Table 18-2 and Table 18-3 show the effects of the assertion and de-assertion of the trigger in all of these various modes for the Road Runner/R3.

Table 18-2 Assertion of Triggers for the Road Runner/R3

TrigMode	Trigger A asserts	Trigger B asserts
CiTrigFreeRun	No effect	No effect
CiTrigAqCmd	Last acquisition command (Snap, Grab, etc.) is initiated.	No effect
CiTrigAqCmdStartStop	Board goes into grab mode.	No effect
CiTrigOneShot	One frame is acquired from camera.	No effect
CiTrigOneShotSelfTriggered	No effect	No effect
CiTrigOneShotStartAStopA	Board starts acquiring lines. Will stop when max is reached or Trigger A deasserts.	No effect
CiTrigOneShotStartAStopB	Board starts acquiring lines. Will stop when max is reached or Trigger B asserts.	Board stops acquiring lines.
CiTrigContinuousData	Board starts acquiring continuous date.	No effect

Table 18-3 De-assertion of Triggers for Road Runner/R3

TrigMode	Trigger A deasserts
CiTrigOneShotStartAStopA	Board Stops acquiring lines.
CiTrigAqCmdStartStop	Board goes into freeze mode.

Table 18-4 and Table 18-5 show the effects of the assertion and de-assertion of the trigger in all of these various modes for theR46/Karbon/Alta/Neon.

Table 18-4 Assertion of Triggers for the R64/Karbon/Alta/Neon

TrigMode	Trigger A asserts
CiTrigFreeRun	No effect
CiTrigOneShot	One frame is acquired from camera.
CiTrigOneShotSelfTriggered	No effect
CiTrigOneShotStartAStopA	Board starts acquiring lines. Will stop when max is reached or Trigger A de-asserts.
CiTrigOneShotStartAStopALevel	Board starts acquiring lines. Will stop when trigger A de-asserts.
CiTriggeredSnap	Board starts acquiring lines. Will stop when max is reached or Trigger B asserts.
CiTrigContinuousData	Board starts acquiring continuous date.
CiTrigContinuousDataQualified	Board starts acquiring continuous date.

Table 18-5 De-assertion of Triggers for R64/Karbon/Alta/Neon

TrigMode	Trigger A deasserts
CiTrigOneShotStartAStopA	Board Stops acquiring lines.
CiTrigAqCmdStartStop	Board goes into freeze mode.

Table 18-6 and Table 18-7 show the effects of the assertion and de-assertion of the trigger in all of these various modes for the Aon/Axion/Cyton.

Table 18-6 Assertion of Triggers for the Aon/Axion/Cyton

TrigMode	Trigger A asserts
CiTrigFreeRun	No effect
CiTrigOneShot	One frame is acquired from camera.
CiTrigOneShotSelfTriggered	No effect
CiTrigOneShotStartAStopA	Board starts acquiring lines. Will stop when max is reached or Trigger A de-asserts.
CiTrigOneShotStartAStopALevel	Board starts acquiring lines. Will stop when trigger A de-asserts.

Table 18-7 De-assertion of Triggers for Aon/Axion/Cyton

TrigMode	Trigger A deasserts
CiTrigOneShotStartAStopA	Board Stops acquiring lines.
CiTrigOneShotStartAStopALevel	Board Stops acquiring lines.

18.6 CiConVTrigModeSetEx

Prototype BFRC CiConVTrigModeSetEx(Bd *Board*, BFU32 *TrigMode*, BFU32 *TrigAssignments*, BFU32 *TrigAPolarity*, BFU32 *TrigBPolarity*, BFU32 *TrigSelect*)

Description Similar to CiContVTrigModeSet except adds the ability to select the trigger source.

Parameters

Board

Handle to board.

TrigMode

Trigger mode to put the board into, see CiContVTrigModeSet.

TrigAssignments

Controls which triggers go to which acquisition engine, see CiContVTrigModeSet.

TrigAPolarity

Polarity for trigger A, , see CiContVTrigModeSet.

TrigBPolarity

Polarity for trigger B, see CiContVTrigModeSet.

TrigSelect

Selects the trigger source.

For Road Runner/R3

CiTrigDiff - Differential trigger
CiTrigTTL - TLL trigger
CiTrigOpto - Opto-isolated trigger
CiIgnore - No change to trigger selection

For Karbon/Alta/Neon/R64

CiTrigDiff - Differential trigger
CiTrigTTL - TLL trigger
CiTrigOpto - Opto-isolated trigger
CiTrigFVAL - Trigger is the FVAL signal on the CL cable
CiTrigNTG - NTG is trigger source
CiIgnore - No change to trigger selection

For Aon/Axion/Cyton

CiTrigDiff - Differential trigger
CiTrigTTL - TLL trigger

CiTrigNTG, BFTrigTSCT0 - NTG/TS is trigger source
CiTrigVFG0TrigSel - Use same trigger as VFG 0
CiTrigButton - Trigger is button
CiTrigCXPTriggerIn - Trigger is CXP trigger (from camera)
CiTrigSWTrigger - Trigger is software trigger
CiTrigScanStep - Trigger comes from quad. encoder circuit in scan step mode
CiTrigNTGVFG0, BFTrigVFG0TSCT0 - Trigger is NTG/TS of VFG 0
CiIgnore - No change to trigger selection

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments

This function is identical to CiConVTrigModeSet except that it allows the trigger source to be programmed.

18.7 CiConVTrigModeGet

Prototype BFRC CiConVTrigModeGet(Bd Board, PBFU32 *TrigMode*, PBFU32 *TrigAssignments*, PBFU32 *TrigAPolarity*, PBFU32 *TrigBPolarity*)

Description Gets the current trigger mode and polarities for both acquisition engines.

Parameters

Board

Handle to board.

TrigMode

Returns the current trigger mode for acquisition engine J. See CiContVTrigModeSet for more information.

TrigAssignments

Note: This parameter is only used with the Raven. For all boards the value returned in this parameter will have not meaning.

Returns the current assignments of triggers to acquisition engines:

CiTrigJandKSameSource - both J and K are run by same trigger(s).
CiTrigJbyAandKbyB - J is run by TRIGGERA and K by TRIGGERB.
CiTrigJbyAandKFreerun – J is trigger and K free runs.

TrigAPolarity

Returns the current polarity for trigger A:

CiTrigAssertedHigh - trigger A is asserted on rising edge.
CiTrigAssertedLow - trigger A is asserted on falling edge.

TrigBPolarity

Note: This parameter is only used with the Raven. For all boards the value returned in this parameter will have not meaning.

Returns the current polarity for trigger B:

CiTrigAssertedHigh - trigger B is asserted on rising edge.
CiTrigAssertedLow - trigger B is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.

CISYS_ERROR_UNKNOWN_PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments

This function returns the current state of the trigger circuitry for both acquisition engines. See the function CiConVTrigModeSet for a complete description of the modes.

18.8 CiConVTrigModeGetEx

Prototype BFRC CiConVTrigModeGetEx(Bd Board, PBFU32 *TrigMode*, PBFU32 *TrigAssignments*, PBFU32 *TrigAPolarity*, PBFU32 *TrigBPolarity*, PBFU32 *TrigSelect*)

Description Similar to CIVonVTrigModeGet but also supports getting of the trigger source.

Parameters

Board

Handle to board.

TrigMode

Returns the current trigger mode for acquisition engine J. See CiContVTrigModeSet for more information.

TrigAssignments

Note: This parameter is only used with the Raven. For all boards the value returned in this parameter will have not meaning.

Returns the current assignments of triggers to acquisition engines:

CiTrigJandKSameSource - both J and K are run by same trigger(s).
CiTrigJbyAandKbyB - J is run by TRIGGERA and K by TRIGGERB.
CiTrigJbyAandKFreerun – J is trigger and K free runs.

TrigAPolarity

Returns the current polarity for trigger A:

CiTrigAssertedHigh - trigger A is asserted on rising edge.
CiTrigAssertedLow - trigger A is asserted on falling edge.

TrigBPolarity

Note: This parameter is only used with the Raven. For all boards the value returned in this parameter will have not meaning.

Returns the current polarity for trigger B:

CiTrigAssertedHigh - trigger B is asserted on rising edge.
CiTrigAssertedLow - trigger B is asserted on falling edge.

TrigSelect

Returns the trigger source.

For Road Runner/R3

CiTrigDiff - Differential trigger

CiTrigTTL - TLL trigger
CiTrigOpto - Opto-isolated trigger

For Karbon/Alta/Neon/R64

CiTrigDiff - Differential trigger
CiTrigTTL - TLL trigger
CiTrigOpto - Opto-isolated trigger
CiTrigFVAL - Trigger is the FVAL signal on the CL cable
CiTrigNTG - NTG is trigger source

For Aon/Axion/Cyton

CiTrigDiff - Differential trigger
CiTrigTTL - TLL trigger
CiTrigNTG, BFTrigTSCT0 - NTG/TS is trigger source
CiTrigVFG0TrigSel - Use same trigger as VFG 0
CiTrigButton - Trigger is button
CiTrigCXPTriggerIn - Trigger is CXP trigger (from camera)
CiTrigSWTrigger - Trigger is software trigger
CiTrigScanStep - Trigger comes from quad. encoder circuit in scan step mode
CiTrigNTGVFG0, BFTrigVFG0TSCT0 - Trigger is NTG/TS of VFG 0

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments

This function is similar to CiConVTrigModeGet expect also supported getting of the selected trigger source.

18.9 CiConHTrigModeSet

Prototype BFRC CiConHTrigModeSet(Bd *Board*, BFU32 *EncMode*, BFU32 *EncPolarity*, BFU32 *EncSelect*)

Description Sets the horizontal trigger mode and polarities for the acquisition engine.

Parameters

Board

Handle to board.

EncMode

The horizontal triggering mode:

CiEncFreeRun – no line trigger is used, board free runs.
CiEncOneShot – horizontal one shot mode, every line needs a line trigger.
CiEncOneShotSelfTriggered – self triggering one shot mode..

EncPolarity

Polarity for all line triggers:

CiEncAssertedHigh - line triggers are asserted on rising edge.
CiEncAssertedLow - line triggers are asserted on falling edge.

EncSelect

Type of encoder.

For all boards:

CiIgnore - Do not make change to the select encoder

For R2/R3:

CiEncA - Encoder A is active and B is disabled
CiEncAlt1 - Reserved.
CiEncAlt2 - Reserved.
CiEncAlt3 - Reserved.

For Karbon/R64/Neon:

CiEncTTL - Single ended TTL level encoder
CiEncDiff - Differential (LVDS) encoder
CiEncOpto - Optocoupled encoder

For Aon/Axion/Cyton:

CiEncTTL - Single ended TTL level encoder

CiEncDiff - Differential (LVDS) encoder
CiEncVFG0EncASel - Selected encoder on VFG0
CiEncNTG, BFEncTSCT0 - NTG/TS is encoder
CiEncButton - The boards buttong is the encoder
CiEncCXPTriggerIn - CXP trigger is the encoder (from camera)
CiEncSWEncoderA - Software encocder A is the encoder
CiEncNTGVFG0, BFEncVFG0TsCT0 - NTG/TS from VFG0

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	One of the parameters is not valid or the particular combination of parameters is not possible.
R64_BAD_CON_PARAM	One of the parameters is not valid or the particular combination of parameters is not possible.
GN2_BAD_CON_PARAM	One of the parameters is not valid or the particular combination of parameters is not possible.

Comments

18.10 CiConHTrigModeGet

Prototype BFRC CiConHTrigModeGet(Bd Board, PBFU32 *EndMode*, PBFU32 *EncPolarity*, PBFU32 *EncSelect*)

Description Gets the current horizontal encoder mode and polarity of the encoder.

Parameters

Board

Handle to board.

EncMode

Returns the current encoder mode:

CiEncFreeRun – no line trigger is used, board free runs.
CiEncOneShot – horizontal one shot mode, every line needs a line trigger.
CiEncOneShotSelfTriggered – self triggering one shot mode.

EncPolarity

Returns the current polarity for the encoder:

CiEncAssertedHigh - trigger A is asserted on rising edge.
CiEncAssertedLow - trigger A is asserted on falling edge.

EncSelect

Type of encoder.

For all boards:

CiIgnore - Do not make change to the select encoder

For R2/R3:

CiEncA - Encoder A is active and B is disabled
CiEncAlt1 - Reserved.
CiEncAlt2 - Reserved.
CiEncAlt3 - Reserved.

For Karbon-CL/R64/Neon:

CiEncTTL - Single ended TTL level encoder
CiEncDiff - Differential (LVDS) encoder
CiEncOpto - Optocoupled encoder

For Aon/Axion/Cyton:

CiEncTTL - Single ended TTL level encoder

CiEncDiff - Differential (LVDS) encoder
CiEncVFG0EncASel - Selected encoder on VFG0
CiEncNTG, BFEncTSCT0 - NTG/TS is encoder
CiEncButton - The boards buttong is the encoder
CiEncCXPTriggerIn - CXP trigger is the encoder (from camera)
CiEncSWEncoderA - Software encocder A is the encoder
CiEncNTGVFG0, BFEncVFG0TSCT0 - NTG/TS from VFG0

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments

18.11 CiConTriggerInputGet

Prototype BFRC CiConTriggerInputGet(Bd Board, PBFU32 *TrigSelect*, PBFU32 *TrigPolarity*)

Description Gets the current source for the trigger input and the trigger polarity.

Parameters *Board*

Handle to board.

TrigSelect

Returns the currently selected trigger input (not all options are available on all models):

BFTrigDiff
BFTrigTTL
BFTrigOpto
BFTrigFVAL
BFTrigVFG0TrigSel
BFTrigNTG
BFTrigButton
BFTrigCXPTriggerIn
BFTrigSWTrigger
BFTrigScanStep
BFTrigNTGVFG0
BFTrigLow
BFTrigHigh
BFTrigTSCT0
BFTrigVFG0TSCT0
BFTrigBoxInTTL0
BFTrigBoxInTTL1
BFTrigBoxInTTL2
BFTrigBoxInTTL3
BFTrigBoxInTTL4
BFTrigBoxInTTL5
BFTrigBoxInTTL6
BFTrigBoxInTTL7
BFTrigBoxInTTL8
BFTrigBoxInTTL9
BFTrigBoxInTTL10
BFTrigBoxInTTL11
BFTrigBoxInDiff0
BFTrigBoxInDiff1
BFTrigBoxInDiff2
BFTrigBoxInDiff3
BFTrigBoxInDiff4
BFTrigBoxInDiff5
BFTrigBoxInDiff6
BFTrigBoxInDiff7

BFTrigBoxInDiff8
BFTrigBoxInDiff9
BFTrigBoxInDiff10
BFTrigBoxInDiff11
BFTrigBoxInOpto0
BFTrigBoxInOpto1
BFTrigBoxInOpto2
BFTrigBoxInOpto3
BFTrigBoxInOpto4
BFTrigBoxInOpto5
BFTrigBoxInOpto6
BFTrigBoxInOpto7
BFTrigBoxIn24V0
BFTrigBoxIn24V1
BFTrigBoxIn24V2
BFTrigBoxIn24V3
BFTrigUnknown

TrigPolarity

Returns the current polarity for the trigger:

BFTrigAssertedHigh - trigger is asserted on rising edge.
BFTrigAssertedLow - trigger is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.

Comments

This function is similar to CiConVTrigModeGet(), however, CiConTriggerInputGet() does not return the current trigger (VTrig) mode.

18.12 CiConTriggerInputSet

Prototype BFRC CiConTriggerInputSet(Bd Board, BFU32 *TrigSelect*, BFU32 *TrigPolarity*)

Description Sets the current source for the trigger input and the trigger polarity.

Parameters *Board*

Handle to board.

TrigSelect

Sets the source of the trigger input (not all options are available on all models):

BFTrigDiff
BFTrigTTL
BFTrigOpto
BFTrigFVAL
BFTrigVFG0TrigSel
BFTrigNTG
BFTrigButton
BFTrigCXPTriggerIn
BFTrigSWTrigger
BFTrigScanStep
BFTrigNTGVFG0
BFTrigLow
BFTrigHigh
BFTrigTSCT0
BFTrigVFG0TSCT0
BFTrigBoxInTTL0
BFTrigBoxInTTL1
BFTrigBoxInTTL2
BFTrigBoxInTTL3
BFTrigBoxInTTL4
BFTrigBoxInTTL5
BFTrigBoxInTTL6
BFTrigBoxInTTL7
BFTrigBoxInTTL8
BFTrigBoxInTTL9
BFTrigBoxInTTL10
BFTrigBoxInTTL11
BFTrigBoxInDiff0
BFTrigBoxInDiff1
BFTrigBoxInDiff2
BFTrigBoxInDiff3
BFTrigBoxInDiff4
BFTrigBoxInDiff5
BFTrigBoxInDiff6
BFTrigBoxInDiff7

BFTrigBoxInDiff8
BFTrigBoxInDiff9
BFTrigBoxInDiff10
BFTrigBoxInDiff11
BFTrigBoxInOpto0
BFTrigBoxInOpto1
BFTrigBoxInOpto2
BFTrigBoxInOpto3
BFTrigBoxInOpto4
BFTrigBoxInOpto5
BFTrigBoxInOpto6
BFTrigBoxInOpto7
BFTrigBoxIn24V0
BFTrigBoxIn24V1
BFTrigBoxIn24V2
BFTrigBoxIn24V3
BFTrigUnknown

TrigPolarity

Sets the trigger polarity:

BFTrigAssertedHigh - trigger is asserted on rising edge.
BFTrigAssertedLow - trigger is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
GN2_BAD_CON_PARAM	One of the input parameters was incorrect
R64_BAD_CON_PARAM	One of the input parameters was incorrect

Comments

This function is similar to CiConVTrigModeSet(), however, CiConTriggerInputSet() does change the current trigger (VTrig) mode.

18.13 CiConEncoderInputGet

Prototype BFRC CiConEncoderInputGet(Bd Board, PBU32 *Encoder*, PBFU32 *EncSelect*, PBFU32 *EncPolarity*)

Description Gets the current source for the encoder input and the encoder polarity.

Parameters

Board

Handle to board.

Encoder

Which encoder to get the input source for:

BFTypeEncA
BFTypeEncB

EncSelect

Returns the currently selected trigger input (not all options are available on all models):

BFEncDiff
BFEncTTL
BFEncOpto
BFEncLow
BFEncHigh
BFEncAlt
BFEncVFG0EncASel
BFEncVFG0EncBSel
BFEncNTG
BFEncButton
BFEncCXPEncgerIn
BFEncSWEncoderA
BFEncSWEncoderB
BFEncScanStep
BFEncNTGVFG0
BFEncTSCT0
BFEncVFG0TSCT0
BFEncBoxInTTL0
BFEncBoxInTTL1
BFEncBoxInTTL2
BFEncBoxInTTL3
BFEncBoxInTTL4
BFEncBoxInTTL5
BFEncBoxInTTL6
BFEncBoxInTTL7
BFEncBoxInTTL8
BFEncBoxInTTL9

BFEncBoxInTTL10
BFEncBoxInTTL11
BFEncBoxInDiff0
BFEncBoxInDiff1
BFEncBoxInDiff2
BFEncBoxInDiff3
BFEncBoxInDiff4
BFEncBoxInDiff5
BFEncBoxInDiff6
BFEncBoxInDiff7
BFEncBoxInDiff8
BFEncBoxInDiff9
BFEncBoxInDiff10
BFEncBoxInDiff11
BFEncBoxInOpto0
BFEncBoxInOpto1
BFEncBoxInOpto2
BFEncBoxInOpto3
BFEncBoxInOpto4
BFEncBoxInOpto5
BFEncBoxInOpto6
BFEncBoxInOpto7
BFEncBoxIn24V0
BFEncBoxIn24V1
BFEncBoxIn24V2
BFEncBoxIn24V3
BFEncUnknown

EncPolarity

Returns the current polarity for the trigger:

CiEncAssertedHigh - Encoder is asserted on rising edge.
CiEncAssertedLow - Encoder is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.

Comments

This function is similar to CiConHTrigModeGet(), however, CiConEncoderInputGet() does not return the current encoder (HTrig) mode.

18.14 CiConEncoderInputSet

Prototype BFRC CiConEncoderInputSet(Bd Board, PBU32 *Encoder*, PBFU32 *EncSelect*, PBFU32 *EncPolarity*)

Description Sets the current source for the encoder input and the encoder polarity.

Parameters

Board

Handle to board.

Encoder

Which encoder to get the input source for:

BFTypeEncA
BFTypeEncB

EncSelect

The source to set for the encoder input (not all options are available on all models):

BFEncDiff
BFEncTTL
BFEncOpto
BFEncLow
BFEncHigh
BFEncAlt
BFEncVFG0EncASel
BFEncVFG0EncBSel
BFEncNTG
BFEncButton
BFEncCXPEncgerIn
BFEncSWEncoderA
BFEncSWEncoderB
BFEncScanStep
BFEncNTGVFG0
BFEncTSCT0
BFEncVFG0TSCT0
BFEncBoxInTTL0
BFEncBoxInTTL1
BFEncBoxInTTL2
BFEncBoxInTTL3
BFEncBoxInTTL4
BFEncBoxInTTL5
BFEncBoxInTTL6
BFEncBoxInTTL7
BFEncBoxInTTL8
BFEncBoxInTTL9
BFEncBoxInTTL10

BFEncBoxInTTL11
BFEncBoxInDiff0
BFEncBoxInDiff1
BFEncBoxInDiff2
BFEncBoxInDiff3
BFEncBoxInDiff4
BFEncBoxInDiff5
BFEncBoxInDiff6
BFEncBoxInDiff7
BFEncBoxInDiff8
BFEncBoxInDiff9
BFEncBoxInDiff10
BFEncBoxInDiff11
BFEncBoxInOpto0
BFEncBoxInOpto1
BFEncBoxInOpto2
BFEncBoxInOpto3
BFEncBoxInOpto4
BFEncBoxInOpto5
BFEncBoxInOpto6
BFEncBoxInOpto7
BFEncBoxIn24V0
BFEncBoxIn24V1
BFEncBoxIn24V2
BFEncBoxIn24V3
BFEncUnknown

EncPolarity

The encoder polarity to use:

CiEncAssertedHigh - Encoder is asserted on rising edge.
CiEncAssertedLow - Encoder is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments

This function is similar to CiConHTrigModeSet(), however, CiConEncoderInputSet() does not take the current encoder (HTrig) mode.

18.15 CiConTriggerInputSet

Prototype BFRC CiConTriggerInputSet(Bd Board, BFU32 *TrigSelect*, BFU32 *TrigPolarity*)

Description Sets the current source for the trigger input and the trigger polarity.

Parameters *Board*

Handle to board.

TrigSelect

Sets the source of the trigger input (not all options are available on all models):

BFTrigDiff
BFTrigTTL
BFTrigOpto
BFTrigFVAL
BFTrigVFG0TrigSel
BFTrigNTG
BFTrigButton
BFTrigCXPTriggerIn
BFTrigSWTrigger
BFTrigScanStep
BFTrigNTGVFG0
BFTrigLow
BFTrigHigh
BFTrigTSCT0
BFTrigVFG0TSCT0
BFTrigBoxInTTL0
BFTrigBoxInTTL1
BFTrigBoxInTTL2
BFTrigBoxInTTL3
BFTrigBoxInTTL4
BFTrigBoxInTTL5
BFTrigBoxInTTL6
BFTrigBoxInTTL7
BFTrigBoxInTTL8
BFTrigBoxInTTL9
BFTrigBoxInTTL10
BFTrigBoxInTTL11
BFTrigBoxInDiff0
BFTrigBoxInDiff1
BFTrigBoxInDiff2
BFTrigBoxInDiff3
BFTrigBoxInDiff4
BFTrigBoxInDiff5
BFTrigBoxInDiff6
BFTrigBoxInDiff7

BFTrigBoxInDiff8
BFTrigBoxInDiff9
BFTrigBoxInDiff10
BFTrigBoxInDiff11
BFTrigBoxInOpto0
BFTrigBoxInOpto1
BFTrigBoxInOpto2
BFTrigBoxInOpto3
BFTrigBoxInOpto4
BFTrigBoxInOpto5
BFTrigBoxInOpto6
BFTrigBoxInOpto7
BFTrigBoxIn24V0
BFTrigBoxIn24V1
BFTrigBoxIn24V2
BFTrigBoxIn24V3
BFTrigUnknown

TrigPolarity

Sets the trigger polarity:

CiEncAssertedHigh - trigger A is asserted on rising edge.
CiEncAssertedLow - trigger A is asserted on falling edge.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
GN2_BAD_CON_PARAM	One of the input parameters was incorrect
R64_BAD_CON_PARAM	One of the input parameters was incorrect

Comments

This function is similar to CiConVTrigModeGet(), however, CiConTriggerInputGet() does not return the current trigger (VTrig) mode.

18.16 CiConSwTrig

Prototype BFRC CiConSwTrig(Bd *Board*, BFU32 *Triggers*, BFU32 *AssertType*)

Description Performs a software trigger.

Parameters

Board

Handle to board.

Triggers

Trigger to assert:

CiTrigA – software trigger A.
CiTrigB – software trigger B.

AssertType

Type of assertion:

CiTrigAssert - assert the trigger.
CiTrigDeassert - de-assert the trigger.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments The BitFlow boards have both software and hardware triggers. The hardware triggers are driven by external signals and are enabled by calling the function CiConExTrigConnect. The software triggers are always enabled. In order for a software trigger to actually cause anything to happen, the board must be in a triggering mode. See the function CiConVTrigModeSet.

The software trigger does not normally need to be deasserted. Normally, you need only assert the software trigger to cause a frame to be acquired. However, in the Start/Stop modes, the acquisition is initiated by the trigger being asserted and terminated when the trigger is deasserted.

Note: The software triggers are not affected by the state of the trigger polarity settings.

18.17 CiConSwTrigStat

Prototype BFRC CiConSwTrigStat(Bd *Board*, BFU32 *CiTrig*, PBFU32 *Status*)

Description Returns the status of the software trigger.

Parameters

Board

Handle to board.

CiTrig

The trigger to inquire about, can be one of the following:

CiTrigA - inquire about trigger A.
CiTrigB - inquire about trigger B.

Status

The status of the trigger can be one of the following:

CiTrigHigh - the software trigger was high.
CiTrigLow - the software trigger was low.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The R64 only uses CiTrigA for CiTrig.
R2_BAD_CON_PARAM	Invalid CiTrig was passed to the function.
R64_BAD_CON_PARAM	Invalid CiTrig was passed to the function.
GN2_BAD_CON_PARAM	Invalid CiTrig was passed to the function.

Comments This function returns the status of the software trigger at the moment that the function is called

18.18 CiConExTrigConnect

Prototype BFRC CiConExTrigConnect(Bd *Board*, BFU32 *CiTrig*, BFU32 *Mode*)

Description Connects or disconnect the external hardware trigger to the acquisition circuitry.

Parameters

Board

Handle to board.

CiTrig

Trigger to change:

CiTrigA – TRIGGERA.
CiTrigB – TRIGGERB.

Mode

Change to make:

CiExTrigConnect - connect the trigger.
CiExTrigDisconnect - disconnect the trigger.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
R64_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.
GN2_BAD_CON_PARAM	Either one of the parameters is not valid or the particular combination or parameters is not possible.

Comments This function connects the external hardware trigger to the acquisition circuitry. This function lets you turn on or off the effect of external triggers without altering another other settings on the board, as well as whatever machinery is driving the trigger signal.

18.19 CiConExTrigStatus

Prototype BFRC CiConExTrigStatus(Bd *Board*, BFU32 *CiTrig*, PBFU32 *Mode*)

Description Returns the status of the hardware trigger to the acquisition circuitry.

Parameters

Board

Handle to board.

CiTrig

Trigger to inquire:

CiTrigA – TRIGGERA.
CiTrigB – TRIGGERB.

Mode

Returns the status of connection:

CiExTrigConnect - connect the trigger.
CiExTrigDisconnect - disconnect the trigger.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.

Comments This function returns the status of the connection between the external hardware trigger and the acquisition circuitry.

18.20 CiConHWTrigStat

Prototype BFRC CiConHWTrigStat(Bd *Board*, BFU32 *CiTrig,* PBFU32 *Status*)

Description Returns the status of the hardware trigger.

Parameters

Board

Handle to board.

CiTrig

The trigger to inquire about, can be one of the following:

CiTrigA - inquire about trigger A.
CiTrigB - inquire about trigger B.

Status

The status of the trigger can be one of the following:

CiTrigHigh - the software trigger was high.
CiTrigLow - the software trigger was low.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The R64 only uses CiTrigA for CiTrig.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
R2_BAD_CON_PARAM	Invalid *CiTrig* was passed to the function.
R64_BAD_CON_PARAM	Couldn't determine the trigger type being used.

Comments This function returns the status of the hardware trigger at the moment that the function is called

18.21 CiConDMACommand

Prototype BFRC CiConDMACommand(Bd *Board*, BFU32 *Command*, BFU32 *DMAChannel*)

Description Issues a DMA command to the board.

Parameters

Board

Handle to board.

Command

DMA command to issue:

CiConDMAGo - start the DMA engine.
CiConDMAAbort - immediately abort the current DMA operation.

DMAChannel

Note: This parameter is only used with the Raven. For all boards this parameter should be set to zero.

Which DMA channel to send command to:

DMAChannel0 - Use DMA Channel 0.
DMAChannel1 - Use DMA Channel 1.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_CON_PARAM	Unknown command.
R64_BAD_CON_PARAM	Unknown command.
GN2_BAD_CON_PARAM	Unknown command.

Comments This function provides low-level access to the DMA engines. Normally there is no need to call this function. The DMA engines are controlled by the acquisition engines, and can therefore be controlled by the acquisition commands (see CiAqCommand). However, in certain circumstances, direct control of the DMA engine may be needed, therefore this function is provided.

18.22 CiShutDown

Prototype BFRC CiShutDown(Bd *Board*, BFU32 *AqEngine*)

Description Aborts all DMA activity and acquisition on the board.

Parameters *Board*

Handle to board.

AqEngine

Note: AqEngK is available on the Raven. For all boards this parameter should be set to AqEngJ.

The acquisition engine to build the QTab for:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	One of the parameters passed to this function is not correct for this board type.
R2_BAD_DMA0_STOP	Time-out waiting for DMA engine 0 to abort.
R2_BAD_DMA1_STOP	Time-out waiting for DMA engine 1 to abort.
R2_BAD_AQ_STOP	Time-out waiting for acquisition to abort.
R2_BAD_FIFO_RESET	Could not reset the FIFOs.
GN2_AQEND_TIMEOUT	Acquisition did not end

Comments This functions aborts all activity on the board. DMA is aborted. Acquisition is aborted. The FIFOs are reset. The board stops what it is currently doing and gets it ready for more acquisition. Normally this function does not need to be called.

Because the Raven is the only board with two acquisition engines, the Raven is the only board where the *AqEngine* parameter can equal AqEngK. For all other boards ignore this parameter must equal AqEngJ.

18.23 CiConAqMode

Prototype BFRC CiConAqMode(Bd *Board*, BFU32 *DestType*)

Description For a given destination type, this function sets the board's acquisition MUX registers, based on the current camera type.

Parameters

Board

Handle to board.

DestType

Type of acquisition to prepare for:

CiDMABitmap - the destination buffer to be used for display.
CiDMADataMem - the destination buffer needs to contain raw data.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
R2_BAD_CON_PARAM	Unknown *DestType* parameter.
R64_BAD_CON_PARAM	Unknown *DestType* parameter.

Comments This function sets up the board's front end acquisition paths for acquiring to a display buffer or a raw data buffer. A display buffer is one that will be used for display on a monitor, and is 8 bits deep. A raw data buffer is one that the data has the same bit depth as the camera.

This function has no effect for 8-bit cameras.

This function is normally called automatically by CiAqSetup, and does not need to be called explicitly by an application. An unpredictable result will occur if this function is called while the board is acquiring.

Currently, this function works with the Road Runner/R3 and the R64 family.

18.24 CiConFIFOReset

Prototype BFRC CiConFIFOReset(Bd *Board*)

Description Resets the FIFO and FIFO freeze register on the board.

Parameters *Board*

Handle to board.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function only works on the R2/R3 family.

18.25 CiConCtabReset

Prototype BFRC CiConCtabReset(Bd *Board*)

Description Resets the control tables on the board.

Parameters *Board*

Handle to board.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.

Comments This function sets the CTab counters, HCOUNT and VCOUNT to zero. This function works on the R2/R3 and R64 families.

18.26 CiConGetFrameCount

Prototype BFRC CiConGetFrameCount(Bd *Board*, PBFU32 *FrameCount*, BFU32 *AqEngine*)

Description Returns the 3-bit frame count from the board.

Parameters

Board

Handle to board.

FrameCount

The board's current frame count.

AqEngine

The acquisition engine to get the frame count for:

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	Unknown *AqEngine* parameter.

Comments On the R2/R3/Karbon/Neon the frame count is a 3 bit counter. On the Aon/Axion/Cyton, the frame count is a 24 bit counter.

Because the Raven is the only board with two acquisition engines, the Raven is the only board where the *AqEngine* parameter can equal AqEngK. For all other boards ignore this parameter must equal AqEngJ.

18.27 CiConIntModeSet

Prototype BFRC CiConIntModeSet(Bd *Board*, BFU32 *Mode*)

Description Sets the interrupt mode for the board.

Parameters

Board

Handle to board.

Mode

Type of interrupt mode to set the board for:

BFIntModeDefault - Interrupts will happen all the time.
BFIntModeEOFAq - EOF interrupts will only happen when acquisition begins.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	This board is not supported by this function.

Comments The R2/R3 and Gen2 families always run in the BFIntModeDefault mode where the interrupts are happening all the time. Only the R64 has the ability to be set in the BFIntModeEOFAq mode.

18.28 CiConIntModeGet

Prototype BFRC CiConIntModeGet(Bd *Board*, PBFU32 *Mode*)

Description Gets the interrupt mode for the board.

Parameters

Board

Handle to board.

Mode

Type of interrupt mode the board is set for:

BFIntModeDefault - Interrupts will happen all the time.
BFIntModeEOFAq - EOF interrupts will only happen when acquisition begins.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.

Comments

18.29 CiConExposureControlSet

Prototype BFRC CiConExposureControlSet(Bd *Board*, BFDOUBLE *ExposurePeriod*, BFDOUBLE *LineFramePeriod*, BFU32 *TriggerMode*, BFBOOL *AssertedHigh*, BFU32 *OutputSignal*)

Description Programs the board's timing generator, used to create waveforms to control the line/frame rate and exposure time of cameras.

Parameters

Board

Handle to board.

ExposurePeriod

The desired exposure period in milliseconds

Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)

LineFramePeriod

The desire line/frame rate period in milliseconds.

Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)

TriggerMode

The triggering mode for the timing generator. Must be one of the following:

BFNTGModeFreeRun - Timing generator is free running.
BFNTGModeOneShotTrigger - Timing generator is in one-shot mode, triggered by the board's trigger input.
BFNTGModeOneShotEncoder - Timing generator is in one-shot mode, triggered by the board's encoder input.

AssertedHigh

The level of the timing generator's output waveform. Must be:

TRUE - Waveform is asserted high.
FALSE - Waveform is asserted low.

OutputSignal

The outputs that the waveform will be output on. Can be one or more of the following ORed together (signal will be output on all pins selected by this parameter):

For the Karbon/Neon/Alta:

BFNTGOutputCC1 - Output on the CC1 signal on CL connector.
BFNTGOutputCC2 - Output on the CC2 signal on CL connector.
BFNTGOutputCC3 - Output on the CC3 signal on CL connector.
BFNTGOutputCC4 - Output on the CC4 signal on CL connector.
BFNTGOutputGP0 - Output on GPOUT0 on the I/O connector.
BFNTGOutputGP1 - Output on GPOUT1 on the I/O connector.
BFNTGOutputGP2 - Output on GPOUT2 on the I/O connector.
BFNTGOutputGP3 - Output on GPOUT3 on the I/O connector.
BFNTGInputTrig - Output goes to Trigger input.
BFNTGInputEncA - Output goes to Encoder A input.

For the Aon/Axion/Cyton

BFNTGOutputCC1 - Output on the CC1 signal.
BFNTGOutputCC2 - Output on the CC2 signal.
BFNTGOutputCC3 - Output on the CC3 signal.
BFNTGOutputCC4 - Output on the CC4 signal.
BFNTGInputTrig - Output goes to Trigger input.
BFNTGInputEncA - Output goes to Encoder A input.
BFNTGInputEncB - Output goes to Encoder B input.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The current board family does not support the NTG.
R64_NTG_NOT_SUPPORTED	The installed frame grabber does not support the NTG or the current firmware does not currently support the NTG (contact BitFlow for more information).
R64_NTG_EXP_OUT_OF_ RANGE	The requested values are out of range of the timing generator
R64_NTG_EXP_GT_LF	The requested exposure time is longer than the requested line/frame period.
R64_NTG_UNKNOWN_MODE	The requested mode triggering mode is not known.
GN2_TS_EXP_OUT_OF_RANGE	The requested values are out of range of the timing generator
GN2_TS_EXP_GT_LF	The requested exposure time is longer than the requested line/frame period.
GN2_TS_UNKNOWN_MODE	The requested mode triggering mode is not known.

GN2_TS_PROG_ERROR	The Timing Sequencer could not be programmed for the given parameters

Comments

This function is used to program the New Timing Generator (NTG) available on the Karbon/Neon/Alta and the Timing Sequencer on the Aon/Axion/Cyton. These timing generator are used to control the line/frame rate and exposure time of attached cameras.

The Exposure time is controlled by the *ExposurePeriod* parameter. This parameter takes a floating point value in units of milliseconds. The line/frame rate is controlled by the *LineFrameRate* parameter. This parameter is also floating point and the units are in milliseconds. Note that although this parameter controls the line/frame rate, it is not in units of Hertz, which it would be if this parameter was the line/frame frequency. Instead this parameters controls the line/frame period, units of time. Refer to the hardware manual of your frame grabber to see the range for these parameters.

The triggering of the timing generators are independent of the triggering configuration of the rest of the frame grabber. They are fully independent of all other components of the frame grabber, and runs completely on its own timing. These timing generator can be triggered either by the currently selected trigger input or the currently selected encoder input.

The output waveform can be routed to one or more outputs. The parameter *OutputSignal* controls which outputs get the waveform. This parameter can take one or more of the defined outputs ORed together. The waveform will appear on all outputs simultaneously selected by this parameter.

The current status of the timing generator can be retrieved using the CiConExposureControlGet function.

Please refer to the hardware manual of your board for more detailed information on how this timing generator works.

18.30 CiConExposureControlGet

Prototype BFRC CiConExposureControlSet(Bd *Board*, PBFDOUBLE *pExposurePeriod*, PBFDOUBLE *pLineFramePeriod*, PBFU32 *pTriggerMode*, PBFBOOL *pAssertedHigh*, PBFU32 *pOutputSignal*)

Description Retrieve the current parameters of the timing generator.

Parameters

Board

Handle to board.

pExposurePeriod

Pointer to a double, returns the current exposure period in milliseconds

Note: This parameter is floating point and can be a in non-whole number values (e.g. 10.523)

pLineFramePeriod

Pointer to a double, returns the current line/frame rate period in milliseconds.

Note: This parameter is floating point and can be in non-whole number values (e.g. 10.523)

pTriggerMode

Pointer to a BFU32, returns the current triggering mode for the timing generator. Will be one of the following:

BFNTGModeFreeRun - Timing generator is free running.
BFNTGModeOneShotTrigger - Timing generator is in one-shot mode, triggered by the board's trigger input.
BFNTGModeOneShotEncoder - Timing generator is in one-shot mode, triggered by the board's encoder input.

pAssertedHigh

Pointer to a BFU32, returns the current the current level of the timing generator's output waveform. Will be:

TRUE - Waveform is asserted high.
FALSE - Waveform is asserted low.

pOutputSignal

Pointer to a BFU32, returns the current outputs that the waveform is being output on. Will be one or more of the following ORed together:

For the Karbon-CL/Neon/Alta:

CiNTGOuputCC1 - Output on the CC1 signal on CL connector.
CiNTGOuputCC2 - Output on the CC2 signal on CL connector.
CiNTGOuputCC3 - Output on the CC3 signal on CL connector.
CiNTGOuputCC4 - Output on the CC4 signal on CL connector.
CiNTGOutputGP0 - Output on GPOUT0 on the I/O connector.
CiNTGOutputGP1 - Output on GPOUT1 on the I/O connector.
CiNTGOutputGP2 - Output on GPOUT2 on the I/O connector.
CiNTGOutputGP3 - Output on GPOUT3 on the I/O connector.
CiNTGInputTrig - Output goes to Trigger input.
CiNTGInputEncA - Output goes to Encoder A input.

For the Aon/Axion/Cyton:

CiNTGOuputCC1 - Output on the CC1.
CiNTGOuputCC2 - Output on the CC2.
CiNTGOuputCC3 - Output on the CC3.
CiNTGOuputCC4 - Output on the CC4.
CiNTGInputTrig - Output goes to Trigger input.
CiNTGInputEncA - Output goes to Encoder A input.
CiNTGInputEncB - Output goes to Encoder B input.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The current board family does not support the NTG.
R64_NTG_NOT_SUPPORTED	The installed frame grabber does not support the NTG or the current firmware does not currently support the NTG (contact BitFlow for more information).

Comments

This function is retrieves the current status of the New Timing Generator (NTG) on the Karbon/Neon/Alta or the Timing Sequencer (TS) on the Aon/Axion/Cyton. The timing generator is used to control the line/frame rate and exposure time of attached cameras.

The timing generator can be programmed using the CiConExposureControlSet function.

18.31 CiEncoderDividerSet

Prototype

BFRC CiEncoderDividerSet(Bd *Board*,BFDOUBLE *ScaleFactor*, BFBOOL *ForceDC*, BFBOOL *OpenLoop*, BFU32 *ClockSelect*);

Description

Programs the encoder divider circuit. This circuit is used to modify the incoming encoder frequency so that the camera can be run at a different line rate.

Parameters

Board

Handle to board.

ScaleFactor

The factor to scale incoming encoder frequency. The incoming encoder frequency is multiplied by this factor to derive the internal encoder frequency. This is a floating point number. For example, if this parameter was 2.5 and the incoming encoder frequency was 1000 Hz, the resulting internal encoder frequency would be 2500 Hz. Range is from 0.00098 to 64.0.

ForceDC

When set to TRUE, the internal encoder frequency is forced to zero when the incoming frequence falls below a certain low level.

OpenLoop

When set to TRUE, the internal encoder runs exactly at the given frequency based on the ScaleFactor, the internal and the external encoder have no phase relationship. When set to FALSE, the internal encoder will be exactly in phase with the external encoder. This causes a small FM component to the internal encoder, but the timing will be exact.

ClockSelect

Reserved for future functionality. Must be set to 0.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
R64_ENCDIV_NOT_SUPPORTED	The board does not support the encoder divider function.
R64_ENCDIV_OUT_OF_RANGE	The *ScaleFactor* parameter is out of range.
R64_ENCDIV_UNKNOWN_CLK	The parameter *ClockSelect* is a value tha is not supported.
GN2_ENCDIV_OUT_OF_RANGE	The *ScaleFactor* parameter is out of range.

Comments

This function is used to programmatically control the encoder divider circuit. This circuit is used to modify the incoming encoder frequency. This frequency is multiplied by the value in the parameter *ScaleFactor* to create the internal encoder frequency. The internal encoder frequency is used to drive the horizontal timing of the board as well as the camera's line rate (assuming a line scan camera is being used).

For more details on how the encoder divider circuit works, see the hardware reference manual for board that you are using.

18.32 CiEncoderDividerGet

Prototype BFRC CiEncoderDividerGet(Bd *Board*, PBFDOUBLE *pScaleFactor*, PBFBOOL *pForceDC*, PBFBOOL p*OpenLoop*, PBFU32 *pClockSelect*);

Description Gets the encoder divider circuit's parameters.

Parameters

Board

Handle to board.

pScaleFactor

Returns a pointer to the the current factor used to scale the incoming encoder frequency. The incoming encoder frequency is multiplied by this factor to derive the internal encoder frequency. This is a floating point number. For example, if this parameter was 2.5 and the incoming encoder frequency was 1000 Hz, the resulting internal encoder frequency would be 2500 Hz. Range is from 0.00098 to 64.0.

pForceDC

Returns a pointer to a boolean. When set to TRUE, the internal encoder frequency is forced to zero when the incoming frequence falls below a certain low level.

pOpenLoop

Returns a pointer to a boolean. When set to TRUE, the internal encoder runs exactly at the given frequency based on the ScaleFactor, the internal and the external encoder have no phase relationship. When set to FALSE, the internal encoder will be exactly inphase with the external encoder. This causes a small FM component to the internal encoder, but the timing will be exact.

pClockSelect

Reserved for future functionality. Must be set to 0.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
R64_ENCDIV_NOT_SUPPORTED	The board does not support the encoder divider function.

Comments This function is used to retrieve the parameters of the encoder divider circuit. For more information see CiEncoderDividerSet.

For more details on how the encoder divider circuit works, see the hardware reference manual for board that you are using.

18.33 CiConNumFramesSet

Prototype BFRC CiConNumFramesSet(Bd *Board*, BFU32 *NumFrames*)

Description Sets the number of frames to acquire for the next acquisition operation. Currently only supported on Gen2 board.

Parameters *Board*

Handle to board.

NumFrames

The number of frames that should be acquired when the next acquisition command is issued. This can be any of the following

1 - One frame will be acquired, this is equivalent to a snap operation.
2 to 0xffffff - The board will grab this number of frames and then freeze automatically.
INFINITE (0xffffffff) - The board will acquire continuously until a freeze or abort command is issued.

Returns

CI_OK	If successful.
GN2_BAD_CON_PARAM	The value of *NumFrames* is not supported.

Comments This function is used to control how many frames are acquired. This function is only needed when programming the board's acquisition engine directly. This is not needed when using higher level APIs (such as the Bi functions).

Note: This function currently only works on Gen2 boards.

18.34 CiConIsCameraReady

Prototype BFRC CiConIsCameraReady(Bd *Board*, PBFBOOL *pReady*)

Description Returns true if a camera is connected and is power up and running.

Parameters

Board

Handle to board.

pReady

Parameter is set to true if camera is ready, false if the camera is not connected or not ready.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.

Comments This function can be use to detect of there is a camera connected to the board and it is powered up and providing some kind of clock.

18.35 CiConCamLineWidthSet

Prototype BFRC CiConCamLineWidthSet(Bd *Board*, BFU32 *CamLineWidth*)

Description Overrides the pixels per line parameter with a user value.

Parameters

Board

Handle to board.

CamLineWidth

New value to set the acquisition line width (in units of pixels)/

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	This board is not supported by this function.

Comments Some cameras require the Pixel Router to be programmed different that the Acquisition Engine. For example, 10-tap cameras output must be on boundaries of 10 pixels while the DMA engine requires boundaries of 16 bytes. Normally is taken care of via the use of the parameter cam_line_width BFML file. However, in situations where the ROi is being manipulated dynamically from software, this function can be used to update the actual camera line width (in pixels), while the ROI functions can be used to update the actual acquisition image width.

Note: This function is only needed on Gen 2 Camera Link models.

Ci Quad Table Functions

Chapter 19

19.1 Introduction

For almost all BitFlow applications there will be no need to call any of the functions in this chapter. These are considered mid-level functions and are generally only called indirectly by other, high level, functions. These functions are listed here in case some specialized programming is required.

Quad Tables (or QTABS) are simple scatter gather DMA tables. A scatter gather DMA table is a list of instructions that tell the board how to DMA images to host memory. The name quad comes from the fact that each DMA instruction consists of four, 32-bit words: DMA source, DMA destination, DMA size, and a pointer to the next DMA instruction.

There are two types of QTABs: relative and physical. These differ only by the kind of memory the DMA destination describes. Relative QTABs describe relative or virtual memory address, which is typically all that your application sees.

Physical QTABs describe actual physical locations of memory. A relative QTab is created based on the current camera and the memory pointer that is handed to the create function. The physical QTab is built from this at the kernel level by locking down the virtual memory and calculating the physical addresses of this memory.

19.2 CiRelQTabCreate

Prototype BFRC CiRelQTabCreate(Bd *Board*, PBFCNF *pCam*, PBFVOID *pDest*, BFU32 *BufferSize*, BFS32 *Stride*, PQTABHEAD *pRelQTabHead*, BFU32 *DestType*, BFU32 *LutBank*, BFU32 *LutType*, BFU32 *Options,* BFU32 *AqEngine*)

Description Builds a relative QTab, used for acquisition from a given camera type to a host memory buffer. The relative QTab can then be converted to a physical QTab, which can be written to the board.

Parameters

Board

Handle to board.

pCam

Camera object of the type to build the QTab for.

pDest

A void pointer to the destination buffer.

BufferSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in pixels, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

pRelQTabHead

Pointer to an allocated QTABHEAD structure.

DestType

Note: This parameter is ignored for the Raven.

CiDMADataMem – host memory.
CiDMABitmap – Display memory.

LutBank

The LUT bank to pass the image through:

Note: This parameter is ignored for the R64.

CiLutBank0 - LUT bank 0
CiLutBank1 - LUT bank 1
CiLutBank2 - LUT bank 2
CiLutBank3 - LUT bank 3
CiLutBypass - bypass LUTs

LutType

The mode of the LUT to use. The Raven will always use 8 bit LUTs, for the RoadRunner the following LUT types are available:

Note: This parameter is ignored for the R64.

CiLut8Bit - LUTs are programmed as 8 bits.
CiLut12Bit - LUTs are programmed as 12 bits.
CiLut16Bit - LUTs are programmed as 16 bits. (only on boards with 16-bit LUTs)

Options

Extra option for the last quad. Can be one or more of:

CiDMAOptInt - set interrupt bit in last quad.
CiDMAOptEOC - set EOC bit in last quad.

AqEngine

The acquisition engine to build the QTab for:

Note: This parameter is ignored for the Road Runner/R3 and the R64.

AqEngJ - set up the J engine.
AqEngK - set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETER	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R2_BAD_CNF	Error extracting information from the camera object.

R2_BAD_MODEL	The camera configuration contains a QTab model or format that is not understood by this version of the SDK. Or, the camera configuration is such that the relative QTab cannot be built.
R2_BAD_CON_PARAM	The *LutBank, LutType* or *AqEngine* parameter is incorrect.
R2_BAD_ROI	There is a problem with the destination memory buffer size or location.
RV_BAD_CNF	Error extracting information from the camera object.
RV_BAD_MODEL	The camera configuration contains a QTab model or format that is not understood by this version of the SDK. Or, the camera configuration is such that the relative QTab cannot be built.
RV_BAD_CON_PARAM	The *LutBank* or *AqEngine* parameter is incorrect.
RV_BAD_ALLOC	Cannot allocate enough memory to build relative QTab.
R64_BAD_CNF	Error extracting information from the camera object.
R64_BAD_MODEL	The camera configuration contains a QTab model or format that is not understood by this version of the SDK. Or, the camera configuration is such that the relative QTab cannot be built.
R64_BAD_CON_PARAM	The *LutBank, LutType* or *AqEngine* parameter is incorrect.
BF_BAD_ALLOC	Cannot allocate enough memory to build relative QTab.
BF_BAD_ROI	There is a problem with the destination memory buffer size or location.
BF_BAD_CON_PARAM	The values to one of the parameters is incorrect.

Comments

This function builds a relative QTab for acquisition of a given camera type into a host memory buffer. The QTab is a table of scatter-gather DMA instructions that the board uses to continuously (and without host intervention) DMA camera data to the host memory. The relative QTab is the version of this table that is built with virtual addresses. These virtual addresses point to the destination buffer as addressed in the application's address space. The relative QTab must be passed to CiPhysQTabCreate to build a physical QTab. The physical QTab is the same as the relative QTab except

that it contains physical addresses that can be used by the board as actual DMA destinations. The physical QTab is stored in the kernel and can be quickly copied to the board.

This is a mid-level function and should not be called except for custom programming. The high-level function CiAqSetup will call this function for you.

Depending on the camera, this function may take a moderate amount of time to calculate the relative QTab. This function should only be called once, for a given camera and destination. The relative QTab can be used repeatedly to acquire from the same camera type into the same memory buffer.

This function allocates memory to hold the relative QTab in the users address space. Call CiRelQTabFree to release this and other resources allocated in this function.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter. When using a Raven, the QTab build will only work with the given acquisition engine. You cannot build a QTab for one engine and use it with another.

19.3 CiRelQTabFree

Prototype BFRC CiRelQTabFree(Bd *Board*, PQTABHEAD *pRelQTabHead*)

Description Frees resources allocated in CiRelQTabCreate.

Parameters *Board*

Handle to board.

pRelQTabHead

Pointer to QTABHEAD structure previously passed to CiRelQtabCreate.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.

Comments This function releases the memory used to hold the relative QTab and any other resources allocated in CiRelQTabCreate.

19.4 CiPhysQTabCreate

Prototype BFRC CiPhysQTabCreate(Bd *Board*, PQTABHEAD *pRelQTabHead,* BFU32 *AqEngine*)

Description Builds a physical QTab that backs a relative QTab that describes a destination buffer in host memory for a given camera type.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure that has been filled out in CiRelQTabCreate.

AqEngine

The acquisition engine to build the QTab for:

Note: The Raven is the only board that supports two acquisition engines. For the Road Runner/R3 and the R64, this parameter must be set to AqEngJ.

AqEngJ – set up the J engine.
AqEngK – set up the K engine.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
R2_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.
RV_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.

Comments This function takes a relative QTab created in CiRelQTabCreate and builds a physical QTab (storage is allocated in this function). The physical QTab contains actual physical addresses of the destination buffer in memory. The physical addresses can be used by the board as DMA destinations. This function also locks the destination buffer into memory (prevents the operating system from swapping the memory to disk). The memory must be locked in order for the DMA request to be satisfied.

When this function returns, the QTABHEAD structure contains the handle to the newly created physical QTab.

The resulting physical QTab is stored in the driver. It can be copied to the board using CiPhysQTabWrite. Multiple physical QTABs can be built and live in the driver at the same time, each with it's own QTABHEAD structure. The physical QTab can be released with a call to CiPhysQTabFree.

This is a mid-level function and should not be called except for custom programming. The high-level function CiAqSetup will call this function for you.

Because the Raven is the only board with two DMA engines, the Raven is the only board where the *AqEngine* parameter is relevant. All other boards ignore this parameter.

19.5 CiPhysQTabWrite

Prototype BFRC CiPhysQTabWrite(Bd *Board*, PQTABHEAD *pRelQTabHead*, BFU32 *Offset*)

Description Writes a physical QTab to a board.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure, containing a valid physical QTab.

Offset

The entry number to start writing the physical QTab. This can be any value between 0 and 32768. However, this value is usually the location of the first quad in a bank, depending on the QTab bank mode, this can change. The following defines are available for your use:

In two bank mode:

0 – start of first bank.
CIQTABBANKSTART1_2 – start of second bank.

In four bank mode:

0 – start of first bank
CIQTABBANKSTART1_4 – start of second bank.
CiQTABBANKSTART2_4 – start of third bank.
CiQTABBANKSTART3_4 – start of fourth bank.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The R64 dose not support this function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R2_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.
RV_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.

Comments

This function is only needed in the case that your application is using board QTABs. As of SDK 4.00 there is almost no reason to use board QTABs. Thus there is almost no reason to use this function.

This function takes an already created physical QTab and copies it into the board's DMA quad tables. These tables are used to tell the board's DMA engine where, and how many pixels to DMA to host.

This is a mid-level function and should not be called except for custom programming. The high-level function CiAqSetup will call this function for you.

19.6 CiPhysQTabFree

Prototype BFRC CiPhysQTabFree(Bd *Board*, PQTABHEAD *pRelQTabHead*)

Description Frees the memory used to hold the physical QTab in the driver memory.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure that contains a valid physical QTab.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_UNKNOWN_ PARAMETERS	The parameter to inquire about is not recognized. Check that the parameter is valid for the board being used.
R2_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.
RV_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.

Comments This function frees the driver level resources used to hold a physical QTab.

19.7 CiPhysQTabEngage

Prototype BFRC CiPhysQTabEngage (Bd *Board*, PQTABHEAD *pRelQTabHead*)

Description This sets the board up to use the given QTab for the next DMA operation. This function should be called for both host and board QTABs.

Parameters

Board

Handle to board.

pRelQTabHead

A pointer to a Relative QTab head structure. This should be the QTab for the host memory buffer that will acquired into when the next acquisition command occurs.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments This function engages the QTab *pRelQTabHead* so that the board will use this QTab for subsequent DMA operations. This is a mid level function which is not need it the high level functions (e.g. CiAqSetup) are being used to set up DMA.

This function is used when building QTABs using the CiRelQTabCreate functions. The normal order of function calls is as follows

```
CiRelQTabCreate
CiPhysQTabCreate
CiPhysQtabWrite
CiPhysQTabEngage
CiConDMACommand
```

This function should be called for both board and host QTABs. The function will set the QTab up appropriately for whichever type of QTab is being used. This function must be called before DMA is started.

19.8 CiPhysQTabChainLink

Prototype BFRC CiPhysQTabChainLink(Bd *Board*, PPQTABHEAD *ChainArray*, BFU32 *NumInChain*)

Description This function chains together a number of QTABs for sequential acquisition in host QTab mode.

Parameters

Board

Handle to board.

ChainArray

A array of pointers to QTABs which describe an set of buffers to be acquired into. The buffers will be filled in the order that their QTABs appear in this array.

NumInChain

The total number of QTab headers in the QTab chain array.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
BF_NULL_POINTER	*ChainArray* is NULL.
BF_NOT_CHAIN	*NumInChain* is not valid.
BF_BAD_CHAIN	Error walking *ChainArray*.

Comments This function effectively sets the board up for continuous acquisition into a sequence of host buffers. Each buffer in is DMAed into in turn, when the last buffer in the chain is filled, the board will DMA the next frame into the first buffer. In other words the chain describes a circular buffer.

The parameter *ChainArray* is an array of pointer to QTab headers. The QTab must already be created by calling CiRelQTabCreate and CiPhysQTabCreate. After this function return successfully, the chain must be engaged by calling CiPhysQTAB-ChainEngage function. The normal calling sequence for this function would be as follows:

```
loop for all buffers
   CiRelQTabCreate

loop for all buffers
   CiPhysQTabCreate

CiPhysQTabChainLink
```

```
CiPhysQTabChainEngage
CiDMACommand
CiConAqCommand
```

In the scenario above, no data will move until an acquisition command is sent to the board and the camera sends a frame to the board. Once data is flowing, the board will fill each buffer as the data comes in. Once the last buffer in the chain is filled, the board will continue starting with the first buffer. No host interaction is required for this process to work. The board will send a signal every frame (assuming CiRelQTabCreate was called with the CiDMAOptInt parameter) to tell your application when a frame is complete (use CiSignalWait).

Note: This function will only work for boards that support QTABs on the host, and will only work when board is in QTABs on the host mode. Only older Road Runners with the PLX 9060 chip have this limitation. All new Road Runners, R3s, Ravens and R64s support host QTABs.

19.9 CiPhysQTabChainBreak

Prototype BFRC CiPhysQTabChainBreak (Bd *Board*, PPQTABHEAD *ChainArray*)

Description Release QTABs from a chain so that they can be reused to build a subsequent chain.

Parameters

Board

Handle to board.

ChainArray

A array of pointers to QTABs which has been already passed to CiPhysQTabChainCreate.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
BF_BAD_CHAIN	Error walking *ChainArray*.

Comments This function is used to release the QTABs that are used by a chain. When a chain is built the QTABs that make it up are modified for use in the chain. If these QTABs need to be used again, the chain must first be broken with this function. After this function is called, the individual QTABs can be use to build another chain, presumable in a different order.

There is no need to call this function during cleanup if the individual QTABs are not going to be used again.

19.10 CiPhysQTabChainEngage

Prototype BFRC CiPhysQTabChainEngage(Bd *Board*, PPQTABHEAD *ChainArray*, BFU32 *NumInChain*)

Description Takes a successfully created chain and sets the board up to use it.

Parameters

Board

Handle to board.

ChainArray

This is an array of pointers to QTABs which describe an set of buffers to be acquired into. This parameter must first be passed to CiPhysQTabChainCreate.

NumInChain

The buffer number of the first frame in the chain to be acquired into.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments After a chain is created using CiPhysQTabChainCreate, the chain must be engaged using this function in order for the board to use it. Creating a chain is not a real time operation and should be done off line. If more than one chain is required, they should all be created first, then this function can be used to select which chain will be acquired into first.

See CiPhysQTabChainCreate for more information.

19.11 CiPhysQTabChainProgress

Prototype

BFRC CiPhysQTabChainProgress(Bd *Board*, PPQTABHEAD *ChainArray*, PBFU32 *pFrameNum*, PBFU32 *pLineNum*)

Description

Returns the line number and frame number of current image being DMAed.

Parameters

Board

Handle to board.

ChainArray

This is an array of pointers to QTABs which describe an set of buffers to be acquired into. This parameter must first be passed to CiPhysQTabChainCreate.

pFrameNum

Pointer to the number of the current frame being DMAed into.

pLineNum

Pointer to the number of the current line being DMAed.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_BOARDPTR	An invalid board handle was passed to the function.
BF_BAD_PTAB	The chain is not valid.

Comments

This function is used to check the progress of acquisition while the board is acquiring using a chain. The function will return both the line number and the frame number. This function is fairly computationally intensive and should not be called in a tight loop to monitor progress. This function is best used intermittently to check progress, for example, it can be interleaved with processing.

The best way to overlap acquisition and processing is to create a signal that waits for the end of frame signal (CiIntTypeEOD). Once the signal is asserted, the CPU can freely process the entire frame.

If you need to monitor the boards progress using a tight loop, read the VCOUNT register. Reading a register uses much less CPU time. Even in this case, you should put a sleep in your loop to not overwhelm the board with register reads (which take precedence over DMAing). Again is it better to interleave processing and checking VCOUNT.

19.12 CiChainSIPEnable

Prototype BFRC CiChainSIPEnable(Bd *Board*, PPQTABHEAD ChainArray)

Description Enables start-stop interrupt processing (SIP).

Parameters *Board*

Handle to board.

pRelQTabHead

Structure holding information about QTab.

Returns

CI_OK	If successful.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
Non-zero	On error.

Comments This function enables start-stop interrupt processing (SIP). This processing is used to reset the DMA engine in a kernel interrupt service routine.

When the board is in start-stop mode, the DMA is terminated before the frame is completely acquired. This termination leaves the DMA engine in an unknown state. The DMA engine must be reset and setup for the next host buffer before the next frame starts. Ordinarily this reset is performed by the application at the user level. However, in the case of a multi threaded application, the reset thread may not be able to reset the DMA engine before the beginning of the next frame (because of CPU load and thread priorities). To solve this problem the BitFlow SDK implements a DMA engine reset in the kernel level interrupt service routine. This code has higher priority than any user level threads. The latency and execution time of the SIP reset is minimized thus reducing the required minimum time between frames. This function turns on this functionality.

SIP only works (and is only required) when the board is in start-stop triggering mode (variable size image acquisition) and when a host QTab chain has been created and engaged. This function must be called before acquisition has started but after the QTab chain is created. This function enable the SIP resetting of the DMA engine, you must call CiChainSIPDisable to turn the SIP off. This SIP is based on the interrupt that occurs when the trigger de-asserts (the actual interrupt type depends on the board family being used).

The example application Flow demonstrates usage of this function.

Currently CiChainSIPEnable is not supported by the Raven.

19.13 CiChainSIPDisable

Prototype BFRC CiChainSIPDisable(Bd *BoardId,* PPQTABHEAD ChainArray)

Description Disables Start-Stop Interrupt Processing mode.

Parameters *Board*

Board ID.

pRelQTabHead

Structure holding information about QTab.

Returns

CI_OK	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
Non-zero	Function failed.

Comments See CiChainSIPEnable for details

Ci Control Tables

Chapter 20

20.1 Introduction

These functions allow an application to write directly to the control tables (CTAB). Normally the CTABs are initialized from a camera configuration file. However, because the CTABs control things like frame rate and exposure time, an application may want to modify them on-the-fly.

20.2 CiCTabPeek

Prototype BFU16 CiCTabPeek(Bd *Board*, BFU32 *Index*, BFU16 *Mask*)

Description Reads a single masked value from the Camera Control Table.

Parameters *Board*

Board ID.

Index

CTAB table offset.

For the Road Runner/R3:

0 - 0x2000 for horizontal CTABs
0 - 0x8000 for vertical CTABs

For the R64:

0 - 0x8000 for horizontal CTABs
0 - 0x20000 for vertical CTABs

Mask

CTAB bit extraction mask.

For the Road Runner/R3:

R2CTab
R2HCTab
R2VCTab
R2HCTabHEnd
R2HCTabHStart
R2HCTabClamp
R2HCTabField
R2HCTabHStrobe
R2HCTabHCon0
R2HCTabHCon1
R2HCTabHCon2
R2VCTabVEnd
R2VCTabVLoad
R2VCTabVStart
R2VCTabIRQ
R2VCTabVStrobe
R2VCTabVCon0
R2VCTabVCon1
R2VCTabVCon2

For the R64:

R64CTab
R64HCTab
R64VCTab
R64HCTabHStart
R64HCTabHReset
R64HCTabENHLoad
R64HCTabReserved
R64HCTabGPH0
R64HCTabGPH1
R64HCTabGPH2
R64HCTabGPH3
R64VCTabVStart
R64VCTabVReset
R64VCTabENVLoad
R64VCTabIRQ
BFVCTabVStrobe
R64VCTabGPV0
R64VCTabGPV1
R64VCTabGPV2
R64VCTabGPV3

Returns

A single masked CTAB entry	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	Unable to determine the model of board being used.

Comments

See the hardware reference manuals for details on what each CTAB column does.

20.3 CiCTabPoke

Prototype BFRC CiCTabPoke(Bd *Board*, BFU32 *Index*, BFU16 *Mask*, BFU16 *Value*)

Description Writes a single masked value to the Camera Control Table.

Parameters

Board

Board ID.

Index

CTAB table offset.

Mask

CTAB bit extraction mask (see CiCtabPeek).

Value

CTAB value.

Returns

CI_OK	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	Unable to determine the model of board being used.
R2_CTAB_POKE_BAD	CTAB poke failed.
R64_CTAB_POKE_BAD	CTAB poke failed.

Comments

20.4 CiCTabRead

Prototype BFRC CiCTabRead(Bd *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, PBFVOID *pDest*)

Description Reads masked CTAB values from the Camera Control Table.

Parameters

Board

Board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to read.

Mask

CTAB bit extraction mask (see CiCtabPeek).

pDest

Pointer to CTAB table storage (32 bits per entry).

Returns

CI_OK	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	Unable to determine the model of board being used.
CISYS_ERROR_UNKNOWN_ PARAMETER	A illegal parameter was passed to the function.
R2_CTAB_READ_BAD	CTAB read failed.
R64_CTAB_READ_BAD	CTAB read failed.

Comments

20.5 CiCTabWrite

Prototype BFRC CiCTabWrite(Bd *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, PBFVOID *pSource*)

Description Writes masked CTAB values to the Camera Control Table.

Parameters

Board

Board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to read.

Mask

CTAB bit extraction mask (see CiCtabPeek).

pSource

CTAB entries to write (32 bits per entry).

Returns

CI_OK	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	Unable to determine the model of board being used.
CISYS_ERROR_UNKNOWN_ PARAMETER	A illegal parameter was passed to the function.
R2_CTAB_WRITE_BAD	CTAB write failed.
R64_CTAB_WRITE_BAD	CTAB write failed.

Comments

20.6 CiCTabFill

Prototype BFRC CiCTabFill(Bd *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, BFU16 *Value*)

Description Writes a masked CTAB fill value to the Camera Control Table.

Parameters

Board

Board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to write.

Mask

CTAB bit extraction mask (see CiCtabPeek).

Value

CTAB fill value to write.

Returns

CI_OK	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	Unable to determine the model of board being used.
CISYS_ERROR_UNKNOWN_ PARAMETER	A illegal parameter was passed to the function.
R64_CTAB_FILL_BAD	CTAB fill failed.
R2_CTAB_FILL_ERR	CTAB fill failed.

Comments

20.7 CiCTabRamp

Prototype BFRC CiCTabRamp(Bd *Board*, BFU32 *StartIndex*, BFU32 *EndIndex*, BFU32 *StartVal*, BFU32 *EndValue*)

Description Writes a ramp function (i.e. 0,1,2,3,4...) to the Camera Control Table.

Parameters

Board

Board ID.

StartIndex

CTab start table offset.

EndIndex

CTab end table offset.

StartVal

CTab start value.

EndValue

CTab end value.

Returns

CI_OK	Function succeeded.
CISYS_ERROR_NOTSUP-PORTED	The model dose not support this function.
CISYS_ERROR_BAD_ BOARDPTR	Unable to determine the model of board being used.
BF_BAD_ALLOC	Unable to allocate memory for the ramp.
R64_CTAB_RAMP_BAD	Error writing ramp.
R2_CTAB_RAMP_BAD	Error writing ramp.

Comments This function writes an integer ramp function, starting at the value *StartVal* and ending at the value *EndValue*. The ramp function is distributed evenly between the CTAB entries start at the index *StartIndex* and ending at *EndIndex*. If the request ramp does no result in integer values at for every entry, the actual values are rounded.

20.8 CiCTabVSize

Prototype BFU32 CiCTabVSize(Bd *Board*)

Description Returns the vertical CTab size.

Parameters *Board*

Board ID.

Returns Returns the vertical CTab size.

Comments

20.9 CiCTabHSize

Prototype BFU32 CiCTabHSize(Bd *Board*)

Description Returns the horizontal CTab size.

Parameters *Board*

Board ID.

Returns Returns the horizontal CTab size.

Comments

Road Runner and R3 Introduction

Chapter 21

21.1 Overview

The Road Runner and the R3 are a high-performance, PCI-based frame grabber families. They are targeted at digital cameras and general purpose digital data acquisition. Their DMA engine transfers data directly into system memory. Their digital ports can accept four 8-bit taps. On board circuitry will reformat the data "on the fly." Both the Road Runner and the R3 come in two major version: one supports the LVDS/RS422 differential cameras and the other supports Camera Link cameras. Figure 21-1 shows the block diagram of the R2422/LVDS version of the Road Runner.

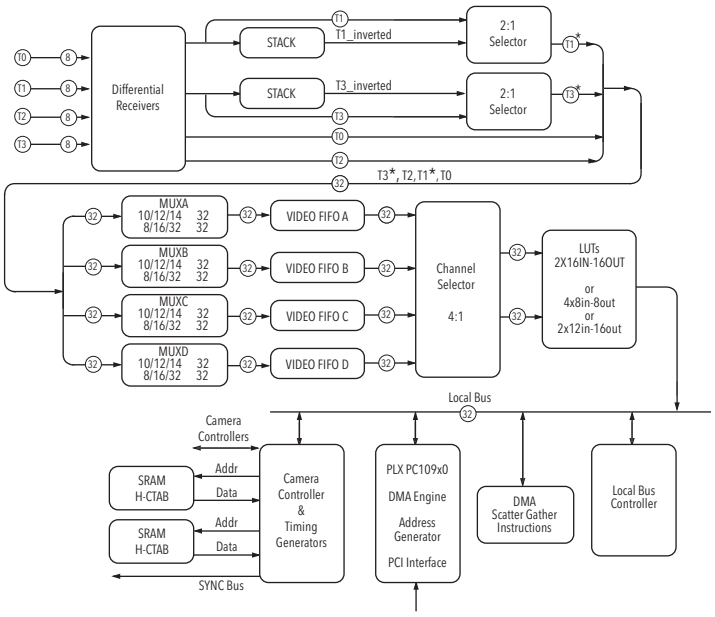

Figure 21-1 Road Runner Block Diagram

Figure 21-2 shows the diagram for the Road Runner Camera Link.

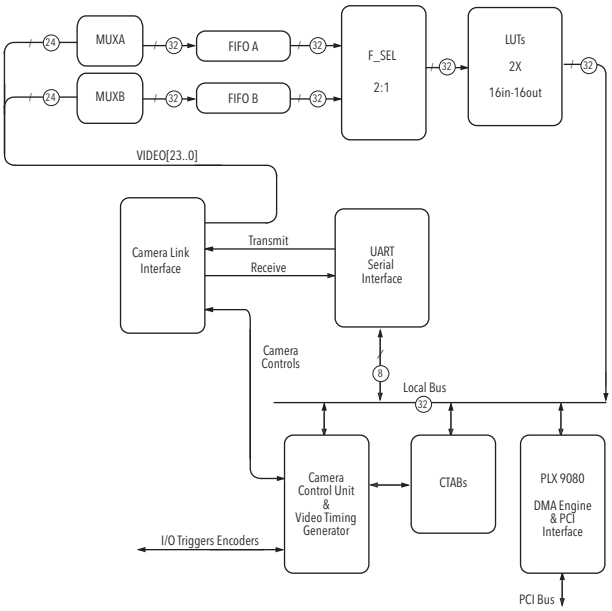

Figure 21-2 Road Runner Camera Link Block Diagram.

21.2 Where is the R3 or PMC API?

The API described in the book was originally written for the Road Runner family, thus all the functions start with "R2". When the R3 product was being developed, the goal was to make a simple migration path from the Road Runner to the R3. The best way to do this is for the new product to use the old API. Thus and Road Runner application will run, unchanged on an R3 (assuming the application is built with a SDK release that supports the R3). Thus the functions in this book apply to both versions of Road Runner and the R3.

In the rare case where an application needs to know specifics of the board installed. There is a function BFIsCl that will indicate which type of front end the board has (i.e. LVDS/R422 or Camera Link). There is also a function BFIsR3 that will indicate where the Road Runner or the R3 family is installed.

The PMC board is a R3-23 in a different form factor. All functions that are valid for the R3 are valid for the PMC.

Road Runner/R3 System Open and Initialization

Chapter 22

22.1 Introduction

The functions described in this chapter are quite simple; the idea is to find the board or boards that you want to work with, then open and optionally initialize them. When you are finished, close the system up, thus cleaning up all resources allocated in the open function.

A normal program would uses these functions, in this order:

```
R2SysBoardFindByXXXX
R2BrdOpen

// acquisition and processing

R2BrdClose
```

If you want to open two boards, the flow would be as follows:

```
R2SysBoardFindByXXXX   // find board 0
R2BrdOpen   // open board 0
R2SysBoardFindByXXXX       // find board 1
R2BrdOpen   // open board 1

// acquisition and processing

R2BrdClose   // close board 0
R2BrdClose   // close board 1
```

The board find functions are used to make sure that you are opening the correct board in a multi-board system. If you have only one board, then the call is trivial.

Note: There is currently only one board find function, R2SysBoardFindByNum.

The handle return by the function R2BrdOpen is used in all subsequent function calls. If you are using two or mode boards, open each board and store each handle in a separate variable. Whenever you want to talk to board X, pass the handle for board X to the function.

There is no need to call R2BrdOpen more than once per process per board. Because this function takes a fair amount of CPU time and allocated resources, we discourage users from repeatedly calling R2BrdOpen and the R2BrdClose in a loop. We recommend opening the board once, when the application starts, and closing it once when the application exits. If you are using a program that has multiple threads, open the board once in the first

main thread and then pass the board handle to every thread that is subsequently created. You must call R2BrdClose for every board that is open with R2BrdOpen. You should also call R2BrdClose in the same thread the R2BrdOpen was called in.

22.2 R2SysBoardFindByNum

Prototype R2RC R2SysBoardFindByNum(BFU32 *Number*, PR2ENTRY *pEntry*)

Description Finds a Road Runner/R3 on the PCI bus with a given number.

Parameters

Number

The number of the board to find. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots as long as the number of boards and the Road Runner/R3 is in the same PCI slot.

pEntry

A pointer to an empty R2ENTRY structure, used to tell the R2BrdOpen function which board to open.

Returns

R2_OK	The board was successfully found.
R2SYS_ERROR_NOT-FOUND	There is no board with this number.
R2SYS_ERROR_REGISTRY	An error occurred searching for the board information in the registry.
R2SYS_ERROR_SIZE	Internal error.

Comments If you have only one board in your system set *Number* = 0 and only call this function once. This function can be used to enumerate all of the boards in a system. It can be called repeatedly, incrementing *Number* each time, until the function returns R2SYS_ERROR_NOTFOUND.

There is no standard way to correlate the *Number* parameter of this function to the PCI slot number. Every motherboard and BIOS manufacturer has a different scheme. You can use the system configuration utility, SysReg, to determine the relationship between slot number and board *Number*, by setting the board ID switches different for each board in your system and walking through all the installed boards.

22.3 R2BrdOpen

Prototype R2RC R2BrdOpen(PR2ENTRY *pEntry*, RdRn **pBoard*, BFU32 *Mode*)

Description Opens a Road Runner/R3 for access. This function must return successfully before any other Road Runner/R3 SDK functions are called (with the exception of R2SysBoard-FindXXXX functions).

Parameters

pEntry

A pointer to a filled out R2ENTRY structure. This structure describes which board is to be opened. The structure is filled out by a call to one of the R2SysBoardFindXXXX functions.

**pBoard*

A pointer to a RdRn handle. This handle is used for all further accesses to the newly opened board. This function takes a pointer to a handle where as all other functions just take a handle.

Mode

This parameter allows for different modes of opening the board, one or more of these parameters can be ORed together:

0 - board will open normally but not initialized. Board registers are not changed.
R2SysInitialize - initialize the board.
R2SysExclusive - open only if no other process has, and do not allow any subsequent process to open the board.
R2SysNoIntThread - do not start interrupt IRP thread.
R2SysNoCameraOpen - do not open any configured cameras.
BFSysNoAlreadyOpenMess - suppress already open warning message.
BFSysNoOpenErrorMess - suppress all error popups in open function
BFSysSecondProcessOpen - special mode that allows the board to be opened twice in the same process (includes of some of the above modes).

Returns

R2_OK	Function was successful.
R2_ALREADY_OPEN_PROC	Another thread in the process has already opened the board, this open not allowed.
R2_ALREADY_OPEN_EXEC_YOU	Another process has opened the board in R2SysExclusive mode, this open is not allowed.
R2_ALREADY_OPEN_EXEC_ME	You have attempted to open the board in R2SysExclusive but the board is already opened by another process, this open not allowed.

R2_BAD_MUTEX	Error occurred allocating a MUTEX object from the operating system.
R2_BAD_CAM	Error opening one of the camera files configured for this board.
R2_BAD_INIT	Error initializing the board.
R2SYS_ERROR_ALLOCATION	Error allocating resources required for this board.
R2SYS_ERROR	Low-level error opening board.

Comments

This function opens the board for all accesses. Call one of the R2SysBoardFindXXXX functions first to find the board you wish to open. Then call this function to open to board. The board must be opened before any other functions can be called. When you are finished accessing the board you must call R2SysBrdClose, before exiting your process. Failure to call R2SysBrdClose will result in incorrect board open counts used by the driver.

If this function fails, you cannot access the board. Also, you do not need to call R2SysBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board.

You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call R2SysBrdClose first.

Calling this function with *Mode* = R2SysInitialize initializes the board and sets it up for the first camera that is configured for this board. If another process has already opened the board using this flag, the board will not be re-initialized, but you will have access to the board in the state that it is.

The *Mode* = R2SysExclusive is designed to guarantee that only one process can have the board open at a time. If the board has already been opened with this flag you will not be able to open it again, regardless of the *Mode* parameter that you use.

If *Mode* = R2SysExclusive, then you will not be able to open the board if any other process has already opened the board, regardless of the mode the other process used to open the board. Finally, if you do succeed in opening the board in this mode, no other processes will be allowed to open the board.

22.4 R2BrdOpenCam

Prototype R2RC R2BrdOpenCam(PR2ENTRY *pEntry*, RdRn **pBoard*, BFU32 *Mode*, PBFCHAR *ForceCamFile*)

Description Opens a Road Runner/R3 for access. This function must return successfully before any other Road Runner/R3 SDK functions are called (with the exception of R2SysBoard-FindXXXX functions).

Parameters

pEntry

A pointer to a filled out R2ENTRY structure. This structure describes which board is to be opened. The structure is filled out by a call to one of the R2SysBoardFindXXXX functions.

**pBoard*

A pointer to a RdRn handle. This handle is used for all further accesses to the newly opened board. This function takes a pointer to a handle where as all other functions just take a handle.

Mode

This parameter allows for different modes of opening the board, one or more of these parameters can be ORed together:

0 - board will open normally but not initialized. Board registers are not changed.
R2SysInitialize - initialize the board.
R2SysExclusive - open only if no other process has, and do not allow any subsequent process to open the board.
R2SysNoIntThread - do not start interrupt IRP thread.
R2SysNoCameraOpen - do not open any configured cameras.
BFSysNoAlreadyOpenMess - suppress already open warning message.
BFSysNoOpenErrorMess - suppress all error popups in open function
BFSysSecondProcessOpen - special mode that allows the board to be opened twice in the same process (includes of some of the above modes).

ForceCamFile

The camera file to open. The camera file should include the name and the file extention. If only the file name and extention are given, the camera configuration path is searched for the camera file. (The camera configuration path by default is the Config folder under the SDK root.) If the full path is given, the camera file will try and be opened from that location.

Returns

R2_OK	Function was successful.

R2_ALREADY_OPEN_PROC	Another thread in the process has already opened the board, this open not allowed.
R2_ALREADY_OPEN_EXEC_YOU	Another process has opened the board in R2SysExclusive mode, this open is not allowed.
R2_ALREADY_OPEN_EXEC_ME	You have attempted to open the board in R2SysExclusive but the board is already opened by another process, this open not allowed.
R2_BAD_MUTEX	Error occurred allocating a MUTEX object from the operating system.
R2_BAD_CAM	Error opening one of the camera files configured for this board.
R2_BAD_INIT	Error initializing the board.
R2SYS_ERROR_ALLOCATION	Error allocating resources required for this board.
R2SYS_ERROR	Low-level error opening board.

Comments

This function opens the board for all accesses. Call one of the R2SysBoardFindXXXX functions first to find the board you wish to open. Then call this function to open to board. The board must be opened before any other functions can be called. When you are finished accessing the board you must call R2SysBrdClose, before exiting your process. Failure to call R2SysBrdClose will result in incorrect board open counts used by the driver.

If this function fails, you cannot access the board. Also, you do not need to call R2SysBrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board.

You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call R2SysBrdClose first.

Calling this function with *Mode* = R2SysInitialize initializes the board and sets it up for the first camera that is configured for this board. If another process has already opened the board using this flag, the board will not be re-initialized, but you will have access to the board in the state that it is.

The *Mode* = R2SysExclusive is designed to guarantee that only one process can have the board open at a time. If the board has already been opened with this flag you will not be able to open it again, regardless of the *Mode* parameter that you use.

If *Mode* = R2SysExclusive, then you will not be able to open the board if any other process has already opened the board, regardless of the mode the other process used to open the board. Finally, if you do succeed in opening the board in this mode, no other processes will be allowed to open the board.

22.5 R2BrdCamSel

Prototype R2RC R2BrdCamSel(RdRn *Board*, BFU32 *CamIndex*, BFU32 *Mode*)

Description Sets a board's current camera to the camera with the given index. Depending on the Mode, the board can also be initialized for this camera.

Parameters

Board

Handle to board.

CamIndex

Index of camera to become current. Index is set in SysReg.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but do not modify the board.
R2SysConfigure - initialize the board for this camera.

Returns

R2_OK	Function was successful.
R2_INCOMP	Camera file is incompatible with this board, or camera file is incompatible with this version of the SDK.
R2_BAD_CNFG	An error occurred initializing the board for this camera file.

Comments Each board has associated with it a list of configured cameras (set in the SysReg application) and a current camera. By default, the current camera is the first camera in the list of configured cameras. The current camera is important because it dictates the parameters used for acquisition. There must be a current camera set in order to use the acquisition functions. This function allows you to pick one of the configured cameras to be the current camera.

If *Mode* = R2SysConfigure, the board will be initialized for the given camera.

This function is useful for switching on-the-fly between multiple pre-configured camera types.

22.6 R2BrdCamSetCur

Prototype R2RC R2BrdCamSetCur(RdRn *Board*, PR2CAM *pCam*, BFU32 *Mode*)

Description Sets the current camera to the camera object pCam that is not necessarily one of the pre-configured cameras. The board can be optionally initialized to the camera.

Parameters

Board

Handle to board.

pCam

A camera object.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but does not modify the board.
R2SysConfigure - initialize the board for this camera.

Returns

R2_OK	Function was successful.
R2_INCOMP	Camera file is incompatible with this board, or camera file is incompatible with this version of the SDK.
R2_BAD_CNFG	An error occurred initializing the board for this camera file.

Comments This function sets the current camera to a camera object that is not one of the cameras already configured for the board (via SysReg). The camera must already be opened successfully (see R2CamOpen).

This function allows you to handle your own camera management. You can select, open, configure and close cameras to suit your applications needs independently of the SDK's camera management.

If *Mode* = R2SysConfigure, the board will be initialized for the given camera.

22.7 R2BrdInquire

Prototype R2RC R2BrdInquire(RdRn *Board*, BFU32 *Member*, PBFU32 *pVal*)

Description Returns parameters about the current board.

Parameters

Board

Handle to board.

Member

Parameter to inquire about:

R2BrdInqModel - returns the board model. The parameter *pVal* will point to one of:

R2BrdValModel11
R2BrdValModel12
R2BrdValModel13
R2BrdValModel14
R2BrdValModel23
R2BrdValModel24
R2BrdValModel44

R2BrdInqSpeed - returns the board receivers speed. The parameter *pVal* will point to one of:

R2BrdValSpeed40MHz
R2BrdValSpeedNormal

R2BrdInqLUT - the type of LUT mounted on this board. The parameter *pVal* will point to one of:

R2BrdValLUTNone
R2BrdValLUT16
R2BrdValLUT8And12

R2BrdInqIDReg - the current setting of the ID switch on the board (0,1,2,3).

Camera inquiry parameters are also valid. The *pVal* parameter will point to the value for the board's current camera. See R2CamInquire for the meaning of these members.

R2CamInqXXXX

pVal

Pointer returned containing the requested value.

Returns

R2_OK	Function was successful.
R2_BAD_INQ_PARAM	The *Member* parameter is unknown.

Comments

This function is used to inquire of the system characteristics of the board. This function can also be called with R2CamInquire *Members*, which are then passed to that function using the board's current camera.

22.8 R2BrdClose

Prototype R2RC R2BrdClose(RdRn *Board*)

Description Closes the board and frees all associated resources.

Parameters *Board*

Handle to board.

Returns

R2_OK	In all cases.

Comments This function closes the board and releases associated resources. This function must be called whenever a process exits regardless of the reason the process is exiting. The only time that this function does not have to be called is if R2SysBrdOpen fails. This function decrements the internal counters that are used to keep track of the number of processes that have opened the board.

22.9 R2BrdAqTimeoutSet

Prototype R2RC R2BrdAqTimeoutSet(RdRn *Board*, BFU32 *Timeout*)

Description Sets the timeout value for this board's current camera.

Parameters *Board*

Board to select the camera for.

Timeout

New value for timeout, in milliseconds.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function sets the timeout value for this board's current camera.

22.10 R2BrdAqSigGetCur

Prototype R2RC R2BrdAqSigGetCur(RdRn *Board*, PBFVOID **pAqSig*)

Description Gets the current acquire signal.

Parameters

Board

Board to select.

****pAqSig***

Pointer to storage for acquire signal.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function gets the current acquire signal. See the section on signals to understand what a signal is.

22.11 R2BrdAqSigSetCur

Prototype R2RC R2BrdAqSigSetCur(RdRn *Board*, PBFVOID *pAqSig*)

Description Sets the current acquire signal to a signal record provided by the caller.

Parameters

Board

Board to select.

pAqSig

Pointer to caller's signal record.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function sets the current acquire signal to a signal record provided by the caller.

22.12 R2BrdQTabGetCur

Prototype R2RC R2BrdQTabGetCur(RdRn *Board*, PBFVOID **pQuad*, BFU32 *QuadBank*)

Description Gets the current quad table pointer.

Parameters

Board

Board to select.

**pQuad*

Pointer to new current quad table.

QuadBank

Quad bank to set

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function gets the current quad table pointer.

22.13 R2BrdQTabSetCur

Prototype R2RC R2BrdQTabSetCur(RdRn *Board*, PBFVOID *pQuad*, BFU32 *QuadBank*)

Description Sets the current quad table pointer to a quad table the user has built.

Parameters

Board

Board to select.

pQuad

Pointer to new current quad table.

QuadBank

Quad bank to set.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function sets the current quad table pointer to a quad table the user has built.

22.14 R2BrdCamGetFileName

Prototype R2RC R2BrdCamGetFileName(RdRn *Board*, BFU32 *Num*, PBFCHAR *CamName,* BFSIZET *CamNameStLen*)

Description Gets the file name of the attached camera(s).

Parameters

Board

Board to select.

Num

Camera number to get the name of.

CamName

Contains the file name of the camera configuration.

CamNameStLen

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *CamName*.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function can be used to get the file name for one of the attached camera configurations. These configurations are attached to the board in SysReg. The *Num* parameter corresponds to the number configuration in the list of attached cameras in SysReg.

22.15 R2BrdCamGetCur

Prototype R2RC R2BrdCamGetCur(RdRn *Board*, PR2CAM **pCam*)

Description Gets a pointer to the current camera configuration structure.

Parameters

Board

Board to select.

**pCam*

Pointer to new current quad table.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function a pointer to the current camera configuration structure. The structure contains all the data that is in the camera configuration file.

Road Runner/R3 Acquisition

Chapter 23

23.1 Introduction

The Acquisition Functions are some of the most important in the SDK. While the initialization functions set up the board's registers for a particular camera, these functions do most of the work required to get the board reading to DMA the images to memory.

The functions are organized into three groups:

Setup functions
Command function
Clean up functions

The concept here is that the setup functions are time and CPU intensive, so they should be called before any time critical processing has begun. In a sense, these are extensions of the initialization process. Once the setup functions are called for a particular destination buffer, they need not be called again.

The command function is designed to be used during time critical operations, and require minimal CPU time. They can be told to return immediately so that other operations can be performed simultaneously with acquisition. The command function can be called over and over, as many times as needed, to acquire the buffers locked down in the setup functions.

The cleanup functions free up any resources allocated in the setup functions, and put the DMA engine in an idle mode.

For example, the basic flow of a program would be:

```
// Initialization

R2AqSetup
Loop
{
   R2AqCommand
}
R2AqCleanup
```

The bulk of the work is done in the R2AqSetup functions. These functions create a scatter gather table based on the virtual memory address, called a relative QTab.

The relative QTab is passed to the kernel driver, where the destination buffer is locked down (so that it cannot be paged to disk) and the physical address are determined for each page of the buffer (NT usually uses 4096 byte pages). These physical addresses are used to build a physical QTab. This physical QTab is then written to the board in preparation scatter gather DMAing.

Finally, the DMA engine is initialized and started. Again, this function need be called only once, for a particular destination buffer.

This function also supports dual buffer acquisition. In this case, the setup functions are called twice, once for each buffer. Whenever an application is finished acquiring to a buffer, one of the R2AqCleanUp functions must be called. You cannot call R2AqSetup for a different buffer before calling R2AqCleanUp for the previous buffer. The only exception to this is in the case dual buffering. When using dual buffers, R2AqClean need only be called once to cleanup from both calls to R2AqSetup.

The R2AqCommand can be called either synchronously or asynchronously. In the synchronous case, the function does not return until the command has completed. In the asynchronous case, the function returns as soon as the command has been issued to the board. If you need to synchronize your process with the acquisition, you can use the R2AqWaitDone function or you can use the signaling system. Signaling is the best way to synchronize to repeated end of frame signals as they do not take any CPU cycles.

The R2AqSetup and R2AqCleanup functions are designed to handle acquisition to host based memory or virtual memory. The R2AqDispSetup and R2AqDispCleanUp are designed to handle acquisition to physical memory, such as a VGA or other destination on the PCI bus. The main difference is that the destination pointer passed to the setup function points to a virtual memory buffer (and thus a physical page address must be calculated) in the R2AqSetup case, and points to actual physical memory in the R2AqDispSetup case.

23.2 R2AqSetup

Prototype R2RC R2AqSetup(RdRn *Board*, PBFVOID *pDest*, BFU32 *DestSize*, BFS32 *Stride*, BFU32 *DestType*, BFU32 *LutBank*, BFU32 *LutMode*, BFU8 *QuadBank*, BFBOOL *FirstBank*)

Description Sets a Road Runner/R3 for acquisition to a host buffer. This function must be called before any acquisition command is issued.

Parameters

Board

Handle to board.

pDest

A void pointer to the destination buffer (already allocated).

DestSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in *bytes*, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

DestType

Type of destination memory:

R2DMADataMem - host memory
R2DMABitmap - display memory

LutBank

The LUT bank to pass the image through:

R2LutBank0 - LUT bank 0
R2LutBank1 - LUT bank 1
R2LutBypass - bypass LUTs

LutMode

The mode of the LUT to use:

R2Lut8Bit - LUTs are programmed as 8 bits
R2Lut12Bit - LUTs are programmed as 12 bits

R2Lut16Bit - board has 16-bit LUTs (only available on 16-bit LUT boards)

QuadBank

The QTab bank to use:

R2QTabBank0 - use bank 0
R2QTabBank1 - use bank 1

FirstBank

For acquisition to single buffer, set to TRUE.

For acquisition using two ping-pong buffers, this parameter is used to indicate which buffer will be acquired into first:

TRUE - for first bank to acquire into
FALSE - for second bank to acquire into

Returns

R2_OK	If successful.
R2_BAD_ALLOC	Resources required for this operation could not be allocated.
R2_CON_QTAB_BANK_ERR	The QuadBank parameter is invalid.
R2_AQSETUP_FAIL	Other failure.

Comments

This function sets up the entire Road Runner/R3's acquisition systems for acquisition to host. It will set up QTABs (relative and physical), and write them to the board. The QTABs are based on the current camera pointer in the board structure. This function need be called only once, before acquisition begins. It does not need to be called again unless R2AqCleanUp is called. R2AqCleanUp should be called when done acquiring in order to free up resources used by this process. The only reason to call this function again is to acquire into a different host buffer or acquire with a different type of camera. Once this function is called, the function R2AqCommand is used to snap, grab, freeze or abort acquisition.

23.3 R2AqCommand

Prototype R2RC R2AqCommand(RdRn *Board*, BFU32 *Command*, BFU32 *Mode*, BFU8 *QuadBank*)

Description Once the Road Runner/R3 is set up for acquisition with R2AqSetup, this function issues the actual acquisition command.

Parameters

Board

Handle to board.

Command

Acquisition command to initiate:

R2ConGrab - starting at the beginning of the next frame, acquire every frame.
R2ConSnap - starting at the beginning of the next frame, acquire one frame.
R2ConFreeze - stop acquiring at the end of the current frame. If in between frames, do not acquire any more frames.
R2ConAbort - stop acquiring immediately. If in the middle of the frame, the rest of the frame will not be acquired.
R2ConReset - reset conditions after an abort or overflow. The board is set up as it was when R2AqSetup was called.

Mode

This function can operate in two modes:

R2ConAsync - as soon as the command is issued return.
R2ConWait - wait for the current command to complete. For a snap, the function will return when the entire frame has been acquired into memory. For a grab, the function will wait until the first frame has begun to be acquired. For a freeze, the function waits for the current frame to end. All other commands return immediately.

QuadBank

When *Command* = R2ConReset, this is the QTab bank to be set up for. See R2AqSetup. For an R2 using host QTabs or an R3, this parameter is ignored.

Returns

R2_OK	If successful.
R2_AQ_NOT_SETUP	R2AqSetup has not yet been called and the board is not ready for an acquisition command.

R2_BAD_AQ_CMD	A snap or grab command has already been issued and the board is already acquiring.
R2_BAD_STOP	The function was unable to reset the board.
R2_CON_QTAB_BANK_ERR	The *QuadBank* parameter is incorrect.
RS_BAD_DMA_SETUP	The board has not been set up properly for DMA.
R2_AQSTRT_TIMEOUT	A time-out occurred waiting for acquisition to begin.
R2_AQEND_TIMEOUT	A time-out occurred waiting for acquisition to end.

Comments

This function can only be called after R2AqSetup is called. R2AqSetup need only be called once for any number and combination of calls to R2AqCommand. Basically, you call R2AqSetup once for a given host buffer, then call R2AqCommand as many times as you need to get data into that buffer. Call R2AqCleanUp when you are done acquiring into that buffer. Then the procedure starts over again for the next buffer.

The R2AqXXXX commands handle both DMA and camera acquisition. No other commands are needed to handle the process of acquiring into memory.

If you call this function with *Mode* = R2ConWait, it will wait for the acquisition to complete, in the case of a snap or freeze command, or wait for the acquisition to begin, in the case of a grab command. This is an efficient wait that consumes minimal CPU cycles. The function will return when the last pixel has been DMAed into memory. Alternatively, you can call the function with *Mode* = R2ConAsync, and the function will return as soon as the command has been issued. You can find out how much data has been DMAed by calling R2AqProgress. You can also just wait for the end of acquisition by calling R2AqWaitDone.

The functions mentioned above use the SDK's signaling system to efficiently wait for events. If you wish to have a higher level of control you can call the R2SignalXXXX functions yourself. These functions use a signaling system that allow processes to be notified of Road Runner/R3 events and interrupts. For acquisition, wait for the R2IntTypeEOD signal. This signal occurs at the end of every frame, in both grab and snap mode. This signal occurs when the last pixel is DMAed into memory.

Calling this function with *Command* = R2ConAbort will stop acquisition immediately. The acquisition process can be left anywhere in the frame. You must call this function with *Command* = R2ConReset before any more acquire commands can be issued. Alternatively, you can call R2AqCleanUp and start over with R2AqSetup

23.4 R2AqCleanUp

Prototype R2RC R2AqCleanUp(RdRn *Board*)

Description Frees all resources used by the acquisition process. Makes sure the board is in a stable state.

Parameters *Board*

Handle to board.

Returns

R2_OK	In all cases.

Comments This function frees all of the resources that were allocated in R2AqSetup. Do not call this function unless you have already called R2AqSetup, and unless you are finished acquiring into the current buffer.

This function does not free the destination buffer passed to R2AqSetup in the *pDest* parameter.

23.5 R2AqWaitDone

Prototype R2RC R2AqWaitDone(RdRn *Board*)

Description Waits for the current acquisition to complete.

Parameters *Board*

Handle to board.

Returns

R2_OK	The current acquisition has completed.
R2_AQ_NOT_SETUP	The acquisition process has not been set up yet.
R2_BAD_WAIT	The board is currently in grab mode and acquisition will not end, or there is another acquisition command pending after this one is completed.
R2_AQEND_TIMEOUT	The acquisition time-out expired before the acquisition command completed.

Comments This function efficiently waits for the current acquisition to complete. The completion is denoted by the last pixel being DMAed into memory. The function will return with a time-out error if the acquisition has not been completed by the designated acquisition time-out amount. This time is normally set in the camera configuration file, but can be changed in software as well, see R2CamSetTimeout. This function will return immediately if the acquisition has already completed. This function will also return immediately (with an error code), if the board is in a state where acquisition will not complete without further acquisition commands.

23.6 R2AqNextBankSet

Prototype R2RC R2AqNextBankSet(RdRn *Board*, BFU8 *QuadBank*)

Description For ping-pong style acquisition, this function sets up the board for the next QTab bank to use for acquisition.

Parameters

Board

Handle to board.

QuadBank

Next QTab bank to use for acquire - must be 0 or 1.

Returns

R2_OK	In all cases.

Comments This function sets the next QTab bank to use for acquisition. The Road Runner/R3 has two QTab banks that can be loaded for two destinations by calling R2AqSetup twice. By default, the Road Runner/R3 will use the same bank continuously. This function can be called any time during the current frame to switch the bank for the next frame.

You must call R2AqSetup twice (once with the parameter *QuadBank* = 0 and once with *QuadBank* = 1) before using this function. Switching to a QTab bank that has not been loaded will cause unpredictable behavior.

In host QTab mode or with an R3 board, this function is not needed since the qtabs are build in the host computer's memory.

23.7 R2AqFrameSize

Prototype R2RC R2AqFrameSize(RdRn *Board*, BFU32 *XSize*, BFU32 *YSize*)

Description This function provides the ability to change the image height and image width.

Parameters

Board

Handle to board.

XSize

The value to change the XSize too.

YSize

The value to change the YSize too.

Returns

R2_OK	If successful.
R2_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R2_CAM_SUPPORT	Cam file being used is not supported by this function.
R2_HCTAB_X16	Pixel clock divided by 16 is not supported.
R2_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R2_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
BF_BAD_ALLOC	Couldn't allocate memory.

Comments This function is used to change the size of the image being acquired, from software. With this function the size of the frame can be changed on the fly, without the use of camera files. This function is limited to use with only free run camera files, and may not work with sophisticated camera files.

This function assumes the CTABs and control registers have already been initialized to a working state by one of the initialization functions (e.g. R2BrdOpen). The function uses the current state to determine how to make the requested modifications. If the current board state is non-functional, this function will fail.

This function can be called before R2AqSetup and the new size will overwrite the size specified by the camera file. To change the size after R2AqSetup has been called R2AqCleanup must be called then R2AqFrameSize and R2AqSetup. The following is an example of the order needed to change the size of the frame after R2AqSetup has been called:

```
// Stop acquisition
```

```
R2AqCleanUp
R2AqFrameSize
R2AqSetup
// Begin acquisition
```

For a complete example on how to use the XXAqFrameSize function, see the CiChangeSize example included in the SDK.

The minimum XSize this functions supports is 4 and a minimum YSize of 2. The maximum YSize is 32768 and the maximum XSize is 8192. This function will return a R2_BAD_FRM_SIZE error for any of these problems. Another precaution to take is that the XSize needs to be a multiple of 4. Any XSize value that is not a multiple of 4 will give a R2_BAD_FRM_SIZE error.

It is left up to the user not to exceed the sensor size of the camera. For example if the user is using a area scan camera with a sensor size of 640x480 and tries and make the frame size 800x600, this function will try to acquire the 800x600 frame size even though the camera can not provide it. The user will end up with a scrambled or unstable image.

This function only supports the pixel clock divided by 4. If the pixel clock divided by 16 is being used, error R2_HCTAB_X16 will be returned.

23.8 R2AqReengage

Prototype R2RC R2AqReengage(RdRn *Board*, BFU8 *QuadBank*)

Description Engages the physical QTab for the given bank.

Parameters

Board

Handle to board.

QuadBank

The next physical QTab bank to use.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	QuadBank is not equal to R2QTabBank0 or R2QTab-Bank1
R2_AQ_NOT_SETUP	R2AqSetup has not yet been called and the board is not ready for an acquisition command.
BF_NULL_POINTER	ChainArray is NULL.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board.
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments This function is used to engage the physical QTab for the bank specified by the Quad-Bank parameter. This function only needs to be used if the acquisition or the DMA is aborted in the middle of the frame (for example, when using start-stop triggering). This function is intended to be used with qtabs on the host. However, calling it with board qtabs will not cause any problems.

23.9 R2AqROISet

Prototype

R2RC R2AqROISet(RdRn *Board*, BFU32 *XOffset*, BFU32 *YOffset*, BFU32 *XSize*, BFU32 *YSize*)

Description

This function provides the ability to change the region of interest acquired by the camera.

Parameters

Board

Handle to board.

XOffset

The number of pixels to offset in the x-axis.

YOffset

The number of pixels to offset in the y-axis.

XSize

The value to change the XSize too.

YSize

The value to change the YSize too.

Returns

R2_OK	If successful.
R2_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R2_CAM_SUPPORT	Cam file being used is not supported by this function.
R2_HCTAB_X16	Pixel clock divided by 16 is not supported.
R2_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R2_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
BF_BAD_ALLOC	Couldn't allocate memory.

Comments

This function is used to change the region of interest (ROI) of the image being acquired from the camera, from software. With this function the ROI of the frame can be changed on the fly, without the use of camera files. This function is limited to use with only free run camera files, and may not work with sophisticated camera files.

This function assumes the CTABs and control registers have already been initialized to a working state by one of the initialization functions (e.g. R2BrdOpen). The function uses the attached camera file to determine how to make the requested modifications. The ROI must stay within the boundaries of the attached camera sensor being use. If the current board state is non-functional, this function will also be non-functional.

This function can be called before R2AqSetup and the new settings will overwrite the settings specified by the camera file. To change the size after R2AqSetup has been called, R2AqCleanup must be called then R2AqROISet and R2AqSetup. The following is an example of the order needed to change the ROI of the frame after R2AqSetup has been called:

```
// Stop acquisition
R2AqCleanUp
R2AqROISet
R2AqSetup
// Begin acquisition
```

The minimum XSize this functions supports is 4 and a minimum YSize of 2. The maximum YSize is 32768 and the maximum XSize is 8192. This function will return a R2_BAD_FRM_SIZE error for any of these problems. Another precaution to take is that the XSize needs to be a multiple of 4. Any XSize value that is not a multiple of 4 will give a R2_BAD_FRM_SIZE error.

It is left up to the user to verify that the ROI dose not exceed the x and y sizes or boundaries in the camera sensor. For example if the user is using a area scan camera with a sensor size of 640x480 and tries and make the frame size 800x600, this function will try to acquire the 800x600 frame size even though the camera can not provide it. The user will end up with a scrambled or unstable image. Another example would be if the same 640x480 camera file is used with an xsize that is less than 640 and a ysize that is less then 480, but the x or y offset puts the ROI beyond the 640x480 borders.

This function only supports the pixel clock divided by 4. If the pixel clock divided by 16 is being used, error R2_HCTAB_X16 will be returned.

Road Runner/R3 Camera Configuration

Chapter 24

24.1 Introduction

One of the most powerful features of the Road Runner/R3 is the ability for the board to interface to an almost infinite variety of cameras. The knowledge behind these interfaces is stored in the camera configuration files.

The normal way a Road Runner/R3 application works is that the board is initialized to interface to the camera currently attached to the board. The currently attached camera is selected in the SysReg utility program. Normally, an application is written so that it will work with whatever camera is attached. The board is initialized for the currently attached camera when R2BrdOpen is called. If an application is written this way there is no need to call any of the functions in this chapter. However, some users may want to manage what cameras are attached and how the user switches between them using their own software. For this reason, these camera configuration functions are provided.

The normal flow for an application that wants to manage its own camera files is as follows:

```
R2BrdOpen
R2CamOpen
R2BrCamSetCur
// processing and acquisition
R2CamClose
R2BrdClose
```

If using more than one camera:

```
R2BrdOpen
R2CamOpen      // open camera 0
R2CamOpen      // open camera 1
R2BrCamSetCur  // configure for camera 0
// processing and acquisition
R2BrCamSetCur  // configure for camera 1
// processing and acquisition
R2CamClose     // close camera 0
R2CamClose     // close camera 1
R2BrdClose
```

24.2 R2CamOpen

Prototype R2RC R2CamOpen(RdRn *Board*, PCHAR *CamName*, PR2CAM **pCam*)

Description Allocates a camera configuration object, opens a camera configuration file, and loads the file into the object.

Parameters

Board

Handle to board.

CamName

The name of the camera file to open. Do not include the path. The camera file must be in the configuration directory (see the SysReg application). For example: "Pn9700.cam".

**pCam*

A pointer to a camera object. The memory to hold the object is allocated in this function.

Returns

R2_OK	If successful.
R2_NO_CNFDIR_REG_KEY	The configuration directory entry is missing in the register (run SysReg).
R2_BAD_PATH	Error building the path to the camera file.
R2_BAD_STRUCT	Error calculating the size of the camera structure.
R2_BAD_ALLOC	Cannot allocate memory to perform open.
R2_BAD_CNF_FILE	Error opening or reading configuration file.
R2_BAD_HEADER	Error in configuration file header. This could include an error in one or more of the following items: signature (Road Runner/R3 configuration), endian test (endian model is unknown), revision (camera revision is incompatible), size (size of file is not the same as written) and CRC (byte error in file).
R2_BAD_BINR	Error reading configuration item from file.
R2_BAD_CNFA	Error inserting configuration item into camera object.

Comments This function allocates memory to hold a camera configuration object, locates the given camera configuration file in the configuration directory, checks the file for errors, then loads the camera configuration parameters into the camera object. The camera object is used to tell the system how to set up the board to acquire from a particular camera. Use the program CamVert to edit camera configuration files.

The resulting camera object can be passed to other functions such as R2BrdCamSet-Cur.

The resources allocated by the function must be freed by calling R2CamClose.

24.3 R2CamInquire

Prototype R2RC R2CamInquire(RdRn *Board*, PR2CAM *pCam*, BFU32 *Member*, PBFU32 *pVal*)

Description Returns information about the given camera.

Parameters

Board

Handle to board.

pCam

Camera whose characteristics are requested.

Member

Characteristic to find the value of. The member must be one of:

R2CamInqXSize - width of image in pixels.
R2CamInqYSize - height of image in lines.
R2CamInqFormat - image format.
R2CamInqPixBitDepth - depth of pixel in bits, as acquired to host.
R2CamInqBytesPerPix - depth of pixel in bytes, as acquired to host.
R2CamInqBytesPerPixDisplay - depth of pixel in bytes, as acquired to display.
R2CamInqBitsPerSequence - depth of multi-channel pixel in bits, as acquired to host.
R2CamInqBitsPerSequenceDisplay - depth of multi-channel pixel in bits, as acquired to display.
R2CamInqHostFrameSize - total size of image in bytes, as acquired to host.
R2CamInqDisplayFrameSize - total size of image in bytes, as acquired to display.
R2CamInqHostFrameWidth - width of image in bytes, as acquired to host.
R2CamInqDisplayFrameWidth - width of image in bytes, as acquired to display.
R2CamInqAqTimeout - number of milliseconds to wait before acquisition command times out.
R2CamInqCamType - camera type.
R2CamInqControlType - type of camera control accessible through API.

pVal

Pointer to value of the characteristic.

Returns

R2_OK	If successful.
R2_BAD_INQ_PARAM	Unknown *Member* parameter.

Non-zero	On error.

Comments

This function is used to inquire about characteristics of a camera. For 8-bit cameras, the parameter R2CamInqHostFrameSize is equal to R2CamInqDisplayFrameSize. The parameter only differs for pixel depths greater than eight.

24.4 R2CamClose

Prototype R2RC R2CamClose(RdRn *Board*, PR2CAM *pCam*)

Description Frees resources used by a camera object.

Parameters *Board*

Handle to board.

pCam

Camera object.

Returns

R2_OK	In all cases.

Comments This function frees all resources used by a camera object.

24.5 R2CamAqTimeoutSet

Prototype R2RC R2CamAqTimeoutSet(RdRn *Board*, PR2CAM *pCam*, BFU32 *Timeout*)

Description Sets the acquisition timeout variable in the given camera configuration.

Parameters

Board

Handle to board.

pCam

Camera whose characteristics are requested.

Timeout

New value for timeout, in milliseconds.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function sets the timeout value for an earlier configuration

Road Runner/R3 Interrupt Signals

Chapter 25

25.1 Introduction

The purpose of the Signal Function calls is to make hardware interrupts available to user-level applications in a simple and efficient set of functions. In fact, under Windows NT, there is no way for a user-level application to get direct notification of a hardware interrupt. Only kernel-level drivers can contain interrupt service routines (ISR). Most customers do not want to deal with the complications of writing ISRs anyway, so BitFlow has come up with this signaling system.

Basically, a signal is a wrapper around a Windows NT semaphore object. The signal has a state and a queue. Every time an interrupt occurs, the signal's state changes. The nice thing about signals is that you can wait for their state to change, without using any CPU cycles. This is what makes them so efficient. This means that you can have one thread processing images while another is waiting for the next image to be completely DMAed. The thread that is waiting for the signal consumes very little CPU time, thus making most of the CPU available for processing.

The way these functions are used is that you start by creating a signal with the R2SignalCreate. There are a number of different interrupts that the signal can wait for, and it is in this function that you specify the one you want. Once the signal is created, your application waits for the interrupt with either the R2SignalWait or the R2SignalWaitNext function. The difference being the R2SignalWait function uses a signals queue. If an interrupt has occurred before this function is called, then this function will return immediately. It will continue to return immediately until there are no more interrupts in the queue. The R2SignalWaitNext function always waits for the next interrupt after being called, regardless of how many have occurred since it was last called.

Signals can be used in a single thread application, but whenever one of the wait functions is called, execution will be blocked until the interrupt occurs. Because this situation can potentially hang a process, a timeout parameter is provided for all of the wait functions. If you need an application to process data while waiting on an image to be captured, create a separate thread to call the wait function. Meanwhile, another thread can be processing with most of the CPUs cycles. A thread waiting on a signal can be cancelled with the function R2ThreadCancel. This causes the waiting thread to return from the wait function with an error code indicating that it has been cancelled.

The following is an example of how these functions can be called:

```
Int ImageIn = 0
main ()
{
   R2BrdOpen// open board
   R2SignalCreate// create the signal for EOF
   CreateThread(EOFThread)// create a thread
   while (KeepProcessing)// main processing loop
   {
         // here we loop until we have an image
         while (ImageIn !=1)
         {
               // secondary processing
         }

         // now we have an image so process it

         ImageIn = 0             // reset variable

         // primary image processing

         }
   }

   // clean up
   R2SignalCancel// cancel signal kill thread
   R2SignalFree// free signal resources
   R2BrdClose// close board
}

// thread to watch for end of frame
EOFThread()
{
   loop
   {
         rv = R2SignalWait       // wait for signal
         if(rv == CANCELED)      // was returned value
cancel?
               exit loop         // yes, kill this thread else
         else
               ImageIn = 1       // no, set new image flag
   }
}
```

25.2 R2SignalCreate

Prototype R2RC R2SignalCreate(RdRn *Board*, BFU32 *Type*, PR2SIGNAL *pSignal*)

Description Creates a signal that will allow user level thread to be notified of hardware interrupts.

Parameters

Board

Handle to board.

Type

Type of interrupt signal to create. Must be one of the following:

R2IntTypeHW - hardware exception.
R2IntTypeFIFO - video FIFO overflow.
R2IntTypeDMADone -Non chaining DMA operation complete.
R2IntTypeEOD - End of DMA for current frame. Occurs when the last pixel has been DMAed into memory. Users will create this signal ninety percent of the time.
R2IntTypeCTab - interrupt column set in CTAB for current line.

pSignal

Pointer to R2SIGNAL structure.

Returns

R2_OK	If successful.
R2_BAD_SEMAPHORE	Could not get semaphore object from operating system.
R2_BAD_ALLOC	Could not allocate memory for signal.

Comments This function creates a signal object that is used to receive interrupt notifications from the Road Runner/R3. The R2SignalWaitXXXX function takes a signal as a parameter. These functions efficiently wait for an interrupt of the given type to occur. The best way to use a signal is to create a separate thread that calls one of the R2SignalWaitXXXX functions. This thread will consume minimal CPU cycles until the interrupt occurs. When the interrupt occurs, the signal is notified and the R2SignalWaitXXXX functions will return. The thread can then take appropriate action, calling whatever functions are necessary and/or send messages to the main application thread.

This signaling system is the only way to handle Road Runner/R3 interrupts at the user application level

More than one signal can be created for the same interrupt on the same board. Also, more than one process and/or thread can wait for the same interrupt. When the interrupt occurs, all of the signals will be notified in the order they were created. The signal created by this function receives interrupt notification only from the Road Runner/R3 passed to this function in the *Board* parameter.

The most frequently used signal is *Type* = R2IntTypeEOD. The R2AqSetup function automatically sets the interrupt bit in the last quad in the QTab of the current image. This signal will be notified when the last pixel of the image has been DMAed into memory, and the current acquisition is done in the case of a snap or freeze.

The signal created by this function must be cleaned up by calling R2SignalFree.

25.3 R2SignalWait

Prototype

R2RC R2SignalWait(RdRn *Board*, PR2SIGNAL *pSignal*, BFU32 *TimeOut*, PBFU32 *pNumInts*)

Description

Efficiently waits for an interrupt to occur. Returns immediately if one has occurred since the function was last called.

Parameters

Board

Handle to board.

pSignal

Pointer to R2SIGNAL previously created by R2SignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout.

pNumInts

Pointer to a BFU32. When the function returns, it will contain the number of interrupts (the interrupt queue) that have occurred since this function was last called.

Returns

R2_OK	Interrupt has occurred.
R2_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
R2_SIGNAL_CANCEL	Signal was canceled by another thread (see R2SignalCancel).
R2_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
R2_WAIT_FAILED	Operating system killed the signal.

Comments

This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function will return immediately if the interrupt has occurred since the function was last called with this signal. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

When this function returns, the *pNumInts* parameter will contain the number of interrupts that have occurred since this function was last called. This is essentially an interrupt queue. Normally this will be one. However, if one or more interrupts have occurred, the function will return immediately and this variable will indicate the number that has occurred. This parameter is useful in determining if frames were missed.

This function will continue to return immediately, reducing the number of interrupts in the queue each time until every interrupt that has occurred has been acknowledged, and the queue is empty.

To wait for the next interrupt and ignore any previous interrupts, use R2SignalWait-Next.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

25.4 R2SignalNextWait

Prototype R2RC R2SignalNextWait(RdRn *Board*, PR2SIGNAL *pSignal*, BFU32 *TimeOut*)

Description Like R2SignalWait, this function waits efficiently for an interrupt. However, this version always ignores any interrupts that might have occurred since it was called last, and just waits for the next interrupt.

Parameters

Board

Handle to board.

pSignal

Pointer to R2SIGNAL previously created by R2SignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout

Returns

R2_OK	Interrupt has occurred.
R2_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
R2_SIGNAL_CANCEL	Signal was canceled by another thread (see R2SignalCancel).
R2_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
R2_WAIT_FAILED	Operating system killed the signal.

Comments This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function waits for the next interrupt, regardless of the number of interrupts in the signal's queue. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

Use R2SignalWait if you need a function that will return immediately if an interrupt has already occurred.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

25.5 R2SignalCancel

Prototype R2RC R2SignalCancel(RdRn *Board*, PR2SIGNAL *pSignal*)

Description Cancels a signal, any R2SignalWaitXXX function will return with a value of R2_SIGNAL_CANCEL.

Parameters

Board

Handle to board.

pSignal

Pointer to R2SIGNAL to cancel.

Returns

R2_OK	If successful.
R2_BAD_SIGNAL	Signal does not exist.

Comments This function will cancel a signal. It is primarily used by multi-threaded applications where one thread is waiting (with one of the R2SignalWaitXXXX functions) for a signal. Another thread can cancel the signal with this function, thereby waking up the waiting thread. When the waiting thread wakes up and the R2SignalWaitXXXX function returns, the return value can be examined. If the return value is R2_SIGNAL_CANCEL, the thread knows that the signal it was waiting for was canceled, and it can take appropriate action.

This function is usually used as a clean way for the main application thread to tell waiting threads to kill themselves.

Canceling a signal with this function will interfere with its internal interrupt counts. Therefore, this function should only be called when synchronization with the interrupt is no longer important and/or the signal is going to be destroyed.

25.6 R2SignalQueueSize

Prototype R2RC R2SignalQueueSize(RdRn *Board*, PR2SIGNAL *pSignal*, PBFU32 *pNumInts*)

Description Reports the current number of interrupts in a signal's queue.

Parameters

Board

Handle to board.

pSignal

Pointer to R2SIGNAL whose queue is to be investigated.

pNumInts

Pointer to BFU32. When the function returns *pNumInts*, it will contain the number of interrupts in the signal's queue.

Returns

R2_OK	If successful.
R2_BAD_SIGNAL	Signal does not exist.

Comments This function returns the number of interrupts in a signal's queue. This function is useful for testing to see if any interrupts have come in for a given signal, when you do not want to call one of the R2SignalWaitXXX functions. This function can be called any time.

25.7 R2SignalQueueClear

Prototype R2RC R2SignalQueueClear(RdRn *Board*, PR2SIGNAL *pSignal*)

Description Clears interrupts from a single queue.

Parameters

Board

Handle to board.

pSignal

Pointer to R2SIGNAL whose queue is to be investigated.

Returns

R2_OK	If successful.
R2_BAD_SIGNAL	Signal does not exist.
R2_WAIT_FAILED	Error clearing queue.

Comments This function clears all of the interrupts for a given signal's queue. This allows a thread to wait for the next interrupt to occur. This function is usually only used to re-synchronize a signal to the current state of acquisition (i.e., ignore any interrupts that have occurred in the past) before calling R2SignalWait. To always wait for the next interrupt, call R2SignalWaitNext.

25.8 R2SignalFree

Prototype R2RC R2SignalFree(RdRn *Board*, PR2SIGNAL *pSignal*)

Description Frees all resources used by a signal.

Parameters

Board

Handle to board.

pSignal

Pointer to R2SIGNAL whose queue is to be investigated.

Returns

R2_OK	In all cases.

Comments This function frees the resources used by a signal and removes it from the list of signals that get interrupt notification.

Road Runner/R3 Camera Control Functions

Chapter 26

26.1 Introduction

The Road Runner/R3 is capable of controlling the timing of the currently attached camera. On many cameras, the Road Runner/R3 can control the exposure time and the line/frame rate. The part of the board that is programmed to set these parameters is the Road Runner/R3's control tables (CTABs). The CTABs are set to some default exposure time and line/frame rate when the board is first initialized. The value of these parameters is set in the camera configuration file. If something besides the default is needed, the CTABs can be modified by the application to drive the camera to some other value. Normally, the CTABs are modified using the R2CTabXXXX functions. However, using these CTAB functions requires detailed knowledge of the camera, and how it is interfaces to the Road Runner/R3 and its CTABs.

The purpose of the camera control functions is to make programming the exposure time and line/frame rate easy and simple. These functions take exposure time and line/frame rate in real world units. Internally, these functions calculate how to modify the CTABs based on the requested parameters.

Currently, functions are only supplied for line scan cameras. There are two groups of functions to control the exposure and line rate of a line scan camera. The first set is used when the camera is in a free running mode. Free running means that all of the timing is generated on the Road Runner/R3. The second set is used when an encoder (line trigger) is being used. In this case, the horizontal timing is in one shot mode, which means the one line is acquired every time the encoder input is asserted. As a result, the encoder controls the line rate, and the board can only control the exposure time.

For each group, there is a function to get the current exposure and line rate, a function that returns the range that is possible for the current camera, and a function to set line rate and exposure.

These functions do work for all cameras from all manufacturers. Please see the software release notes for specific cameras that are supported.

26.2 R2CamLineScanTimingFreeRunGetRange

Prototype

R2RC R2CamLineScanTimingFreeRunGetRange(RdRn *Board*, BFU32 *PixelClockFrequency*, PBFU32 *pMinExposureTime*, PBFU32 *pMaxExposureTime*, PBFU32 *pMinLineRate*, PBFU32 *pMaxLineRate*)

Description

Gets the minimum and maximum exposure period and line rate for a line scan camera in free running mode.

Parameters

Board

Handle to board.

PixelClockFrequency

Pixel clock frequency coming onto the board. If zero, we assume the Board is generating a master clock and the pixel clock is derived from that or that the pixel clock is a constant for the given camera type.

pMinExposureTime

Minimum exposure time in nanoseconds.

pMaxExposureTime

Maximum exposure time in nanoseconds.

pMinLineRate

Minimum line rate in lines per second.

pMaxLineRate

Maximum line rate in lines per second.

Returns

R2_OK	If successful.
R2_CAM_BAD_CAM_TYPE	Current camera is not a line scan camera.
R2_CAM_NO_CONTROL	Camera control not supported for the current camera.
R2_CAM_BAD_HWIN	The horizontal active window for the current camera does not make sense.
R2_CAM_BAD_FREQ	The pixel clock frequency is invalid.

Comments

This function calculates the minimum and maximum line rate and exposure time for currently selected line scan camera in free-running mode. The function makes the rate and exposure range calculations independently, that is, you may not be able to

achieve camera control over the entire range of both rate and exposure. In other words, you may not be able to achieve all possible combinations of rate and exposure, even if both parameters are kept with the ranges returned by this function.

If min = max = 0 for a particular parameter, then that parameter cannot be controlled.

26.3 R2CamLineScanTimingFreeRunSet

Prototype R2RC R2CamLineScanTimingFreeRunSet(RdRn *Board*, BFU32 *PixelClockFrequency*, PBFU32 *pExposureTime*, PBFU32 *pLineRate*, BFU32 *Priority*)

Description Sets the exposure period and line rate for a line scan camera in free running mode.

Parameters

Board

Handle to board.

PixelClockFrequency

Pixel clock frequency coming onto the board. If zero, we assume the board is generating a master clock and the pixel clock is derived from that or that the pixel clock is a constant for the given camera type.

pExposureTime

Exposure time desire in nanoseconds. If zero, exposure time not important.

pLineRate

Line rate in lines per second. If zero, line rate not important.

Priority

This variable indicates which parameter exposure or rate is more important in the event that board is not capable of producing both the rate and the exposure desired. This parameter must be one of the following values:

- R2CamExposurePriority - attempt to set both the rate and exposure. If not possible, then set the given exposure and adjust the rate.
- R2CamRatePriority - attempt to set both the rate and exposure. If not possible, then set the given rate and adjust the exposure.
- R2CamFailOnNotExact - if both rate and exposure cannot be achieved, then return an error.
- R2CamExposureUnimportant - set the exposure based on the given line rate.
- R2CamRateUnimportant - set the line rate based on the given exposure.

Returns

R2_OK	If successful.
R2_CAM_BAD_CAM_TYPE	Current camera is not a line scan camera.
R2_CAM_NO_CONTROL	Camera control not supported for the current camera.

R2_CAM_BAD_HWIN	The horizontal active window for the current camera does not make sense.
R2_CAM_BAD_FREQ	The pixel clock frequency is invalid.
R2_CAM_BAD_EXP	The desired exposure is not possible.
R2_CAM_BAD_RATE	The desired line rate is not possible.
R2_CAM_BAD_PRIORITY	The priority parameter is not valid.
R2_CAM_NOT_EXACT	Both the exact rate and exposure cannot be achieved.

Comments

This function sets the exposure time and the line rate for the current line scan camera in free running mode.

26.4 R2CamLineScanTimingFreeRunGet

Prototype R2RC R2CamLineScanTimingFreeRunGet(RdRn *Board*, BFU32 *PixelClockFrequency*, PBFU32 *pExposureTime*, PBFU32 *pLineRate*)

Description Gets the current exposure period and line rate for a line scan camera in free running mode.

Parameters

Board

Handle to board.

PixelClockFrequency

Pixel clock frequency coming onto the board. If zero, we assume the board is generating a master clock and the pixel clock is derived from that or that the pixel clock is a constant for the given camera type.

pExposureTime

The current exposure time that the board is set for.

pLineRate

The current line rate that the board is set for.

Returns

R2_OK	If successful.
R2_CAM_BAD_CAM_TYPE	Current camera is not a line scan camera.
R2_CAM_NO_CONTROL	Camera control not supported for current camera.
R2_CAM_BAD_HWIN	The horizontal active window for the current camera does not make sense.
R2_CAM_BAD_FREQ	The pixel clock frequency is invalid.
R2_CAM_BAD_RESET	The CTAB's horizontal reset does not make sense.
R2_CAM_BAD_EXPSR	The exposure period cannot be calculated.

Comments This function gets the current exposure time and the line rate for the current line scan camera in free running mode.

26.5 R2CamLineScanTimingOneShotGetRange

Prototype

R2RC R2CamLineScanTimingOneShotGetRange(RdRn *Board*, BFU32 *PixelClock-Frequency*,
PBFU32 *pMinExposureTime*, PBFU32 *pMaxExposureTime*)

Description

Gets the minimum and maximum of exposure period for a line scan camera in one shot mode.

Parameters

Board

Handle to board.

PixelClockFrequency

Pixel clock frequency coming onto the board. If zero, we assume the board is generating a master clock and the pixel clock is derived from that or that the pixel clock is a constant for the given camera type.

pMinExposureTime

Minimum exposure time in nanoseconds.

pMaxExposureTime

Maximum exposure time in nanoseconds.

Returns

R2_OK	If successful.
R2_CAM_BAD_CAM_TYPE	Current camera is not a line scan camera.
R2_CAM_NO_CONTROL	Camera control not supported for current camera.
R2_CAM_BAD_HWIN	The horizontal active window for the current camera does not make sense.
R2_CAM_BAD_FREQ	The pixel clock frequency is invalid.

Comments

This function calculates the minimum and maximum exposure time for currently selected line scan camera in one shot mode.

26.6 R2CamLineScanTimingOneShotSet

Prototype R2RC R2CamLineScanTimingOneShotSet(RdRn *Board*, BFU32 *PixelClockFrequency*, PBFU32 *pExposureTime*)

Description Sets the exposure for a line scan camera in one shot mode.

Parameters

Board

Handle to board.

PixelClockFrequency

Pixel clock frequency coming onto the board. If zero, we assume the board is generating a master clock and the pixel clock is derived from that or that the pixel clock is a constant for the given camera type.

pExposureTime

The resulting exposure time that the board is set for.

Returns

R2_OK	If successful
R2_CAM_BAD_CAM_TYPE	Current camera is not a line scan camera.
R2_CAM_NO_CONTROL	Camera control not supported for current camera.
R2_CAM_BAD_HWIN	The horizontal active window for the current camera does not make sense.
R2_CAM_BAD_FREQ	The pixel clock frequency is invalid.
R2_CAM_BAD_EXP	The desired exposure is not possible.

Comments This function sets the exposure time the current line scan camera in one shot mode

26.7 R2CamLineScanTimingOneShotGet

Prototype R2RC R2CamLineScanTimingOneShotGet(RdRn *Board*, BFU32 *PixelClockFrequency*, PBFU32 *pExposureTime*)

Description Gets the current exposure period for a line scan camera in one shot mode.Parameters

Parameters

Board

Handle to board.

PixelClockFrequency

Pixel clock frequency coming onto the board. If zero, we assume the board is generating a master clock and the pixel clock is derived from that or that the pixel clock is a constant for the given camera type.

pExposureTime

The resulting exposure time that the board is set for.

Returns

R2_OK	If successful.
R2_CAM_BAD_CAM_TYPE	Current camera is not a line scan camera.
R2_CAM_NO_CONTROL	Camera control not supported for current camera.
R2_CAM_BAD_HWIN	The horizontal active window for the current camera does not make sense.
R2_CAM_BAD_FREQ	The pixel clock frequency is invalid.
R2_CAM_BAD_RESET	The CTAB's horizontal reset does not make sense.
R2_CAM_BAD_EXPSR	The exposure period cannot be calculated.

Comments This function gets the current line scan camera's exposure in one shot mode.

Road Runner/R3 LUTS

Chapter 27

27.1 Introduction

These functions allow an application full control over the Look Up Tables (LUTs) on the Road Runner/R3. The LutPeek and LutPoke functions are fairly inefficient and should only be used in the case of modifying a small number of entries. For accessing a larger number of entries or the entire LUT, create an array on the host and use the R2LutWrite and R2LutRead functions. To create a “ramp” function in the LUTs, use the R2LutRamp function.

27.2 R2LutPeek

Prototype BFU32 R2LutPeek(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*)

Description Reads a single LUT value.

Parameters

Board

Road Runner/R3 board ID.

Mode

LUT mode:

R2Lut8Bit - peek an 8-bit value out of an 8-bit LUT.
R2Lut12Bit - peek a 16-bit value out of a 12-bit LUT.
R2Lut16Bit - peek a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

R2LutBank0 - peek LUT bank 0.
R2LutBank1 - peek LUT bank 1.

Lane

One or more LUT lanes ORed together:

R2LutLane0 - peek LUT lane 0.
R2LutLane1- peek LUT lane 1.
R2LutLane2 - peek LUT lane 2.
R2LutLane3 - peek LUT lane 3.

Addr

LUT address.

Returns The LUT value.

Comments LUT definitions are declared in R2Reg.h.

27.3 R2LutPoke

Prototype R2RC R2LutPoke(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *Value*)

Description Writes a single LUT value to one or more LUT lanes.

Parameters

Board

Road Runner/R3 board ID.

Mode

LUT mode:

R2Lut8Bit - poke an 8-bit value into an 8-bit LUT.
R2Lut12Bit - poke a 16-bit value into a 12-bit LUT.
R2Lut16Bit - poke a 16-bit value into a 16-bit LUT.

Bank

LUT bank:

R2LutBank0 - poke LUT bank 0.
R2LutBank1 - poke LUT bank 1.

Lane

One or more LUT lanes ORed together:

R2LutLane0 - poke LUT lane 0.
R2LutLane1- poke LUT lane 1.
R2LutLane2 - poke LUT lane 2.
R2LutLane3 - poke LUT lane 3.

Addr

LUT address.

Value

LUT write value.

Returns

R2_OK	Function succeeded.
R2_NO_BIG_LUTS	Road Runner/R3 board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.

R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_POKE_ERR	LUT poke failed.

Comments LUT definitions are declared in R2Reg.h.

27.4 R2LutRead

Prototype R2RC R2LutRead(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *NumEntries*, PBFVOID *pDest*)

Description Reads a LUT.

Parameters

Board

Road Runner/R3 board ID.

Mode

LUT mode:

R2Lut8Bit - read an 8-bit value out of an 8-bit LUT.
R2Lut12Bit - read a 16-bit value out of a 12-bit LUT.
R2Lut16Bit - read a 16-bit value out of a 16-bit LUT.

Bank

LUT bank:

R2LutBank0 - read LUT bank 0.
R2LutBank1 - read LUT bank 1.

Lane

One or more LUT lanes ORed together:

R2LutLane0 - read LUT lane 0.
R2LutLane1- read LUT lane 1.
R2LutLane2 - read LUT lane 2.
R2LutLane3 - read LUT lane 3.

Addr

LUT address.

NumEntries

Number of LUT entries to read.

pDest

Storage for LUT entries. The size of the destination is based on the LUT *mode* being used and the *NunEntries*. If R2Lut8Bit LUT mode is being used, memory should be allocated for *NumEntries* of the BFU8 data type (a byte). Both R2Lut12Bit and R2Lut16Bit modes should use *NumEntries* of the BFU16 data type (a word). A example of the usage would be:

```
BFU8 LUT8[256];  // R2Lut8Bit LUT mode.
BFU16 LUT16[4096]; // R2Lut12Bit and R2Lut16Bit LUT modes.
```

Returns

R2_OK	Function succeeded.
R2_NO_BIG_LUTS	Road Runner/R3 board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_TOO_MANY_LANES	Only one lane may be read at a time.
R2_LUT_READ_ERR	LUT read failed.

Comments

LUT definitions are declared in R2Reg.h.

27.5 R2LutWrite

Prototype R2RC R2LutWrite(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *NumEntries*, PBFVOID *pSource*)

Description Writes a LUT.

Parameters

Board

Road Runner/R3 board ID.

Mode

LUT mode:

R2Lut8Bit - write an 8-bit value into an 8-bit LUT.
R2Lut12Bit - write a 16-bit value into a 12-bit LUT.
R2Lut16Bit - write a 16-bit value into a 16-bit LUT.

Bank

LUT bank:

R2LutBank0 - write LUT bank 0.
R2LutBank1 - write LUT bank 1.

Lane

One or more LUT lanes ORed together:

R2LutLane0 - write LUT lane 0.
R2LutLane1- write LUT lane 1.
R2LutLane2 - write LUT lane 2.
R2LutLane3 - write LUT lane 3.

Addr

LUT address.

NumEntries

Number of LUT entries to write.

pSource

Storage LUT data. The size of the source is based on the LUT *mode* being used and the *NunEntries*. If R2Lut8Bit LUT mode is being used, memory should be allocated for *NumEntries* of the BFU8 data type (a byte). Both R2Lut12Bit and R2Lut16Bit modes should use *NumEntries* of the BFU16 data type (a word). A example of the usage would be:

```
BFU8 LUT8[256];      // R2Lut8Bit LUT mode.
BFU16 LUT16[4096];   // R2Lut12Bit and R2Lut16Bit LUT modes.
```

Returns

R2_OK	Function succeeded.
R2_NO_BIG_LUTS	Road Runner/R3 board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_WRITE_ERR	LUT write failed.

Comments

LUT definitions are declared in R2Reg.h.

27.6 R2LutFill

Prototype R2RC R2LutFill(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *NumEntries*, BFU32 *Val*)

Description Fills a LUT with a constant.

Parameters

Board

Road Runner/R3 board ID.

Mode

LUT mode:

R2Lut8Bit - write an 8-bit value into an 8-bit LUT.
R2Lut12Bit - write a 16-bit value into a 12-bit LUT.
R2Lut16Bit - write a 16-bit value into a 16-bit LUT.

Bank

LUT bank:

R2LutBank0 - write LUT bank 0.
R2LutBank1 - write LUT bank 1.

Lane

One or more LUT lanes ORed together:

R2LutLane0 - write LUT lane 0.
R2LutLane1- write LUT lane 1.
R2LutLane2 - write LUT lane 2.
R2LutLane3 - write LUT lane 3.

Addr

LUT address.

NumEntries

Number of LUT entries to fill.

Val

Fill value.

Returns

R2_OK	Function succeeded.

R2_NO_BIG_LUTS	Road Runner/R3 board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_FILL_ERR	LUT fill failed.

Comments

LUT definitions are declared in R2Reg.h.

27.7 R2LutRamp

Prototype R2RC R2LutRamp(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *StartAddr*, BFU32 *EndAddr*, BFU32 *StartVal*, BFU32 *EndVal*)

Description Fills a LUT with a ramp.

Parameters

Board

Road Runner/R3 board ID.

Mode

LUT mode:

R2Lut8Bit - writes an 8-bit value into an 8-bit LUT.
R2Lut12Bit - writes a 16-bit value into a 12-bit LUT.
R2Lut16Bit - writes a 16-bit value into a 16-bit LUT.

Bank

LUT bank:

R2LutBank0 - write LUT bank 0.
R2LutBank1 - write LUT bank 1.

Lane

One or more LUT lanes ORed together:

R2LutLane0 - write LUT lane 0.
R2LutLane1- write LUT lane 1.
R2LutLane2 - write LUT lane 2.
R2LutLane3 - write LUT lane 3.

StartAddr

LUT start address.

EndAddr

LUT end address.

StartVal

LUT start value.

EndVal

LUT end value.

Returns

R2_OK	Function succeeded.
R2_NO_BIG_LUTS	Road Runner/R3 board doesn't support 16-bit LUTs.
R2_BAD_BANK	Illegal LUT bank.
R2_BAD_LUT_ADDR	Illegal LUT address.
R2_LUT_RAMP_ERR	LUT ramp failed.

Comments

LUT definitions are declared in R2Reg.h.

27.8 R2LutMax

Prototype R2RC R2LutMax(RdRn *Board*, LutModePtr *LutSizePtr*)

Description Gets the maximum LUT size.

Parameters *Board*

Road Runner/R3 board ID.

LutSizePtr

Pointer to LUT size storage.

Returns

R2_OK	Function succeeded.

Comments This function gets the minimum LUT size.

Road Runner/R3 Mid-Level Control Functions

Chapter 28

28.1 Introduction

These functions are used to control the board at a lower level than the R2AqCommand function. In general, an application should not need to use these functions unless special circumstances exist. These functions talk directly to the hardware and make no assumptions about how the rest of the board is set up. Generally, it is a bad idea to mix high-level functions and these mid-level functions.

28.2 R2ConAqCommand

Prototype R2RC R2ConAqCommand(RdRn *Board*, BFU32 *Command*)

Description Sends an acquisition command to the board.

Parameters *Board*

Handle to board.

Command

Command send to board:

R2ConSnap - snap one frame.
R2ConGrab - start continuous acquisition.
R2ConFreeze - stop continuous acquisition at the end of the current frame.
R2ConAbort - stop acquisition immediately.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	Unknown *Command* parameter.

Comments This function sends an acquisition command directly to the hardware. This is a low-level function and makes no assumptions about the state of the rest of the board.

This command returns immediately.

28.3 R2ConAqStatus

Prototype R2RC R2ConAqStatus(RdRn *Board*, PBFU32 *pStatus*)

Description Gets the current acquisition state of the board.

Parameters *Board*

Handle to board.

pStatus

Pointer to BFU32. When this function returns it contains the status of the board. The status will be one of the following:

R2ConFreeze - the board is not acquiring.
R2ConSnap - the board is currently acquiring one frame.
R2ConGrab - the board is currently in continuous acquisition mode.

Returns

R2_OK	If successful.

Comments This function returns the current acquisition status of the board.

28.4 R2ConAqMode

Prototype R2RC R2ConAqMode(RdRn *Board*, BFU32 *DestType*)

Description For a given destination type, this function sets the board's acquisition MUX registers, based on the current camera type.

Parameters *Board*

Handle to board.

DestType

Type of acquisition to prepare for:

R2DMABitmap - the destination buffer to be used for display.
R2DMADataMem - the destination buffer needs to contain raw data.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	Unknown *DestType* parameter.

Comments This function sets up the Road Runner/R3's front end acquisition paths for acquiring to a display buffer or a raw data buffer. A display buffer is one that will be used for display on a monitor, and is 8 bits deep. A raw data buffer is one that the data has the same bit depth as the camera. The function sets up the MUX[A,B,C,D] registers as well as the TRIGAQWIDTH register. This function does not alter the state of the LUTs or QTABs.

This function has no effect for 8-bit cameras.

This function is normally called automatically by R2AqSetup, and does not need to be called explicitly by an application. An unpredictable result will occur if this function is called while the board is acquiring.

28.5 R2ConInt

Prototype R2RC R2ConInt(RdRn *Board*, BFU32 *IntType*, BFU32 *Action*)

Description Disables or enables individual hardware interrupts.

Parameters

Board

Handle to board.

IntType

Type of interrupt:

R2IntTypeHW - hardware exception.
R2IntTypeFIFO - video FIFO overflow.
R2IntTypeDMADone - Non chaining DMA operation complete.
R2IntTypeEOD - End of DMA for current frame. Occurs when the last pixel has been DMAed into memory. Users will create this signal ninety percent of the time.
R2IntTypeCTab - interrupt bit in VCTAB is set.

Action

Indicates whether to enable or disable the interrupt:

R2ConEnable - enable the interrupt.
R2ConDisable - disable the interrupt.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	Either the parameter *IntType* or *Action* is unknown.

Comments This function enables or disables the specified hardware interrupt for being invoked on the PCI bus. The driver always has an interrupt service (ISR) routine ready to handle any interrupts that come in. The driver's ISR will automatically reset the appropriate interrupt bits on the board when an interrupt occurs.

To receive notification of interrupts at the user application level, use the signaling system (see the R2SignalXXXX functions). These functions automatically enable the appropriate interrupt when the signal is created, so you do not have to call this function to use an interrupt with the signaling system. However, you can use this function to enable and disable interrupts, based on your application needs, without creating and destroying signals. As a general rule, you should disable any interrupts that you are not using. Every interrupt uses a certain amount of CPU time, even if no application is waiting for it.

When the board is initialized, by default, all interrupts are turned off.

28.6 R2ConDMACommand

Prototype R2RC R2ConDMACommand(RdRn *Board*, BFU32 *Command*, BFU32 *Mode*, BFU8 *Bank*)

Description Issues a DMA command to the board.

Parameters

Board

Handle to board.

Command

DMA command to issue:

R2ConDMAGo - start the DMA engine.
R2ConDMAAbort - immediately abort the current DMA operation.
R2ConDMAReset - reset the DMA engine.

Mode

Behavior of this function once the command is issued:

R2ConWait - wait for current command to be implemented.
R2ConAsync - return as soon as command is issued.

Bank

This parameter is the QTab bank to use for the next DMA operation. Must be 0 or 1. This parameter is ignored if the board is in Host QTab mode.

Returns

R2_OK	If successful.
R2_AQ_NOT_SETUP	R2AqSetup has not yet been called and the board is not ready for an acquisition command.
R2_BAD_CON_PARAM	Unknown command.
R2_TIMEOUT	Timeout waiting for command to complete. This is only possible if *Mode* = R2ConWait.

Comments This function sends a DMA command to the board. If the *Command* = R2ConDMAGo, this will tell the board to start DMAing. No data will actually be moved until an acquisition command has been issued. The best way to use the Road Runner/R3's DMA engine is to start the DMA and leave it on all the time. Then, control the time when data gets moved to the host by using the acquisition commands.

The command R2ConDMAAbort will stop DMA immediately. This is actually a faster way to stop moving data than aborting acquisition. If acquisition is aborted the board will still DMA until the FIFO is empty. If DMA is aborted, the board will be in an

unknown state. Call this function again with *Command* = R2ConDMAReset, to get the board ready for DMA again. Finally, call this function with *Command* = R2ConDMAGo, when ready to start DMAing again.

This function is automatically called by R2AqSetup, and does not normally need to be called by the applications.

If this function is called with *Mode* = R2ConWait, the function will not return until the command has been implemented. Table 28-1 lists what the function will wait for.

Table 28-1 Function Waiting

Command	Waits for…
R2ConDMAGo	DMA engine to get ready for DMA (this is a negligible amount of time).
R2ConDMAAbort	DMA engine to abort current transfer.
R2ConDMAReset	Does not wait.

When *Mode* = R2ConWait, this function does not efficiently wait, it polls the DMA registers for completion. This is necessary since none of the above conditions causes an interrupt. If the command has not completed before the DMA timeout has expired, the function will return with a timeout error. This DMA timeout is set using the SysReg utility, or with the function R2DMATimeout.

28.7 R2DMATimeout

Prototype R2RC R2DMATimeout(RdRn *Board*, BFU32 *Timeout*)

Description Sets the DMA timeout value for the given board.

Parameters *Board*

Handle to board.

Timeout

Number of milliseconds that must elapse before DMA operations time out.

Returns

R2_OK	In all cases.

Comments The DMA timeout is the number of milliseconds DMA operations will wait for a DMA command to complete. If more than this number of milliseconds have expired before the current DMA command has completed, then an error is set and the function will return with a timeout error.

Normally this value is the same for every board. The default value is set in the SysReg utility. However, this function can set the timeout to some other value for a given board. The function will only effect the timeout for the given Road Runner/R3.

The timeout value does not affect normal DMA of data into the host. Once the Road Runner/R3 is set up, the DMA is usually given the GO command. No actual data will move until an acquisition command is issued and the camera sends data to the board. The DMA timeout does not cover the interval between when the DMA engine is told to GO and before camera data has come in. This time can be covered by the acquisition time out value set in the camera configuration file. The DMA timeout covers the time between when a DMA command is issued and when the command has completed.

For example, when the DMA engine is told to go, the engine must report that it is going before the timeout has expired or a timeout error will result. This timeout is most useful for aborting or ending DMA. If the board is not programmed correctly or there is a camera problem, it is possible the DMA engine will never finish or abort. It is important to catch these conditions and raise an error.

28.8 R2DMAProgress

Prototype

R2RC R2DMAProgress(RdRn *Board*, PQTABHEAD *pRelQTabHead*, PBFU32 *pBytesAqed*)

Description

Returns the instantaneous number of bytes that have been DMAed so far in the current image.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to QTABHEAD structure already filled out.

pBytesAqed

Pointer to BFU32. When the function returns it will contain the number of bytes that have been DMAed.

Returns

R2_BAD_IOCTL	Error writing physical QTab to board. Check error stack for other errors.

Comments

This function returns the number of bytes of the current image that have been DMAed so far. The returned value is an instantaneous value that is accurate at the moment the board was checked. Since DMA can occur very quickly, the returned value may not be accurate for a very long. The value returned is also approximate, the granularity depends on the number of bytes transferred per quad (individual DMA instruction), which can vary from quad to quad. As a rule of thumb, this function usually is accurate to plus or minus one line's worth of bytes.

Calling this function in a loop is not a very efficient way to wait for a frame to be acquired. Use the signaling system instead. This function can be used to check the progress of the DMA and to find out how much new data is in the host memory.

28.9 R2LastLine

Prototype R2RC R2LastLine(RdRn *Board*, PBFU32 *pCurLine*)

Description Returns the line number of the last line in the frame.

Parameters

Board

Handle to board.

pCurLine

Pointer to the last line number.

Returns

R2_OK	In all cases.

Comments This functions returns the line number of the last line in the frame. The returned value is actually the Vertical CTAB counter value for the last line. If the camera being used is a line scan camers then this value will be equivalent to the line number. However, for area scan camera the start of the vertical active region will have to be subtracted from the returned value (usually the vertical active region starts at 0x1000).

This function is most useful when acquiring variable sized images and thus the frame size is unknown. This function will return the value from the last frame up until the end of the following frame. In other words, the value of the last line stays constant for the entire duration of the next frame. Once the next frame ends, then the last line is the value for that frame.

28.10 R2ShutDown

Prototype R2RC R2ShutDown(RdRn *Board*)

Description Aborts all DMA activity and acquisition on the board.

Parameters *Board*

Handle to board.

Returns

R2_OK	If successful.
R2_BAD_DMA0_STOP	Timeout waiting for DMA engine 0 to abort.
R2_BAD_DMA1_STOP	Timeout waiting for DMA engine 1 to abort.
R2_BAD_AQ_STOP	Timeout waiting for acquisition to abort.
R2_BAD_FIFO_RESET	Could not reset the FIFOs.

Comments This functions aborts all activity on the board. DMA is aborted. Acquisition is aborted. The FIFOs are reset. The board stops what it is currently doing and gets it ready for more acquisition. Normally this function does not need to be called.

28.11 R2ConSwTrigStat

Prototype R2RC R2ConSwTrigStat(RdRn *Board*, BFU32 *BFTrig,* PBFU32 *Status*)

Description Returns the status of the software trigger.

Parameters

Board

Handle to board.

BFTrig

The trigger to inquire about, can be one of the following:

BFTrigA - inquire about trigger A.
BFTrigB - inquire about trigger B.

Status

The status of the trigger can be one of the following:

BFTrigHigh - the software trigger was high.
BFTrigLow - the software trigger was low.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	Invalid BFTrig was passed to the function.

Comments This function returns the status of the software trigger at the moment that the function is called

28.12 R2ConHWTrigStat

Prototype R2RC R2ConHWTrigStat(RdRn *Board*, BFU32 *BFTrig,* PBFU32 *Status*)

Description Returns the status of the hardware trigger.

Parameters

Board

Handle to board.

BFTrig

The trigger to inquire about, can be one of the following:

BFTrigA - inquire about trigger A.
BFTrigB - inquire about trigger B.

Status

The status of the trigger can be one of the following:

BFTrigHigh - the software trigger was high.
BFTrigLow - the software trigger was low.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	Invalid BFTrig was passed to the function.

Comments This function returns the status of the hardware trigger at the moment that the function is called

28.13 R2ConFIFOReset

Prototype R2RC R2ConFIFOReset(RdRn *Board*)

Description Resets the FIFO and FIFO freeze register on the board.

Parameters *Board*

Handle to board.

Returns

R2_OK	In all cases.

Comments

28.14 R2ConCtabReset

Prototype R2RC R2ConCtabReset(RdRn *Board*)

Description Resets the control tables on the board.

Parameters *Board*

Handle to board.

Returns

R2_OK	In all cases.

Comments

28.15 R2ConVTrigModeSet

Prototype R2RC R2ConVTrigModeSet(RdRn *Board*, BFU32 *TrigMode*, BFU32 *TrigPolarity*)

Description Sets the trigger mode and polarities on the board.

Parameters *Board*

Handle to board.

TrigMode

The trigger mode can be one of the following:

BFTrigFreeRun - no trigger is used, board free runs.
BFTrigAqCmd - triggered acquire command mode, non-resettable cameras.
BFTrigAqCmdStartStop - start/stop triggered acquire mode, non-resettable cameras.
BFTrigOneShot - one shot mode, for asynchronously resettable cameras.
BFTrigOneShotSelfTriggered - self triggering one shot mode.
BFTrigContinuousData - for continuous data sources.
BFTrigOneShotStartAStopB - one shot mode, where acquisition starts with the rising edge on trigger A and ends acquisition with the rising edge of trigger B.
BFTrigOneShotStartAStopA - one shot mode, where acquisition starts with the rising edge on trigger A and ends acquisition with the falling edge of trigger A.

TrigPolarity

Polarity for trigger can be one of the following:

BFTrigAssertedHigh - TRIGGER is asserted on rising edge.
BFTrigAssertedLow - TRIGGER is asserted on falling edge.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	One of the parameters is not valid.

Comments This function works in conjunction with the camera configuration files. It is important to understand that not all cameras support all triggering modes. Usually a particular camera will only support one or two triggering modes. Furthermore, a different camera configuration file is usually needed for each triggering mode. For example, a camera will almost always have a free running configuration file, useful for set up and offline testing. A camera may also have a one shot file, which would be used in time-critical applications. You cannot usually put the board, set up by the free running file,

into one shot mode because the latter mode requires special triggering signals to be sent to the camera. However, you can put the board, set up by a one shot file, into self triggering one shot mode. This is useful for camera set up and system debugging.

The exception to the paragraph above is the triggered acquire command mode, which will work with all cameras. This mode is really no different than just issuing an acquisition command at a specific point in time in the future. When the board is in this mode, an acquisition command is written by the host but not latched. Basically, the board is armed but does not acquire any data. When the trigger is asserted the command latches. Once the command is latched, it acts as it normally does, that is, the board starts acquiring data at the start of the next frame from the camera. The only acquisition commands that are affected are snap and grab. The freeze and abort commands work normally, and do not need a trigger to be latched. The disadvantage of this mode is that it can add up to a frame time of latency to any trigger, because the camera's timing is not being reset.

All of the modes above, except the Start Stop trigger mode, work in conjunction with acquisition commands. The acquisition command should always be issued before any trigger. When the trigger does assert, the boards will be able to react immediately. When the board is in any of the Start Stop modes, the acquisition commands are actually issued by the hardware automatically. When using this command, there is no need to issue acquisition commands from the host.

If you want to find out what mode the board is in, call the function R2ConVTrigModeGet.

Note: This function only controls how the board is vertically triggered. Vertical triggers cause the board to acquire a whole frame from an area camera or a number of lines from a line scan camera.

Note: You must enable the connection of the external trigger with the function R2ConExTrigConnect. The software triggers are always available.

28.16 R2ConVTrigModeGet

Prototype R2RC R2ConVTrigModeGet(RdRn Board, PBFU32 *TrigMode*, PBFU32 *TrigPolarity*)

Description Gets the current trigger mode and polarities for the board.

Parameters *Board*

Handle to board.

TrigMode

Returns one of the following current trigger modes:

BFTrigFreeRun - no trigger is used, board free runs.
BFTrigAqCmd - triggered acquire command mode, non-resettable cameras.
BFTrigAqCmdStartStop - start/stop triggered acquire mode, non-resettable cameras.
BFTrigOneShot - one shot mode, for asynchronously resettable cameras.
BFTrigOneShotSelfTriggered – self triggering one shot mode.
BFTrigContinuousData – for continuous data sources.
BFTrigOneShotStartAStopB – one shot mode, where acquisition starts with the rising edge on trigger A and ends acquisition with the rising edge of trigger B.
BFTrigOneShotStartAStopA – one shot mode, where acquisition starts with the rising edge on trigger A and ends acquisition with the falling edge of trigger A.

TrigPolarity

Returns one of the following polarities for the trigger:

BFTrigAssertedHigh - trigger is asserted on rising edge.
BFTrigAssertedLow - trigger is asserted on falling edge.

Returns

R2_OK	In all cases.

Comments This function returns the current state of the trigger circuitry for the board.

28.17 R2ConHTrigModeSet

Prototype R2RC R2ConHTrigModeSet(RdRn *Board*, BFU32 *EncMode*, BFU32 *EncPolarity*, BFU32 *EncSelect*)

Description Sets the horizontal trigger mode and polarities for the acquisition engine.

Parameters

Board

Handle to board.

EncMode

The horizontal triggering mode:

BFEncFreeRun – no line trigger is used, board free runs.
BFEncOneShot – horizontal one shot mode, every line needs a line trigger.
BFEncOneShotSelfTriggered – self triggering one shot mode..

EncPolarity

Polarity for all line triggers:

BFEncAssertedHigh - line triggers are asserted on rising edge.
BFEncAssertedLow - line triggers are asserted on falling edge.

EncSelect

Type of encoder:

BFEncA - Encoder A is active and B is disabled on the R2 or R3.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	One of the parameters is not valid or the particular combination of parameters is not possible.

Comments

28.18 R2ConHTrigModeGet

Prototype R2RC R2ConHTrigModeGet(RdRn *Board*, PBFU32 *EndMode*, PBFU32 *EncPolarity*, PBFU32 *EncSelect*)

Description Gets the current horizontal encoder mode and polarity of the encoder.

Parameters

Board

Handle to board.

EncMode

Returns the current encoder mode:

BFEncFreeRun – no line trigger is used, board free runs.
BFEncOneShot – horizontal one shot mode, every line needs a line trigger.

EncPolarity

Returns the current polarity for the encoder:

BFEncAssertedHigh - trigger A is asserted on rising edge.
BFEncAssertedLow - trigger A is asserted on falling edge.

EncSelect

Returns the encoder input type:

BFEncA - Encoder A is active and B is disabled on the R2 or R3.
BFEncUnknown - Could not determine the encoder input type.

Returns

R2_OK	In all cases.

Comments

28.19 R2ConExTrigConnect

Prototype R2RC R2ConExTrigConnect(RdRn *Board*, BFU32 *Mode*)

Description Connects or disconnect the external hardware trigger to the acquisition circuitry.

Parameters

Board

Handle to board.

Mode

Status of connection:

BFExTrigConnect - connect the trigger.
BFExTrigDisconnect - disconnect the trigger.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	The Mode parameter is not valid.

Comments This function connects the external hardware trigger to the acquisition circuitry. This function lets you turn on or off the effect of external triggers without altering other settings on the board, as well as whatever machinery is driving the trigger signal.

28.20 R2ConExTrigStatus

Prototype R2RC R2ConExTrigStatus(RdRn *Board*, PBFU32 *Mode*)

Description Returns the status of the hardware trigger to the acquisition circuitry.

Parameters *Board*

Handle to board.

Mode

Returns the status of connection:

BFExTrigConnect - connect the trigger.
BFExTrigDisconnect - disconnect the trigger.

Returns

R2_OK	In all cases.

Comments This function returns the status of the connection between the external hardware trigger and the acquisition circuitry.

28.21 R2ConGPOutSet

Prototype R2RC R2ConGPOutSet(RdRn *Board*, BFU32 *Mask*, BFU32 *Value*)

Description Sets the bits on the General Purpose Output registers.

Parameters

Board

Handle to board.

Mask

Type of GPOut (GPOuts may be or'ed together - BFGPOut0|BFGPOut1):

BFGPOut0 - Set the the value of GPOut0.
BFGPOut1 - Set the the value of GPOut1.
BFGPOut2 - Set the the value of GPOut2.
BFGPOut3 - Set the the value of GPOut3.
BFGPOut4 - Set the the value of GPOut4.
BFGPOut5 - Set the the value of GPOut5.

Value

The value to set the bit(s) too. The value can be either a 0 or 1.

Returns

R2_OK	If successful.
R2_BAD_CON_PARAM	*Value* was not a 0 or a 1.
R2_BAD_GPOUT	A invalid *Mask* value was passed into the function.
R2_CON_GPOUT_BAD	The setting of the register value failed.

Comments The Camera Link and PMC boards only have three general purpose outputs, BFGPOut0, BFGPOut1, and BFGPOut2. All other R2/R3 boards use all general purpose outputs.

28.22 R2ConGPOutGet

Prototype R2RC R2ConGPOutGet(RdRn *Board*, PBFU32 *Value*)

Description Returns the value of the all the general purpose output bits.

Parameters

Board

Handle to board.

Value

A pointer to the value of the GPOut bits.

Returns

R2_OK	In all cases.

Comments The Camera Link and PMC boards only have three general purpose outputs, BFGPOut0, BFGPOut1, and BFGPOut2. This function returns the values for all the GPOuts on the board.

Each digit in *Value* represents a GPOut, GPOut0 being the right most digit. For example, if *Value* has a value of 0x0000003e all the GPOuts have a value of 1 except GPOut0 which has a value of 0.

Road Runner/R3 Data Control Functions

Chapter 29

29.1 Introduction

These functions control how the Road Runner/R3 interfaces to the camera. In general, the registers on the board are set up with the camera configuration file, and these functions need not be called. However, if an application needs to make minor changes to the boards setup, it is often easier to call these functions than to switch between camera modes.

These functions are essentially wrappers around register reads and writes. Some users may find it easier to write directly to the registers as it more closely imitates modifying the camera configuration files.

29.2 R2ConQTabBank

Prototype R2RC R2ConQTabBank(RdRn *Board*, BFU8 *Bank*)

Description Selects the QTab bank for the next transfer.

Parameters *Board*

Road Runner/R3 board ID.

Bank

Bank number:

R2QTabBank0 - set bank 0
R2QTabBank1 - set bank 1

Returns

R2_OK	Function succeeded.
R2_BAD_BANK	Illegal bank number.
R2_CON_QTAB_BANK_ERR	Bank switch failed.

Comments QTab bank selections are declared in R2TabRegister.h.

The indicated bank is activated when the host writes the DMA GO bit or when a new QTab sequence is initiated by an R2_DMA_QUAD_LOCAL_EOX (end-of-sequence) bit at the end of an in-progress QTab driven DMA transfer.

Register REG DBANK is the only register that may be modified by this function.

29.3 R2ConFreq

Prototype R2RC R2ConFreq(RdRn *Board*, BFU8 *Freq*)

Description Sets the Road Runner/R3 internal clock generator frequency.

Parameters *Board*

Road Runner/R3 board ID.

Freq

Internal clock frequency:

R2Freq000 - set to 0.0 MHz
R2Freq025 - set to 2.5 MHz
R2Freq050 - set to 5.0 MHz
R2Freq075 - set to 7.5 MHz
R2Freq100 - set to 10.0 MHz
R2Freq150 - set to 15.0 MHz
R2Freq200 - set to 20.0 MHz
R2Freq300 - set to 30.0 MHz

Returns

R2_OK	Function succeeded.
R2_BAD_FREQ	Illegal clock frequency.
R2_CON_FREQ_ERR	Frequency switch failed.

Comments Road Runner/R3 clock frequencies are declared in R2TabRegister.h.

R2ConFreq only sets the internal clock frequency. To enable the internal clock, register REG_CLKCON must be set to R2ClockInternal.

This clock does not affect board operation, it is only provided for cameras that need an external master clock.

Register REG_CFREQ is the only register that may be modified by this function.

29.4 R2ConGPOut

Prototype R2RC R2ConGPOut(RdRn *Board*, BFU8 *Value*)

Description Sets Road Runner/R3 GPOUT pins.

Parameters *Board*

Road Runner/R3 board ID.

Value

One or more output pins ORed together:

R2GPOut0 - general output pin 0
R2GPOut1 - general output pin 1
R2GPOut2 - general output pin 2
R2GPOut3 - general output pin 3 (CL only)
R2GPOut4 - general output pin 4 (CL only)
R2GPOut5 - general output pin 5 (CL only)

Returns

R2_OK	Function succeeded.
R2_BAD_GPOUT	Illegal general purpose output pin number.
R2_CON_GPOUT_ERR	Output pin set failed.

Comments The way this function works is if the GPOut is defined as a *Value*, it will be set high. The GPOuts that are not defined as a *Value* will be set low. Here is an example with an R2:

R2ConGPOut(Board, R2GPOut0|R2GPOut1);

With the above function call, GPOut 0 and 1 will be set to a 1 (high) and GPOut2 will be set to a 0 (low).

General output pin bit masks are declared in R2TabRegister.h.

R2ConGPOut enables the signal outputs for all the general purpose output pins.

Registers REG_ENGPOUT0, REG_ENGPOUT1, REG_ENGPOUT2, REG_GPOUT0, REG_GPOUT1 and REG_GPOUT2 may be modified by this function.

29.5 R2ConSwTrig

Prototype R2RC R2ConSwTrig(RdRn *Board*, BFU32 *Triggers*, BFU32 *AssertType*)

Description Trips software trigger.

Parameters *Board*

Road Runner/R3 board ID.

Triggers

One or more triggers ORed together:

R2TrigA - trip TRIGGERA
R2TrigB - trip TRIGGERB

AssertType

Type of trigger to cause:

R2TrigAssert - asserts the trigger
R2TrigDeassert - de-asserts the trigger

Returns

R2_OK	Function succeeded.
R2_BAD_SW_TRIG	Illegal software trigger.
R2_CON_SW_TRIG_ERR	Software trigger failed.

Comments Software trigger bit masks are declared in R2TabRegister.h.

Registers REG_SWTRIGA and REG_SWTRIGB may be modified by this function.

29.6 R2ConTrigAqCmd

Prototype R2RC R2ConTrigAqCmd(RdRn *Board*, BFU8 *Mode*)

Description Sets acquisition trigger mode.

Parameters *Board*

Road Runner/R3 board ID.

Mode

Trigger mode:

R2TrigAqCmdOn - enable acquire command triggering.
R2TrigAqCmdOff - disable acquire command triggering.

Returns

R2_OK	Function succeeded.
R2_BAD_TRIGAQ	Illegal trigger mode.
R2_CON_TRIG_AQ_ERR	Trigger mode set failed.

Comments Trigger modes are declared in R2TabRegister.h.

If R2ConTrigAqCmd is set to R2TrigAqCmdOn, R2ConTrigSel should be set to R2TrigRisingA.

Register REG_TRIGAQCMD may be modified by this function.

29.7 R2ConTrigSel

Prototype R2RC R2ConTrigSel(RdRn *Board*, BFU8 *Mode*)

Description Selects acquisition trigger method.

Parameters *Board*

Road Runner/R3 board ID.

Mode

Trigger control method:

R2TrigRisingA - trigger on rising edge of TRIGGERA.
R2TrigFallingA - trigger on falling edge of TRIGGERB.
R2TrigRisingAToB - start on rising edge of A, stop on rising edge of B.
R2TrigFallingAToB - start on falling edge of A, stop on falling edge of B.
R2TrigContinuous - continuously acquire data as long as TRIGGERA is asserted High.

Returns

R2_OK	Function succeeded.
R2_BAD_TRIG_SEL	Illegal trigger method.
R2_CON_TRIG_SEL_ERR	Trigger method selection failed.

Comments Trigger methods are declared in R2TabRegister.h.

Registers REG_TRIGCON and REG_TRIGPOL may be modified by this function.

29.8 R2ConVMode

Prototype R2RC R2ConVMode(RdRn *Board*, BFU8 *Mode*)

Description Sets vertical control mode.

Parameters *Board*

Road Runner/R3 board ID.

Mode

Vertical control mode:

R2FreeRun - allows the vertical counter to run free.
R2OneShot - after the vertical counter is reset, waits for a trigger.

Returns

R2_OK	Function succeeded.
R2_BAD_VSTOP	Illegal vertical control mode.
R2_CON_VMODE_ERR	Vertical control mode set error.

Comments Vertical control modes are declared in R2TabRegister.h.

Register REG_VSTOP may be modified by this function.

29.9 R2ConHMode

Prototype R2RC R2ConHMode(RdRn *Board*, BFU8 *Mode*)

Description Sets horizontal control mode.

Parameters *Board*

Road Runner/R3 board ID.

Mode

Horizontal control mode:

R2FreeRun - allows the horizontal counter to run free.
R2OneShot - after the horizontal counter is reset, wait for an encoder.

Returns

R2_OK	Function succeeded.
R2_BAD_HSTOP	Illegal horizontal control mode.
R2_CON_HMODE_ERR	Horizontal control mode set error.

Comments Horizontal control modes are declared in R2TabRegister.h.

Register REG_HSTOP may be modified by this function.

29.10 R2ConTapMirror

Prototype R2RC R2ConTapMirror(RdRn *Board*, BFU8 *Mode*)

Description Enables/disables scan reverse for taps 1 and 3.

Parameters *Board*

Road Runner/R3 board ID.

Mode

Mirror mode:

R2MirrorOn - enable scan reverse of taps 1 and 3.
R2MirrorOff - disable scan reverse of taps 1 and 3.

Returns

R2_OK	Function succeeded.
R2_CON_MIRROR_MODE	Illegal mirror mode.
R2_CON_MIRROR_ERR	Tap mirror mode set error.

Comments Tap mirror modes are declared in R2TabRegister.h.

Register REG_STRAIGHT may be modified by this function.

Road Runner/R3 Quad Table Functions

Chapter 30

30.1 Introduction

For almost all Road Runner/R3 applications there will be no need to call any of the functions in this chapter. These are considered mid-level functions and are generally only called indirectly by other, high level, functions. These functions are listed here in case some specialized programming of the Road Runner/R3 is required.

Quad Tables (or QTABS) are simple scatter gather DMA tables. A scatter gather DMA table is a list of instructions that tell the Road Runner/R3 how to DMA images to host. The name quad comes from the fact that each DMA instruction consists of four, 32-bit words: DMA source, DMA destination, DMA size, and a pointer to the next DMA instruction.

There are two types of QTABs: relative and physical. These differ only by the kind of memory the DMA destination describes. Relative QTABs describe relative or virtual memory address, which is typically all that your application sees.

Physical QTABs describe actual physical locations of memory. A relative QTab is created based on the current camera and the memory pointer that is handed to the create function. The physical QTab is built from this at the kernel level by locking down the virtual memory and calculating the physical addresses of this memory.

30.2 R2RelQTabCreate

Prototype

R2RC R2RelQTabCreate(RdRn *Board*, PR2CAM *pCam*, PBFVOID *pDest*, BFU32 *BufferSize*, BFS32 *Stride*, PQTABHEAD *pRelQTabHead*, BFU32 *DestType*, BFU32 *LutBank*, BFU32 *LutType*, BFU32 *Options*)

Description

Builds a relative QTab, used for acquisition from a given camera type to a host memory buffer. The relative QTab can then be converted to a physical QTab, which can be written to the board.

Parameters

Board

Handle to board.

pCam

Camera object of the type to build the QTab for.

pDest

A void pointer to the destination buffer.

BufferSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in pixels, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

pRelQTabHead

Pointer to an allocated QTABHEAD structure.

DestType

Type of destination memory:

R2DMADataMem - host memory
R2DMABitmap - display memory

LutBank

The LUT bank to pass the image through:

R2LutBank0 - LUT bank 0

R2LutBank1 - LUT bank 1
R2LutBypass - bypass LUTs

LutType

The mode of the LUT to use:

R2Lut8Bit - LUTs are programmed as 8 bits
R2Lut12Bit - LUTs are programmed as 12 bits
R2Lut16Bit - board has 16-bit LUTs (only on boards with 16-bit LUTs)

Options

Extra option for the last quad. Can be one or more of:

R2_DMA_OPT_INT - set interrupt bit in last quad.
R2_DMA_OPT_EOC - set EOC bit in last quad.

Returns

R2_OK	If successful.
R2_BAD_CNF	Error extracting information from the camera object.
R2_BAD_MODEL	The camera configuration contains a QTab model or format that is not understood by this version of the SDK. Or, the camera configuration is such that the relative QTab cannot be built.
R2_BAD_CON_PARAM	The *LutBank*, *LutType*, or *DestType* parameter is incorrect.
R2_BAD_ALLOC	Cannot allocate enough memory to build relative QTab.

Comments

This function builds a relative QTab for acquisition of a given camera type into a host memory buffer. The QTab is a table of scatter-gather DMA instructions that the Road Runner/R3 uses to continuously (and without host intervention) DMA camera data to the host memory. The relative QTab is the version of this table that is built with virtual addresses. These virtual addresses point to the destination buffer as addressed in the application's address space. The relative QTab must be passed to R2PhysQTabCreate to build a physical QTab. The physical QTab is the same as the relative QTab except that it contains physical addresses that can be used by the board as actual DMA destinations. The physical QTab is stored in the kernel and can be quickly copied to the board.

This is a mid-level function and should not be called except for custom programming of the Road Runner/R3. The high-level function R2AqSetup will call this function for you.

Depending on the camera, this function may take a moderate amount of time to calculate the relative QTab. This function should only be called once, for a given camera and destination. The relative QTab can be used repeatedly to acquire from the same camera type into the same memory buffer.

This function allocates memory to hold the relative QTab in the users address space. Call R2RelQTabFree to release this and other resources allocated in this function.

30.3 R2RelQTabCreateRoi

Prototype

R2RC R2RelQTabCreateRoi(RdRn *Board*, PR2CAM *pCam*, PBFVOID *pDest*, BFU32 *BufferSize*, BFS32 *Stride*, PQTABHEAD *pRelQTabHead*, BFU32 *DestType*, BFU32 *LutBank*, BFU32 *LutType*, BFU32 *Options*, BFU32 *DestX*, BFU32 *DestY*, BFU32 *DestDx*, BFU32 *DestDy*)

Description

Builds a relative QTab, used for acquisition from a given camera type to a Region Of Interest (ROI) in a host memory buffer. The relative QTab can then be converted to a physical QTab, which can be written to the board.

Parameters

Board

Handle to board.

pCam

Camera object of the type to build the QTab for.

pDest

A void pointer to the destination buffer.

BufferSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in pixels, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

pRelQTabHead

Pointer to an allocated QTABHEAD structure.

DestType

Type of destination memory:

R2DMADataMem - host memory
R2DMABitmap - display memory

LutBank

The LUT bank to pass the image through:

R2LutBank0 - LUT bank 0
R2LutBank1 - LUT bank 1
R2LutBypass - bypass LUTs

LutType

The mode of the LUT to use:

R2Lut8Bit - LUTs are programmed as 8 bits
R2Lut12Bit - LUTs are programmed as 12 bits
R2Lut16Bit - board has 16-bit LUTs (only on boards with 16-bit LUTs)

Options

Extra option for the last quad. Can be one or more of:

R2DMAOptInt - set interrupt bit in last quad.
R2DMAOptEOC - set EOC bit in last quad.

DestX

The X coordinate of the upper left hand pixel of the destination ROI.

DestY

The Y coordinate of the upper left hand pixel of the destination ROI.

DestDx

The width of the destination ROI in pixels.

DestDy

The height of the destination ROI in Pixels.

Returns

R2_OK	If successful.
R2_BAD_CNF	Error extracting information from the camera object.
R2_BAD_MODEL	The camera configuration contains a QTab model or format that is not understood by this version of the SDK. Or, the camera configuration is such that the relative QTab cannot be built.
R2_BAD_CON_PARAM	The *LutBank*, *LutType*, or *DestType* parameter is incorrect.
R2_BAD_ALLOC	Cannot allocate enough memory to build relative QTab.

Comments

This function builds a relative QTab for acquisition of a given camera type into a Region of Interest (ROI) host memory buffer. The QTab is a table of scatter-gather DMA instructions that the Road Runner/R3 uses to continuously (and without host intervention) DMA camera data to the host memory. The relative QTab is the version of this table that is built with virtual addresses. These virtual addresses point to the destination buffer as addressed in the application's address space. The relative QTab must be passed to R2PhysQTabCreate to build a physical QTab. The physical QTab is the same as the relative QTab except that it contains physical addresses that can be used by the board as actual DMA destinations. The physical QTab is stored in the kernel and can be quickly copied to the board.

All the pixels of the source image are DMAed. Pixels that fall out of the destination ROI are DMAed to "garbage can" buffer on the host. This garbage can is allocated by the kernel driver and not extra code is required to use this buffer. The parameter *DestX* must be on a 4 pixel boundary. This width of the ROI, *DestDx*, must be a multiple of 4 pixels.

This is a mid-level function and should not be called except for custom programming of the Road Runner/R3. The high-level function R2AqSetup will call this function for you.

Depending on the camera, this function may take a moderate amount of time to calculate the relative QTab. This function should only be called once, for a given camera and destination. The relative QTab can be used repeatedly to acquire from the same camera type into the same memory buffer.

This function allocates memory to hold the relative QTab in the users address space. Call R2RelQTabFree to release this and other resources allocated in this function.

30.4 R2RelQTabFree

Prototype R2RC R2RelQTabFree(RdRn *Board*, PQTABHEAD *pRelQTabHead*)

Description Frees resources allocated in R2RelQTabCreate.

Parameters *Board*

Handle to board.

pRelQTabHead

Pointer to QTABHEAD structure previously passed to R2RelQtabCreate.

Returns

R2_OK	In all cases.

Comments This function releases the memory used to hold the relative QTab and any other resources allocated in R2RelQTabCreate.

30.5 R2PhysQTabCreate

Prototype R2RC R2PhysQTabCreate(RdRn *Board*, PQTABHEAD *pRelQTabHead*)

Description Builds a physical QTab that backs a relative QTab that describes a destination buffer in host memory for a given camera type.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure that has been filled out in R2ReQtabCreate.

Returns

R2_OK	If successful.
R2_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.

Comments This function takes a relative QTab created in R2RelQTabCreate and builds a physical QTab (storage is allocated in this function). The physical QTab contains actual physical addresses of the destination buffer in memory. The physical addresses can be used by the Road Runner/R3 as DMA destinations. This function also locks the destination buffer into memory (prevents the operating system from swapping the memory to disk). The memory must be locked in order for the DMA request to be satisfied.

When this function returns, the QTABHEAD structure contains the handle to the newly created physical QTab.

The resulting physical QTab is stored in the driver. It can be copied to the board using R2PhysQTabWrite. Multiple physical QTABs can be built and live in the driver at the same time, each with it's own QTABHEAD structure. The physical QTab can be released with a call to R2PhysQTabFree.

This is a mid-level function and should not be called except for custom programming of the Road Runner/R3. The high-level function R2AqSetup will call this function for you.

30.6 R2PhysQTabWrite

Prototype R2RC R2PhysQTabWrite(RdRn *Board*, PQTABHEAD *pRelQTabHead*, BFU32 *Offset*)

Description Writes a physical QTab to a board.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure, containing a valid physical QTab.

Offset

The entry number to start writing the physical QTab. This can be any value between 0 and 32768. However, this value is usually the location of the first bank, starting at 0, or the second bank, starting at address 16384 (0x4000).

Returns

R2_OK	If successful.
R2_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.

Comments This function takes an already created physical QTab and copies it into the board's DMA quad tables. These tables are used to tell the Road Runner/R3's DMA engine where, and how many pixels to DMA to host.

There are two QTab banks on the board. The two banks allow for ping-pong type acquisition between two host buffers. Also, one bank can be currently used for DMA while the next is being loaded for a subsequent destination buffer. The Road Runner/R3 can switch banks, on-the-fly, when instructed to (see R2AqNextBankSet). The bank switch will take place just after the board DMAs the last quad in the current bank (the EOX of the last quad must be set, see R2RelQTabCreate). This function can also write to any location in the quad tables.

This is a mid-level function and should not be called except for custom programming of the Road Runner/R3. The high-level function R2AqSetup will call this function for you.

30.7 R2PhysQTabEOC

Prototype

R2RC R2PhysQTabEOC(RdRn *Board*, PQTABHEAD *pRelQTabHead*, BFU32 *Set*, BFU32 *Offset*)

Description

Sets or resets the EOC bit in the last quad of a physical QTab that is already written to the board. This function is used to gracefully end continuous DMA at the end of a frame.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure that contains a valid physical QTab.

Set

Whether to set the bit or not.

1 - set the EOC bit
0 - reset the EOC bit.

Offset

The entry number to start writing the physical QTab. This value must be either the location of the start of the first bank, address = 0, or the second bank, at address 16384 (0x4000).

Returns

R2_OK	If successful.
R2_BAD_IOCTL	Error creating physical QTab. Check error stack for other errors.

Comments

The QTABs on a Road Runner/R3 can be set up for continuous DMAing to host. Every image that is acquired is automatically sent to host, frame after frame, without host intervention. This process is facilitated by reuse of the same physical QTab on the board. QTABs can be built so that they loop back on themselves. With the QTab set up this way, every frame that comes in is DMAed. Normally, this process is stopped by stopping acquisition. However, the DMA can also be stopped. This function allows for DMA to be stopped after the last pixel of the current image is DMAed to memory.

The *Set* parameter should = 1 to stop the DMA after the current frame is completed. You can use the same QTab again, and in continuous mode, by calling this function with *Set* = 0. This will allow the current QTab to be used in continuous fashion again.

This is a mid-level function and should not be called except for custom programming of the Road Runner/R3. The high-level function R2AqSetup will call this function for you.

This function actually sets or resets the EOC (End of Chain) bit in the last quad of the current QTab.

30.8 R2PhysQTabFree

Prototype R2RC R2PhysQTabFree(RdRn *Board*, PQTABHEAD *pRelQTabHead*)

Description Frees the memory used to hold the physical QTab in the driver memory.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to a QTABHEAD structure that contains a valid physical QTab.

Returns

R2_OK	If successful.
R2_BAD_IOCTL	Error writing physical QTab to board. Check error stack for more error information.

Comments This function frees the driver level resources used to hold a physical QTab.

30.9 R2RelDisplayQTabCreate

Prototype

R2RC R2RelDisplayQTabCreate(RdRn *Board*, PR2CAM *pCam*, PBFVOID *pDest*, BFU32 *BufferSize*, BFS32 *Stride*, PQTABHEAD *pRelQTabHead*, BFU32 *DestType*, BFU32 *LutBank*, BFU32 *LutType*, BFU32 *Options*, BFU32 *SrcX*, BFU32 *SrcY*, BFU32 *SrcDX*, BFU32 *SrcDY*, BFU32 *DestX*, BFU32 *DestY*)

Description

Creates a relative QTab used to acquire into display memory (or other physical memory).

Parameters

Board

Handle to board.

pCam

Pointer to camera structure.

pDest

Pointer to destination memory buffer.

BufferSize

Size of buffer in bytes.

Stride

Number of bytes to move down one line in destination (if 0, *Stride* = xsize * pixel size, if < 0 fills bottom up).

pRelQTabHead

Pointer to QTab head structure.

DestType

Type of destination memory:

R2DMADataMem - host memory
R2DMABitmap - display memory

LutBank

The LUT bank to pass the image through:

R2LutBank0 - LUT bank 0
R2LutBank1 - LUT bank 1
R2LutBypass - bypass LUTs

LutType

The mode of the LUT to use:

R2Lut8Bit - LUTs are programmed as 8 bits
R2Lut12Bit - LUTs are programmed as 12 bits
R2Lut16Bit - board has 16-bit LUTs (only on boards with 16-bit LUTs)

Options

Extra option for the last quad. Can be one or more of:

R2DMAOptInt - set interrupt bit in last quad.
R2DMAOptEOC - set EOC bit in last quad.

SrcX

X coordinate of source ROI in pixels.

SrcY

Y coordinate of source ROI in pixels.

SrcDX

Width of source ROI in pixels (if 0 - use entire image, *DestX* must also be zero).

SrcDY

Height coordinate of source ROI in pixels (if 0 - use entire image, *DestY* must also be zero).

DestX

X coordinate of destination ROI in pixels (screen coordinates).

DestY

Y coordinate of destination ROI in pixels (screen coordinates).

Returns

R2_OK	If successful.
Non-zero	On error.

Comments

This function takes information about source and destination ROIs (assumes the destination ROI is in the VGA, thus *pDest* points to the beginning of VGA memory and stride is based on the current resolution) and builds a relative QTab (in host memory)

based on information for the camera structure. This function allocates memory to hold the quads (free it with R2RelQTabFree()). The QTABHEAD, however, should already be allocated.

The SrcX,SrcY,SrcDX,SrcDY indicate a region of interest (ROI) in the image to be transferred in terms of the camera coordinates (in pixels). *DestX* and *DestY* indicate where in the VGA the ROI is going to land, this should also be in terms of pixels. Based on these parameters and the *pDest* and *Stride* parameters, this function calculates the virtual coordinates of the destination of the first pixel in the image (this may end up being outside of the VGA memory image is bigger than the VGA). Then R2RelQTabCreateRoi() is called with these values. Pixels outside of the display window are DMAed to the garbage buffer. This includes pixels that may end up being offscreen.

NOTE: Because of the Road Runner/R3 architecture, the pixels outside of the ROI must still be DMAed off the board. This function DMAs these pixels to a garbage buffer allocated in the driver. This means there is no saving of PCI bandwidth by moving only a ROI to the host.

NOTE: *SrcX, SrcY, SrcDX, SrcDY, DestX*, and *DestY* must be expressed in terms of pixels and must all be on four pixel boundaries. This function can also be used to DMA to other devices where physical address of memory is known. For example, you can use this function to DMA to memory on a card on the PCI bus. If the memory is linear, set *SrcX, SrcY, SrcDX, SrcDY, DestX*, and *DestY* to zero.

30.10 R2PhysQTabEngage

Prototype R2RC R2PhysQTabEngage(RdRn *Board*, PQTABHEAD *pRelQTabHead*)

Description Sets the board up to use the given QTab for the next DMA operation. This function should be called for both host and board QTABs.

Parameters

Board

Handle to board.

pRelQTabHead

A pointer to a Relative QTab head structure. This should be the QTab for the host memory buffer that will acquired into when the next acquisition command occurs.

Returns

R2_OK	If successful.
BF_NULL_POINTER	Invalid *pRelQTabHead* pointer.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments This function engages the QTab *pRelQTabHead* so that the board will use this QTab for subsequent DMA operations. This is a mid level function which is not need it the high level functions (e.g. r2AqSetup) are being used to set up DMA.

This function is used when building QTABs using the R2RelQTabCreate functions. The normal order of function calls is as follows

R2RelQTabCreate
R2PhysQTabCreate
R2PhysQtabWrite
R2PhysQTabEngage
R2ConDMACommand

This function should be called for both board and host QTABs. The function will set the QTab up appropriately for whichever type of QTab is being used. This function must be called before DMA is started.

30.11 R2PhysQTabChainLink

Prototype R2RC R2PhysQTabChainLink(Bd *Board*, PPQTABHEAD *ChainArray*, BFU32 *NumInChain*)

Description Chains together a number of QTABs for sequential acquisition in host QTab mode.

Parameters

Board

Handle to board.

ChainArray

A array of pointers to QTABs which describe an set of buffers to be acquired into. The buffers will be filled in the order that their QTABs appear in this array.

NumInChain

The total number of QTab headers in the QTab chain array.

Returns

R2_OK	If successful.
BF_NULL_POINTER	*ChainArray* is NULL.
BF_NOT_CHAIN	*NumInChain* is not valid.
BF_BAD_CHAIN	Error walking *ChainArray*.

Comments This function effectively sets the board up for continuous acquisition into a sequence of host buffers. Each buffer in is DMAed into in turn, when the last buffer in the chain is filled, the board will DMA the next frame into the first buffer. In other words the chain describes a circular buffer.

The parameter *ChainArray* is an array of pointer to QTab headers. The QTab must already be created by calling R2RelQTabCreate and R2PhysQTabCreate. After this function returns successfully, the chain must be engaged by calling R2PhysQTAB-ChainEngage function. The normal calling sequence for this function would be as follows:

```
loop for all buffers
   R2RelQTabCreate

loop for all buffers
   R2PhysQTabCreate

R2PhysQTabChainLink
R2PhysQTabChainEngage
R2ConDMACommand
R2ConAqCommand
```

In the scenario above, no data will move until an acquisition command is sent to the board and the camera sends a frame to the board. Once data is flowing, the board will fill each buffer as the data comes in. Once the last buffer in the chain is filled, the board will continue starting with the first buffer. No host interaction is required for this process to work. The board will send a signal every frame (assuming R2RelQTabCreate was called with the R2DMAOptInt parameter) to tell your application when a frame is complete (use R2SignalWait).

NOTE: This function will only work for boards that support QTABs on the host, and will only work when board is in QTABs on the host mode.

30.12 R2PhysQTabChainBreak

Prototype R2RC R2PhysQTabChainBreak(RdRn *Board*, PPQTABHEAD *ChainArray*)

Description Release QTABs from a chain so that they can be reused to build a subsequent chain.

Parameters

Board

Handle to board.

ChainArray

A array of pointers to QTABs which has been already passed to R2PhysQTabChain-Create.

Returns

R2_OK	If successful.
BF_BAD_CHAIN	Error walking *ChainArray*.

Comments This function is used to release the QTABs that are used by a chain. When a chain is built the QTABs that make it up are modified for use in the chain. If these QTABs need to be used again, the chain must first be broken with this function. After this function is called, the individual QTABs can be use to build another chain, presumable in a different order.

There is no need to call this function during cleanup if the individual QTABs are not going to be used again.

30.13 R2PhysQTabChainEngage

Prototype

R2RC R2PhysQTabChainEngage(RdRn *Board*, PPQTABHEAD *ChainArray*, BFU32 *FrameNum*)

Description

Takes a successfully created chain and sets the board up to use it.

Parameters

Board

Handle to board.

ChainArray

This is an array of pointers to QTABs which describe an set of buffers to be acquired into. This parameter must first be passed to CiPhysQTabChainCreate.

FrameNum

The buffer number of the first frame in the chain to be acquired into.

Returns

R2_OK	If successful.
BF_NULL_POINTER	*ChainArray* is NULL.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments

After a chain is created using R2PhysQTabChainCreate, the chain must be engaged using this function in order for the board to use it. Creating a chain is not a real time operation and should be done off line. If more than one chain is required, they should all be created first, then this function can be used to select which chain will be acquired into first.

See R2PhysQTabChainCreate for more information.

30.14 R2PhysQTabChainProgress

Prototype R2RC R2PhysQTabChainProgress(RdRn *Board*, PPQTABHEAD *ChainArray*, PBFU32 *pFrameNum*, PBFU32 *pLineNum*)

Description Returns the line number and frame number of current image being DMAed.

Parameters

Board

Handle to board.

ChainArray

This is an array of pointers to QTABs which describe an set of buffers to be acquired into. This parameter must first be passed to CiPhysQTabChainCreate.

pFrameNum

Pointer to receive the number of the current frame being DMAed into.

pLineNum

Pointer to receive the number of the current line being DMAed.

Returns

R2_OK	If successful.
BF_BAD_PTAB	The chain is not valid.

Comments This function is used to check the progress of acquisition while the board is acquiring using a chain. The function will return both the line number and the frame number. This function is fairly computationally intensive and should not be called in a tight loop to monitor progress. This function is best used intermittently to check progress, for example, it can be interleaved with processing.

The best way to overlap acquisition and processing is to create a signal that waits for the quad done signal (end of frame interrupt). Once the signal is asserted, the CPU can freely process the entire frame.

If you need to monitor the boards progress using a tight loop, read the VCOUNT register. Reading a register uses much less CPU time. Even in this case, you should put a sleep in your loop to not overwhelm the board with register reads (which take precedence over DMAing). Again is it better to interleave processing and checking VCOUNT.

30.15 R2ChainSIPEnable

Prototype R2RC R2ChainSIPEnable(RdRn *Board*, PPQTABHEAD ChainArray)

Description Enables start-stop interrupt processing (SIP).

Parameters *Board*

Handle to board.

pRelQTabHead

Structure holding information about QTab.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments This function enables start-stop interrupt processing (SIP). This processing is used to reset the DMA engine in a kernel interrupt service routine.

When the board is in start-stop mode, the DMA is terminated before the frame is completely acquired. This termination leaves the DMA engine in an unknown state. The DMA engine must be reset and setup for the next host buffer before the next frame starts. Ordinarily this reset is performed by the application at the user level. However, in the case of a multi threaded application, the reset thread may not be able to reset the DMA engine before the beginning of the next frame (because of CPU load and thread priorities). To solve this problem the BitFlow SDK implements a DMA engine reset in the kernel level interrupt service routine. This code has higher priority than any user level threads. The latency and execution time of the SIP reset is minimized thus reducing the required minimum time between frames. This function turns on this functionality.

SIP only works (and is only required) when the board is in start-stop triggering mode (variable size image acquisition) and when a host QTab chain has been created and engaged. This function must be called before acquisition has started but after the QTab chain is created. This function enable the SIP resetting of the DMA engine, you must call BFChainSIPDisable to turn the SIP off. This SIP is based on the CTAB interrupt (vertical CTAB column IRQ) which must have an interrupt at location zero.

The example application Flow demonstrates usage of this function.

30.16 R2ChainSIPDisable

Prototype BFRC R2ChainSIPDisable(Bd *BoardId,* PPQTABHEAD ChainArray)

Description Disables Start-Stop Interrupt Processing mode.

Parameters *Board*

Board ID.

pRelQTabHead

Structure holding information about QTab.

Returns

BF_OK	Function succeeded.
Non-zero	Function failed.

Comments See R2ChainSIPEnable for details.

Road Runner/R3 Register Access

Chapter 31

31.1 Introduction

These functions allow an application to read and write directly to every bit on the Road Runner/R3. The individual bit names and their functions are described in full detail in the following section, the BitFlow Hardware Reference.

The most important functions here are R2RegPoke and R2RegPeek. The other functions are for more esoteric uses that treat registers as generic objects. Generally, these latter functions are not useful in customer applications.

31.2 R2RegPeek

Prototype R2RC R2RegPeek(RdRn *Board*, BFU32 *RegId*)

Description Reads a bit field out of a full 32-bit register.

Parameters *Board*

Road Runner/R3 board ID.

RegId

Register ID.

Returns The bit field value.

Comments Register IDs for partial registers (bit fields) and for full registers (32-bits) are intermingled into the same table. Function R2RegFlags may be used to distinguish the two register types (BFF_NSET for bit field registers, BFF_WSET for full registers).

Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

31.3 R2RegPeekWait

Prototype R2RC R2RegPeekWait(RdRn *Board*, BFU32 *RegId*, BFU32 *WaitValue*, BFU32 *WaitMilliseconds*)

Description Waits for a register value to match a desired value or return after a time-out.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

WaitValue

Register value to wait on.

WaitMilliseconds

Wait time-out in milliseconds.

Returns The desired register value or the register contents at time-out.

Comments The register value is immediately tested against the *WaitValue* and will return if the values match.

R2RegPeekWait spins on the register contents during the current threads entire time slice. Processor time is not shared back to other threads.

The minimum wait time will always be at least as long as the specified wait regardless of clock granularity.

The actual maximum wait time is shown in Table 31-1

Table 31-1 Maximum Wait Time

Formula	Conditions
T + 2G - T% - 1	when T%G != 0
T + G – 1	when T%G == 0

Where:

T = WaitMilliseconds
G = The clock granularity in milliseconds

Example

Enable triggered acquires and issues a snap command. Use R2RegPeekWait to wait for the triggered snap to start and then disable triggering.

31.4 R2RegPoke

Prototype R2RC R2RegPoke(RdRn *Board*, BFU32 *RegId*, BFU32 *RegValue*)

Description Writes a value to a register.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

RegValue

Value to write into the register.

Returns

R2_OK	If successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

Full (32-bit) registers are written directly. A read-modify-write is used to set bit field registers within a full register.

31.5 R2RegRMW

Prototype R2RC R2RegRMW(RdRn *Board*, BFU32 *RegId*, BFU32 *RegValue*, BFU32 *RegMask*)

Description Read-modify-write to a masked area within a register.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

RegValue

Value to write into the register.

RegMask

Mask bits defining register bits to be modified.

Returns

R2_OK	Function successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

The *RegValue* is masked with *RegMask* and the result is masked into the target register.

31.6 R2RegName

Prototype R2RC R2RegName(RdRn *Board*, BFU32 *RegId*, LPSTR *pRegName*, BFU32 *Size*)

Description Gets a register's name.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

pRegName

Pointer to register name storage.

Size

Register name array size.

Returns

R2_OK	Function successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

There are RegCount register IDs from ID 0 to ID RegCount - 1.

The register name will be truncated to fit the provided name buffer.

31.7 R2RegFlags

Prototype R2RC R2RegFlags(RdRn *Board*, BFU32 *RegId*, PBFU32 *FlagsPtr*)

Description Gets a register's type flags.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

FlagsPtr

Pointer to register flag storage.

Returns

R2_OK	Function successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

There are RegCount register IDs from ID 0 to ID RegCount - 1.

31.8 R2RegShift

Prototype R2RC R2RegShift(RdRn *Board*, BFU32 *RegId*, PBFU32 *ShiftPtr*)

Description Gets a register's bit field shift count.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

ShiftPtr

Pointer to shift count storage.

Returns

R2_OK	Function successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments The register shift field count is the number of bit positions a register value must be shifted to the left to fall within the register's bit field.

Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

Example

The following code uses R2RegObjectId, R2RegShift and R2RegMask to extract bit field REG_MUXC out of the 32-bit register, REG_CON0. The same result may be obtained by calling R2RegPeek(Board, REG_MUXC).

```
BFU32 ParentValue, ChildValue, Parent, Shift, Mask
R2RegObjectId(Board, REG_MUXC, &Parent)
R2RegShift(Board, REG_MUXC, &Shift)
R2RegMask(Board, REG_MUXC, &Mask)
ParentValue = R2RegPeek(Board, Parent)
ChildValue = (ParentValue & Mask) >> Shift
```

31.9 R2RegMask

Prototype R2RC R2RegMask(RdRn *Board*, BFU32 *RegId*, PBFU32 *MaskPtr*)

Description Gets a register's bit field mask.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

MaskPtr

Pointer to storage for the register's bit field mask.

Returns

R2_OK	Function successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments A bit field mask is used to access bit field data stored in a full 32-bit register.

Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

Example

The following code uses R2RegObjectId, R2RegShift and R2RegMask to extract bit field REG_MUXC out of the 32-bit register, REG_CON0. The same result may be obtained by calling R2RegPeek(Board, REG_MUXC).

```
BFU32 ParentValue, ChildValue, Parent, Shift, Mask
R2RegObjectId(Board, REG_MUXC, &Parent)
R2RegShift(Board, REG_MUXC, &Shift)
R2RegMask(Board, REG_MUXC, &Mask)
ParentValue = R2RegPeek(Board, Parent)
ChildValue = (ParentValue & Mask) >> Shift
```

31.10 R2RegObjectId

Prototype R2RC R2RegObjectId(RdRn *Board*, BFU32 *RegId*, PBFU32 *ObjectIdPtr*)

Description Gets a register's wide register object ID.

Parameters

Board

Road Runner/R3 board ID.

RegId

Register ID.

ObjectIDPtr

Pointer to wide register object ID storage.

Returns

R2_OK	Function successful.
R2_BAD_BIT_ID	Illegal register ID.

Comments A bit field register's parent ID is used with a register's bit field mask and bit field count to directly access bit fields within a full 32-bit register.

Register IDs are declared in BFTabRegister.h and/or R2TabRegister.h.

Example

The following code uses R2RegObjectId, R2RegShift and R2RegMask to extract bit field REG_MUXC out of the 32-bit register, REG_CON0. The same result may be obtained by calling R2RegPeek(Board, REG_MUXC).

```
BFU32 ParentValue, ChildValue, Parent, Shift, Mask
R2RegObjectId(Board, REG_MUXC, &Parent)
R2RegShift(Board, REG_MUXC, &Shift)
R2RegMask(Board, REG_MUXC, &Mask)
ParentValue = R2RegPeek(Board, Parent)
ChildValue = (ParentValue & Mask) >> Shift
```

Road Runner/R3 Control Tables

Chapter 32

32.1 Introduction

These functions allow an application to write directly to the control tables (CTAB) on the Road Runner/R3. Normally the CTABs are initialized from a camera configuration file. However, because the CTABs control things like frame rate and exposure time, an application may want to modify them on-the-fly.

32.2 Modifying CTABS from Software

It is often necessary for an application to modify the CTABs dynamically, based on user input or as the result of a calculation on image data. The SDK provides functionality for modifying the CTABs from an application. The primary function used to program the CTAB is "R2CtabFill()". This function writes a masked value to the Road Runner/R3's camera control table. The following sections illustrate the use of "R2CtabFill()" to modify CTABs.

32.3 Controlling the Exposure on a Dalsa Line Scan Camera

The PRIN (pixel reset) input to the Dalsa line scan camera can be used to reduce the exposure time. BitFlow's Dalsa specific cabling connects PRIN to the Road Runner/ R3's CT2 output. CT2 is controlled by the HCON2 entry in the HCTAB. When PRIN is low, the CCD is continuously discharged. The rising edge of PRIN starts the exposure period, which continues until the EXSYNC pulse moves the charges into the camera's output shift registers.

One of the advantages to using PRIN is that the line rate and the exposure period can be de-coupled. The length of the PRIN window determines the exposure, and the location of the horizontal reset (HSTART=1, HEND=1) determines the line rate. This method allows independent adjustment of line rate and exposure period.

The following example will program the Road Runner/R3 for a given exposure time based on a variable exp_time. This example also uses line_rate, which is the desired line rate and pix_clock, which is the pixel clock output from the camera.

```
/*
* Determine the boundaries of the Horizontal Window
*/

// find start of video
i = 0;
while (i < 0x2000 && R2CTabPeek(board,i,R2HCTabStart) = 0)
   i++;

if (i == 0x2000) return -1;// no horizontal video window found

vid_win_start = i;

// find end of video
while (i < 0x2000 && R2CTabPeek(board,i, R2HCTabStop) = 0)
   i++;

vid_win_end = i;
/*
* Calculate start horizontal reset position from
* line rate parameter:
* HRESET = 0x0800 + (pixel clock / 4) / line rate
* pixel clock must be determined from
* the camera's PVAL output
* but is usually half the MCLCK
*/

hreset_pos = 0x0800 + (pix_clock / 4) / line_rate;

if (hreset_pos >= 0x2000)
   return -1;              // line rate too slow
if (hreset_pos < vid_win_end)
   return -1;        // line rate too fast
```

```
/*
* Program HRESET
*/

//program in horizontal reset
R2CTabFill(board,hreset_pos,0x0001,R2HCTabStart,0xFFFF);
R2CTabFill(board,hreset_pos,0x0001,R2HCTabStop,0xFFFF);

/*
* Calculate start and end points exposure period
* exp_time is in seconds (must be float)
*/

exp_prd_size = (int) (exp_time * (float)(pix_clock / 4);

// check to see if exposure period is possible
if (exp_prd_size < 0)
   return -1;        // too short
if (hreset_pos - exp_prd_size < vid_win_start)
   return -1;        // too long

//make sure entire table is high
R2CTabFill(board,0x0000,0x2000,R2HCTabHCON2,0xFFFF); // High
//program PRIN low up to start of exposure period
R2CTabFill(board,vid_win_start,
   hreset_pos - vid_win_start -exp_prd_size,
   R2HCTabHCON2,0); // Low
```

32.4 Changing Exposure Time in Double Pulse Mode on the Pulnix TM-9700

Double pulse mode allows the Road Runner/R3 to control the TM-9700's exposure time. This example uses the file "PnTn9700E1.cam," which by default has a 32-line exposure time. It changes the VCTAB to achieve a one-line exposure time. The video starts a fixed number of lines after the second pulse ends. Therefore, the vertical acquisition window must be moved by the same amount as the second pulse. Note: the VINIT signal is active low, so our "pulses" are actually "0s" in the VCON0 table. The rest of this column must be filled with ones.

```
   /* make sure camera is configured for a one-shot mode */
   if (R2RegPeek(board,REG_VSTOP) != 1)
         return error;

   /* hold CTAB output so camera is not disturbed */
   R2RegPoke(board,REG_CTABHOLD,1);

 /* program new exposure time */
   /* program reset for 1 line exposure */
   /* This means 10 lines before second pulse, start first
pulse.
   As per Pulnix manual, 10 lines between pulses is equivalent
   to one line exposure time. */
   //Start first pulse at 0x0001
   R2CTabFill(board,0,0x0001,R2VCTAB_VCON0,0xFFFF);// High
   R2CTabFill(board,0,0x0002,R2VCTAB_VCON0,0);// Low

   //Second pulse starts at 0x000D
   R2CTabFill(board,0x0003,0x000A,R2VCTAB_VCON0,0xFFFF);//
High
   R2CTabFill(board,0x000D,0x0002,R2VCTAB_VCON0,0);// Low

   // rest of table is ones
   R2CTabFill(board,0x000F,0x1FF0,R2VCTAB_VCON0,0xFFFF);//
High
    // now move video window, always starts 0x0028 lines after
   // leading edge of second pulse
   R2CTabFill(board,0,0x2000, R2HCTabStart, 0);//Clear
   R2CTabFill(board,0,0x2000, R2HCTabStop, 0);//Clear
   R2CTabFill(board,0x0035,0x0001, R2HCTabStart, 0xFFFF);//
Video Start
   R2CTabFill(board,0x0219,0x0001, R2HCTabStop, 0xFFFF);//
Video End

   /* enable CTAB output */
   R2RegPoke(board,REG_CTABHOLD,0);

   /* start grab mode */
```

```
   R2RegPeek(board,REG_VSTOP);// allow Road Runner/R3 to free-
run
   rc = R2AqCommand(board,R2ConGrab,R2ConAsync,0);
   if (rc != R2_OK) return error;

   /* perform live operations */
```

32.5 Controlling Exposure Time in the One Shot Mode on Kodak Cameras

The Road Runner/R3 is able to control the exposure time of the one shot mode. Note: The camera must be in "Controlled" mode (MDE CD). The exposure is determined by how long the Road Runner/R3 drives the camera's exposure input low. This is programmed into the VCON0 bit of the VCTAB. The exposure time is determined by the number of lines during which VCON0 is low, multiplied by the camera's line time, which is 139.4 microseconds for the model 1.4i or 236.8 microseconds for the model 2.4i.

The example below cuts two-thirds off the exposure time contained in the camera configuration file. Since the exposure setting in the configuration files is approximately 100 microseconds, this will result in an exposure of about 33 microseconds.

```
/* make sure camera is configured for a one-shot mode */
if (R2RegPeek(board,REG_VSTOP) != 1)
   return error;

/* put camera into shutter "controlled" mode */
R2ConGPOut(board,0);

/* hold CTAB output so camera is not disturbed */
R2RegPoke(board,REG_CTABHOLD,1);

/* search VCTAB for shutter close */
sc = 1;
while (!R2CTabPeek(board,sc,R2VCTAB_VCON0))
   ++sc;

/* adjust exposure time */
nc = sc/3;           // any formula can be used here...
                     // ...this one cuts exposure by 1/3

/* make sure new exposure time is valid */
if (nc > 1900)       // don't hold open past 1900...
   nc = 1900;        // ...shutter needs time to close

/* program new exposure time */

// shutter open from 1 to nc:
R2CTabFill(board,1,nc-1,R2VCTAB_VCON0,0);

// shutter closed from nc to 2048:
R2CTabFill(board,nc,2048-nc,R2VCTAB_VCON0,0xFFFF);

/* enable CTAB output */
R2RegPoke(board,REG_CTABHOLD,0);

/* acquire a single frame with software trigger */
```

```
// start cycle with a software trigger
R2ConSwTrig(board,R2TrigA);

rc = R2AqCommand(board,R2ConSnap,R2ConWait,0);
if (rc != R2_OK)
   return error;
```

32.6 R2CTabPeek

Prototype BFU16 R2CTabPeek(RdRn *Board*, BFU32 *Index*, BFU16 *Mask*)

Description Reads a single masked value from the Road Runner/R3 Camera Control Table.

Parameters *Board*

Road Runner/R3 board ID.

Index

CTAB table offset.

0x0000 to 0x2000 for horizontal CTABs
0x0000 to 0x8000 for vertical CTABs

Mask

CTAB bit extraction mask.

R2CTab
R2HCTab
R2VCTab
R2HCTabHEnd
R2HCTabHStart
R2HCTabClamp
R2HCTabField
R2HCTabHStrobe
R2HCTabHCon0
R2HCTabHCon1
R2HCTabHCon2
R2VCTabVEnd
R2VCTabVLoad
R2VCTabVStart
R2VCTabIRQ
R2VCTabVStrobe
R2VCTabVCon0
R2VCTabVCon1
R2VCTabVCon2

Returns A single masked CTAB entry.

Comments CTAB bit masks and other definitions are declared in R2Reg.h.

Example

Check for a horizontal start bit in the horizontal CTAB at the horizontal load point location R2HLoad (0x800).

```
BFU32  HStartBit
HStartBit = R2CTabPeek(Board, R2HLoad, R2HCTabHStart)
```

32.7 R2CTabPoke

Prototype R2RC R2CTabPoke(RdRn *Board*, BFU32 *Index*, BFU16 *Mask*, BFU16 *Value*)

Description Writes a single masked value from the Road Runner/R3 Camera Control Table.

Parameters

Board

Road Runner/R3 board ID.

Index

CTAB table offset.

Mask

CTAB bit extraction mask (see R2CtabPeek).

Value

CTAB value.

Returns

R2_OK	Function succeeded.
R2_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R2_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R2_CTAB_POKE_ERR	CTAB poke failed.

Comments CTAB bit masks and other definitions are declared in R2Reg.h.

Example

Write a vertical stop bit in the vertical CTAB 0x100 lines beyond the vertical load location (R2VLoad = 0x1000).

```
R2CTabPoke(Board, R2VLoad + 0x100, R2VCTabVStop, R2CTabSet)
```

32.8 R2CTabRead

Prototype R2RC R2CTabRead(RdRn *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, PBFVOID *pDest*)

Description Reads masked CTAB values from the Road Runner/R3 Camera Control Table.

Parameters

Board

Road Runner/R3 board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to read.

Mask

CTAB bit extraction mask (see R2CtabPeek).

pDest

Pointer to CTAB table storage (32 bits per entry).

Returns

R2_OK	Function succeeded.
R2_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R2_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R2_CTAB_READ_ERR	CTAB read failed.

Comments CTAB bit masks and other definitions are declared in R2Reg.h.

Example

Read 0x100 values from the vertical CTAB starting at the vertical load point.

```
BFU32  VTab[0x100]
R2CTabRead(Board, R2VLoad, 0x100, R2VCTab, &VTab[0])
```

32.9 R2CTabWrite

Prototype R2RC R2CTabWrite(RdRn *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, PBFVOID *pSource*)

Description Writes masked CTAB values from the Road Runner/R3 Camera Control Table.

Parameters

Board

Road Runner/R3 board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to read.

Mask

CTAB bit extraction mask (see R2CtabPeek).

pSource

CTAB entries to write (32 bits per entry).

Returns

R2_OK	Function succeeded.
R2_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R2_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R2_CTAB_WRITE_ERR	CTAB write failed.

Comments CTAB bit masks and other definitions are declared in R2Reg.h.

Example

Write an entire horizontal CTAB to the Road Runner/R3.

```
BFU32  HTab[R2HCTABSIZE]
R2CTabWrite(Board, 0, R2HCTABSIZE, R2HCTab, &HTab[0])
```

32.10 R2CTabFill

Prototype R2RC R2CTabFill(RdRn *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, BFU16 *Value*)

Description Writes a masked CTAB fill value from the Road Runner/R3 Camera Control Table.

Parameters

Board

Road Runner/R3 board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to write.

Mask

CTAB bit extraction mask (see R2CtabPeek).

Value

CTAB fill value to write.

Returns

R2_OK	Function succeeded.
R2_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R2_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R2_CTAB_FILL_ERR	CTAB fill failed.

Comments CTAB bit masks and other definitions are declared in R2Reg.h.

Example

Clear the horizontal and vertical CTAB tables.

```
R2CTabFill(Board, 0, R2HCTABSIZE, R2HCTab, 0x0000)
R2CTabFill(Board, 0, R2VCTABSIZE, R2VCTab, 0x0000)
```

Road Runner Quad Tables

Chapter 33

33.1 Introduction

These functions allow an application to access the Road Runner's quad tables (QTab). There is almost never a situation where a user's application will need to write directly to the QTABs. Generally, application access to the QTABs is handled at a much higher level and is invisible to the user. If an application does need lower level control, then the R2RelQTab functions can be used. These functions are included here for completeness only.

Note: There is now memory for QTABs mounted on the R3 family. Therefore, none of these low level functions will work with an R3. The R3 uses host QTABs only.

33.2 R2QTabPeek

Prototype BFU32 R2QTabPeek(RdRn *Board*, BFU8 *Bank*, BFU32 *Index*)

Description Reads a single 32-bit value from the Road Runner Quad Table.

Parameters *Board*

Road Runner board ID.

Bank

QTab bank:

R2QTabBank0 - set bank 0
R2QTabBank1 - set bank 1

Index

QTab table offset.

Returns A single 32-bit QTab entry.

Comments QTab bank definitions are declared in R2Reg.h.

33.3 R2QTabPoke

Prototype R2RC R2QTabPoke(RdRn *Board*, BFU8 *Bank*, BFU32 *Index*, BFU32 *Value*)

Description Writes a single 32-bit value to the Road Runner Quad Table.

Parameters

Board

Road Runner board ID.

Bank

QTab bank:

R2QTabBank0 - set bank 0
R2QTabBank1 - set bank 1

Index

QTab table offset.

Value

QTab value.

Returns

R2_OK	Function succeeded.
R2_BAD_QTAB_BANK	Illegal QTab bank.
R2_BAD_QTAB_ADDR	Illegal QTab address.
R2_QTAB_POKE_ERR	QTab poke failed.

Comments QTab bank definitions are declared in R2Reg.h.

33.4 R2QTabRead

Prototype R2RC R2QTabRead(RdRn *Board*, BFU8 *Bank*, BFU32 *Index*, BFU32 *NumEntries*, PBFVOID *pDest*)

Description Reads 32-bit QTab values from the Road Runner Quad Table.

Parameters

Board

Road Runner board ID.

Bank

QTab bank:

R2QTabBank0 - set bank 0
R2QTabBank1 - set bank 1

Index

QTab table offset.

NumEntries

Number of QTab values to read.

pDest

Pointer to QTab table storage (32-bits per entry).

Returns

R2_OK	Function succeeded.
R2_BAD_QTAB_BANK	Illegal QTab bank.
R2_BAD_QTAB_ADDR	Illegal QTab address.
R2_QTAB_READ_ERR	QTab read failed.

Comments QTab bank definitions are declared in R2Reg.h.

33.5 R2QTabWrite

Prototype R2RC R2QTabWrite(RdRn *Board*, BFU8 *Bank*, BFU32 *Index*, BFU32 *NumEntries*, PBFVOID *pSource*)

Description Writes 32-bit QTab values to the Road Runner Quad Table.

Parameters

Board

Road Runner board ID.

Bank

QTab bank:

R2QTabBank0 - set bank 0
R2QTabBank1 - set bank 1

Index

QTab table offset.

NumEntries

Number of QTab values to write.

pSource

Pointer to QTab entries to write (32-bits per entry).

Returns

R2_OK	Function succeeded.
R2_BAD_QTAB_BANK	Illegal QTab bank.
R2_BAD_QTAB_ADDR	Illegal QTab address.
R2_QTAB_WRITE_ERR	QTab write failed.

Comments QTab bank definitions are declared in R2Reg.h.

33.6 R2QTabFill

Prototype R2RC R2QTabFill(RdRn *Board*, BFU8 *Bank*, BFU32 *Index*, BFU32 *NumEntries*, BFU32 *Value*)

Description Writes 32-bit QTab fill values to the Road Runner Quad Table.

Parameters

Board

Road Runner board ID.

Bank

QTab bank:

R2QTabBank0 - set bank 0
R2QTabBank1 - set bank 1

Index

QTab table offset.

NumEntries

Number of QTab values to write.

Value

QTab fill value to write.

Returns

R2_OK	Function succeeded.
R2_BAD_QTAB_BANK	Illegal QTab bank.
R2_BAD_QTAB_ADDR	Illegal QTab address.
R2_QTAB_FILL_ERR	QTab fill failed.

Comments QTab bank definitions are declared in R2Reg.h.

Road Runner/R3 Error Handling

Chapter 34

34.1 Introduction

All of the SDK functions will return error codes if the function does not execute correctly. High-level functions call mid level functions which in turn call low-level functions, which in turn call kernel functions. Because this chain can be so long and some of the errors may happen in low-level call, the return code of the top level function may not be very useful in debugging the root cause of the problem.

For these reasons, the SDK uses an error stack. As errors occur, they are put on the stack. The stack can be walked and the errors can be examined, one by one, to determine the root cause of the malfunction.

Errors can be sent to a number of destinations; each destination is independent of all other destinations. The possible destinations are the even viewer, a pop-up dialog, a debugger (if there is one running), and a file. Also, an error can cause the application to abort at either the user or the kernel level. Each destination can be turned on or off independently. Furthermore, each destination can be programmed to selectively filter out one or more particular errors.

34.2 R2ErrorXXXXXX

Prototype

R2RC R2ErrorEnableEvent(RdRn *Board*, BFU32 *Filter*) - enables event viewer errors

R2RC R2ErrorDisableEvent(RdRn *Board*, BFU32 *Filter*) - disable event viewer errors

R2RC R2ErrorEnableDebugger(RdRn *Board*, BFU32 *Filter*) - enables debugger errors

R2RC R2ErrorDisableDebugger(RdRn *Board*, BFU32 *Filter*) - disables debugger errors

R2RC R2ErrorEnableDialog(RdRn *Board*, BFU32 *Filter*) - enables dialog errors

R2RC R2ErrorDisableDialog(RdRn *Board*, BFU32 *Filter*) - disables dialog errors

R2RC R2ErrorEnableFile(RdRn *Board*, BFU32 *Filter*) - enables log file errors

R2RC R2ErrorDisableFile(RdRn *Board*, BFU32 *Filter*) - disables log file errors

R2RC R2ErrorEnableBreakUser(RdRn *Board*, BFU32 *Filter*) - enables user level break after error

R2RC R2ErrorDisableBreakUser(RdRn *Board*, BFU32 *Filter*) - disables user level break after error

R2RC R2ErrorEnableAll(RdRn *Board*, BFU32 *Filter*) - enables errors for all error devices

R2RC R2ErrorDisableAll(RdRn *Board*, BFU32 *Filter*) - disable errors for all error devices

Description

These functions controls which error messages are active for each error destination.

Parameters

Board

Board to set error destination(s) on.

Filter

Error(s) to enable or disable.

Returns

R2_OK	If successful.
Non-zero	On error.

Comments

Each of the functions are independent. Each error destination can have its own list of

enable and disable errors independent of all other destinations. If an error destination gets conflicting instructions (e.g., calling R2ErrorEnable(board,ErrorBug) followed by R2ErrorDisable(board,ErrorAll)), the last error function that is called takes precedence.

Filter can be one of the following options:

A single error number (see "R2TabError.h" and "BFTabError.h").

One or more error types ORed together. The error types are:

ErrorBug - outright bug in the code that must be fixed.
ErrorFatal - deep, deep trouble. Program execution cannot continue.
ErrorWarn - something is wrong but recovery is possible.
ErrorInfo - informational messages.

The error type that represents none of the errors.

ErrorNone

The error type that represents all errors.

ErrorAll

R64 Introduction

Chapter 35

35.1 Overview

The R64 architecture was first implemented on the R64. This was BitFlow's 5th generation frame grabber, so it is a highly evolved architecture. Subsequently the same architecture was used on the following frame grabbers, the Karbon family, the Neon and the Alta family. The R64 API can be use to access any of these board families, although we encourage you to use the Ci API, which is guaranteed to support all current and future board families.

The R64 architecture is based around BitFlow's Flow-Thru architecture. This design allows data to be DMAed into host memory with the least amount of latency and virtually no CPU involvement. The camera link boards interface with base, medium and full configuration Camera Link cameras as well as two base configuration cameras (dual base). To accommodate full configuration cameras, the boards internal data paths are all 64 bits wide. Pixels from cameras that output less than 64 bits are packed into 64 bit words. DMA operations are always 64 bits. Output from multi-tapped camera is reformatted on-the-fly into raster order by the boards formatting circuitry. For more detailed hardware information, please refer to the appropriate Hardware Reference Manuals. Figure 35-1 shows the block diagram of the R64 architecture.

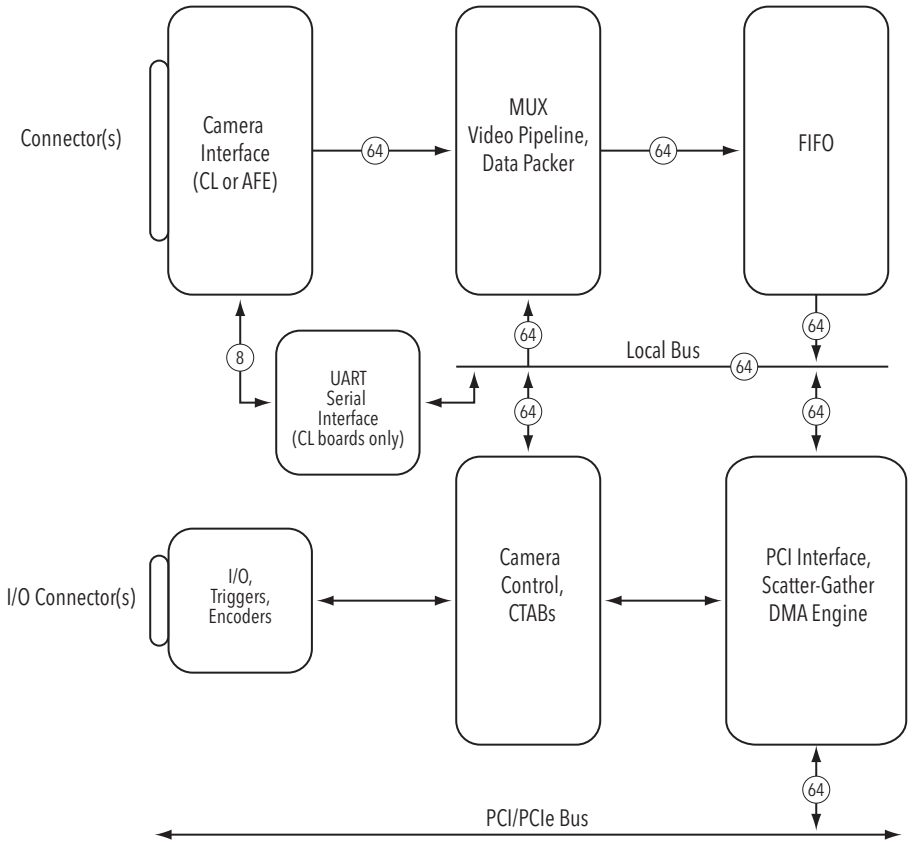

Figure 35-1 R64 architecture Block Diagram

R64 System Open and Initialization

Chapter 36

36.1 Introduction

The functions described in this chapter are quite simple; the idea is to find the board or boards that you want to work with, then open and optionally initialize them. When you are finished, close the system up, thus cleaning up all resources allocated in the open function.

A normal program would uses these functions, in this order:

```
R64SysBoardFindByXXXX
R64BrdOpen

// acquisition and processing

R64BrdClose
```

If you want to open two boards, the flow would be as follows:

```
R64SysBoardFindByXXXX   // find board 0
R64BrdOpen   // open board 0
R64SysBoardFindByXXXX       // find board 1
R64BrdOpen   // open board 1

// acquisition and processing

R64BrdClose   // close board 0
R64BrdClose   // close board 1
```

The board find functions are used to make sure that you are opening the correct board in a multi-board system. If you have only one board, then the call is trivial.

Note: There is currently only one board find function, R64SysBoardFindByNum.

The handle return by the function R64BrdOpen is used in all subsequent function calls. If you are using two or mode boards, open each board and store each handle in a separate variable. Whenever you want to talk to board X, pass the handle for board X to the function.

There is no need to call R64BrdOpen more than once per process per board. Because this function takes a fair amount of CPU time and allocated resources, we discourage users from repeatedly calling R64BrdOpen and the R64BrdClose in a loop. We recommend opening the board once, when the application starts, and closing it once when the application exits. If you are using a program that has multiple threads, open the board once in the first main thread and then pass the board handle to every thread that is subse-

quently created. You must call R64BrdClose for every board that is open with R64BrdOpen. You should also call R64BrdClose in the same thread the R64BrdOpen was called in.

36.2 R64SysBoardFindByNum

Prototype R64RC R64SysBoardFindByNum(BFU32 *Number*, PR64ENTRY *pEntry*)

Description Finds a R64 on the PCI bus with a given number.

Parameters

Number

The number of the board to find. Boards are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots as long as the number of boards and the R64 is in the same PCI slot.

pEntry

A pointer to an empty R64ENTRY structure, used to tell the R64BrdOpen function which board to open.

Returns

R64_OK	The board was successfully found.
BFSYS_ERROR_NOTFOUND	There is no board with this number.

Comments If you have only one board in your system set *Number* = 0 and only call this function once. This function can be used to enumerate all of the boards in a system. It can be called repeatedly, incrementing *Number* each time, until the function returns BFSYS_ERROR_NOTFOUND.

There is no standard way to correlate the *Number* parameter of this function to the PCI slot number. Every motherboard and BIOS manufacturer has a different scheme. You can use the system configuration utility, SysReg, to determine the relationship between slot number and board *Number*, by setting the board ID switches different for each board in your system and walking through all the installed boards.

36.3 R64BrdOpen

Prototype R64RC R64BrdOpen(PR64ENTRY *pEntry*, R64 **pBoard*, BFU32 *Mode*)

Description Opens a R64 for access. This function must return successfully before any other R64 SDK functions are called (with the exception of R64SysBoardFindNum).

Parameters

pEntry

A pointer to a filled out R64ENTRY structure. This structure describes which board is to be opened. The structure is filled out by a call to the R64SysBoardFindNum function.

**pBoard*

A pointer to a R64 handle. This handle is used for all further accesses to the newly opened board. This function takes a pointer to a handle where as all other functions just take a handle.

Mode

This parameter allows for different modes of opening the board, one or more of these parameters can be ORed together:

0 - board will open normally but not initialized. Board registers are not changed.
BFSysInitialize - initialize the board.
BFSysExclusive - open only if no other process has, and do not allow any subsequent process to open the board.
BFSysNoIntThread - do not start interrupt IRP thread.
BFSysNoCameraOpen - do not open any configured cameras.
BFSysNoAlreadyOpenMess - suppress already open warning message.
BFSysNoOpenErrorMess - suppress all error popups in open function
BFSysSecondProcessOpen - special mode that allows the board to be opened twice in the same process (includes of some of the above modes).

Returns

R64_OK	Function was successful.
BF_ALREADY_OPEN_PROC	Another thread in the process has already opened the board, this open not allowed.
BF_ALREADY_OPEN_EXEC_YOU	Another process has opened the board in BFSysExclusive mode, this open is not allowed.
BF_ALREADY_OPEN_EXEC_ME	You have attempted to open the board in BFSysExclusive but the board is already opened by another process, this open not allowed.

BF_BAD_MUTEX	Error occurred allocating a MUTEX object from the operating system.
BF_BAD_CAM	Error opening one of the camera files configured for this board.
BF_BAD_INIT	Error initializing the board.
BFSYS_ERROR_ALLOCATION	Error allocating resources required for this board.
BFSYS_ERROR	Low-level error opening board.

Comments

This function opens the board for all accesses. Call the R64SysBoardFindNum function first to find the board you wish to open. Then call this function to open to board. The board must be opened before any other functions can be called. When you are finished accessing the board you must call R64BrdClose, before exiting your process. Failure to call R64BrdClose will result in incorrect board open counts used by the driver.

If this function fails, you cannot access the board. Also, you do not need to call R64BrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board.

You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call R64BrdClose first.

Calling this function with *Mode* = BFSysInitialize initializes the board and sets it up for the first camera that is configured for this board. If another process has already opened the board using this flag, the board will not be re-initialized, but you will have access to the board in the state that it is.

The *Mode* = BFSysExclusive is designed to guarantee that only one process can have the board open at a time. If the board has already been opened with this flag you will not be able to open it again, regardless of the *Mode* parameter that you use.

If *Mode* = BFSysExclusive, then you will not be able to open the board if any other process has already opened the board, regardless of the mode the other process used to open the board. Finally, if you do succeed in opening the board in this mode, no other processes will be allowed to open the board.

36.4 R64BrdOpenCam

Prototype R64RC R64BrdOpenCam(PR64ENTRY *pEntry*, R64 **pBoard*, BFU32 *Mode*, PBFCHAR *ForceCamFile*)

Description Opens a R64 for access. This function must return successfully before any other R64 SDK functions are called (with the exception of R64SysBoardFindNum).

Parameters

pEntry

A pointer to a filled out R64ENTRY structure. This structure describes which board is to be opened. The structure is filled out by a call to the R64SysBoardFindNum function.

**pBoard*

A pointer to a R64 handle. This handle is used for all further accesses to the newly opened board. This function takes a pointer to a handle where as all other functions just take a handle.

Mode

This parameter allows for different modes of opening the board, one or more of these parameters can be ORed together:

0 - board will open normally but not initialized. Board registers are not changed.
BFSysInitialize - initialize the board.
BFSysExclusive - open only if no other process has, and do not allow any subsequent process to open the board.
BFSysNoIntThread - do not start interrupt IRP thread.
BFSysNoCameraOpen - do not open any configured cameras.
BFSysNoAlreadyOpenMess - suppress already open warning message.
BFSysNoOpenErrorMess - suppress all error popups in open function
BFSysSecondProcessOpen - special mode that allows the board to be opened twice in the same process (includes of some of the above modes).

ForceCamFile

The camera file to open. The camera file should include the name and the file extention. If only the file name and extention are given, the camera configuration path is searched for the camera file. (The camera configuration path by default is the Config folder under the SDK root.) If the full path is given, the camera file will try and be opened from that location.

Returns

R64_OK	Function was successful.

BF_ALREADY_OPEN_PROC	Another thread in the process has already opened the board, this open not allowed.
BF_ALREADY_OPEN_EXEC_YOU	Another process has opened the board in BFSysExclusive mode, this open is not allowed.
BF_ALREADY_OPEN_EXEC_ME	You have attempted to open the board in BFSysExclusive but the board is already opened by another process, this open not allowed.
BF_BAD_MUTEX	Error occurred allocating a MUTEX object from the operating system.
BF_BAD_CAM	Error opening one of the camera files configured for this board.
BF_BAD_INIT	Error initializing the board.
BFSYS_ERROR_ALLOCATION	Error allocating resources required for this board.
BFSYS_ERROR	Low-level error opening board.

Comments

This function opens the board for all accesses. Call the R64SysBoardFindNum function first to find the board you wish to open. Then call this function to open to board. The board must be opened before any other functions can be called. When you are finished accessing the board you must call R64BrdClose, before exiting your process. Failure to call R64BrdClose will result in incorrect board open counts used by the driver.

If this function fails, you cannot access the board. Also, you do not need to call R64BrdClose.

This function must be called once for each board that needs to be opened. Each board will have its own handle when opened. When you want to perform an operation on a certain board, pass the function the handle to that board.

You should only call this function once per process per board and in only one thread. You can call this function again in the same process but you must call R64BrdClose first.

Calling this function with *Mode* = BFSysInitialize initializes the board and sets it up for the first camera that is configured for this board. If another process has already opened the board using this flag, the board will not be re-initialized, but you will have access to the board in the state that it is.

The *Mode* = BFSysExclusive is designed to guarantee that only one process can have the board open at a time. If the board has already been opened with this flag you will not be able to open it again, regardless of the *Mode* parameter that you use.

If *Mode* = BFSysExclusive, then you will not be able to open the board if any other process has already opened the board, regardless of the mode the other process used to open the board. Finally, if you do succeed in opening the board in this mode, no other processes will be allowed to open the board.

36.5 R64BrdCamSel

Prototype R64RC R64BrdCamSel(R64 *Board*, BFU32 *CamIndex*, BFU32 *Mode*)

Description Sets a board's current camera to the camera with the given index. Depending on the Mode, the board can also be initialized for this camera.

Parameters

Board

Handle to board.

CamIndex

Index of camera to become current. Index is set in SysReg.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but do not modify the board.
BFSysConfigure - initialize the board for this camera.

Returns

BF_OK	Function was successful.
R64_INCOMP	Camera file is incompatible with this board, or camera file is incompatible with this version of the SDK.
R64_BAD_CNFG	An error occurred initializing the board for this camera file.
R64_BAD_FIRMWARE	An error occurred downloading the camera file requested firmware.
BF_BAD_CAM_INDEX	The camera index is not valid or the camera index is empty.

Comments Each board has associated with it a list of configured cameras (set in the SysReg application) and a current camera. By default, the current camera is the first camera in the list of configured cameras. The current camera is important because it dictates the parameters used for acquisition. There must be a current camera set in order to use the acquisition functions. This function allows you to pick one of the configured cameras to be the current camera.

If *Mode* = BFSysConfigure, the board will be initialized for the given camera.

This function is useful for switching on-the-fly between multiple pre-configured camera types.

36.6 R64BrdCamSetCur

Prototype R64RC R64BrdCamSetCur(R64 *Board*, PR64CAM *pCam*, BFU32 *Mode*)

Description Sets the current camera to the camera object pCam that is not necessarily one of the pre-configured cameras. The board can be optionally initialized to the camera.

Parameters

Board

Handle to board.

pCam

A camera object.

Mode

When setting the current camera, additional initialization can be performed:

0 - make the camera the current camera but does not modify the board.
BFSysConfigure - initialize the board for this camera.

Returns

R64_OK	Function was successful.
R64_INCOMP	Camera file is incompatible with this board, or camera file is incompatible with this version of the SDK.
R64_BAD_CNFG	An error occurred initializing the board for this camera file.
R64_BAD_FIRMWARE	An error occurred downloading the camera file requested firmware.
BF_BAD_CAM_INDEX	The camera index is not valid or the camera index is empty.

Comments This function sets the current camera to a camera object that is not one of the cameras already configured for the board (via SysReg). The camera must already be opened successfully (see R64CamOpen).

This function allows you to handle your own camera management. You can select, open, configure and close cameras to suit your applications needs independently of the SDK's camera management.

If *Mode* = BFSysConfigure, the board will be initialized for the given camera.

36.7 R64BrdInquire

Prototype R64RC R64BrdInquire(R64 *Board*, BFU32 *Member*, PBFU32 *pVal*)

Description Returns parameters about the current board.

Parameters

Board

Handle to board.

Member

Parameter to inquire about:

BFBrdInqModel - returns the board model. The parameter *pVal* will point to one of:

BFBrdValModelR64Cl - 128K DPM, Normal Speed.
BFBrdValModelR64ClB - 256K DPM, Normal Speed.
BFBrdValModelR64ClH - 128K DPM, High Speed.
BFBrdValModelR64ClHB - 256K DPM, High Speed.
BFBrdValModelR64Dif - 128K DPM, Normal Speed.
BFBrdValModelR64DifB - 256K DPM, Normal Speed.
BFBrdValModelR64DifH - 128K DPM, High Speed.
BFBrdValModelR64DifHB - 256K DPM, High Speed.

BFBrdInqLUT - the type of LUT mounted on this board. The parameter *pVal* will point to one of:

BFBrdValLUTNone
BFBrdValLUT16

BFBrdInqIDReg - the current setting of the ID switch on the board (0,1,2,3).

BFBrdInqNumCams - the number of cameras attached to the board.

Camera inquiry parameters are also valid. The *pVal* parameter will point to the value for the board's current camera. See R64CamInquire for the meaning of these members.

R64CamInqXXXX

pVal

Pointer returned containing the requested value.

Returns

R64_OK	Function was successful.
R64_BAD_INQ_PARAM	The *Member* parameter is unknown.

BF_BAD_CNF_PTR	Invalid configuration pointer.
BF_BAD_ITEM_ID	The ID of configuration item is not in configuration structure.
BF_BAD_ID_EMPTY	The configuration item is empty.
BF_DEST_TO_SMALL	The configuration item is larger than the destination area.
BF_BAD_INDEX	The configuration item has a bad index parameter.

Comments

This function is used to inquire of the system characteristics of the board. This function can also be called with R64CamInquire *Members*, which are then passed to that function using the board's current camera.

36.8 R64BrdClose

Prototype R64RC R64BrdClose(R64 *Board*)

Description Closes the board and frees all associated resources.

Parameters *Board*

Handle to board.

Returns

R64_OK	In all cases.

Comments This function closes the board and releases associated resources. This function must be called whenever a process exits regardless of the reason the process is exiting. The only time that this function does not have to be called is if R64BrdOpen fails. This function decrements the internal counters that are used to keep track of the number of processes that have opened the board.

36.9 R64BrdAqTimeoutSet

Prototype R64RC R64BrdAqTimeoutSet(R64 *Board*, BFU32 *Timeout*)

Description Sets the timeout value for this board's current camera.

Parameters

Board

Board to select the camera for.

Timeout

New value for timeout, in milliseconds.

Returns

R64_OK	If successful.
BF_BAD_CNF_PTR	Invalid configuration pointer.
BF_BAD_ITEM_ID	The ID of configuration item is not in configuration structure.
BF_BAD_INDEX	The configuration item has a bad index parameter.
BF_BAD_ID_EMPTY	The configuration item is empty.
BF_BAD_CNF_SIZE	The configuration structure not big enough to accommodate operation.

Comments This function sets the timeout value for this board's current camera.

36.10 R64BrdAqSigGetCur

Prototype R64RC R64BrdAqSigGetCur(R64 *Board*, PBFVOID **pAqSig*)

Description Gets the current acquire signal.

Parameters

Board

Board to select.

**pAqSig*

Pointer to storage for acquire signal.

Returns

R64_OK	If successful.
BF_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.

Comments This function gets the current acquire signal. See the section on signals to understand what a signal is.

36.11 R64BrdAqSigSetCur

Prototype R64RC R64BrdAqSigSetCur(R64 *Board*, PBFVOID *pAqSig*)

Description Sets the current acquire signal to a signal record provided by the caller.

Parameters

Board

Board to select.

pAqSig

Pointer to caller's signal record.

Returns

R64_OK	In all cases.

Comments This function sets the current acquire signal to a signal record provided by the caller.

36.12 R64BrdCamGetFileName

Prototype R64RC R64BrdCamGetFileName(R64 *Board*, BFU32 *Num*, PBFCHAR *CamName,* BFSIZET *CamNameStLen*)

Description Gets the file name of the attached camera(s).

Parameters

Board

Board to select.

Num

Camera number to get the name of.

CamName

Contains the file name of the camera configuration.

CamNameStLen

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *CamName*.

Returns

R64_OK	If successful.
BF_BAD_CAM_LIST	There was not list of cameras found.
BF_BAD_CAM_INDEX	Invalid camera index.

Comments This function can be used to get the file name for one of the attached camera configurations. These configurations are attached to the board in SysReg. The *Num* parameter corresponds to the number configuration in the list of attached cameras in SysReg.

36.13 R64BrdCamGetCur

Prototype R64RC R64BrdCamGetCur(R64 *Board*, PR64CAM **pCam*)

Description Gets a pointer to the current camera configuration structure.

Parameters

Board

Board to select.

**pCam*

Pointer to new current quad table.

Returns

R64_OK	In all cases.

Comments This function a pointer to the current camera configuration structure. The structure contains all the data that is in the camera configuration file.

R64 Acquisition

Chapter 37

37.1 Introduction

The acquisition functions are some of the most important in the SDK. While the initialization functions set up the board's registers for a particular camera, these functions do most of the work required to get the board reading to DMA the images to memory.

The functions are organized into three groups:

Setup functions
Command function
Clean up functions

The concept here is that the setup functions are time and CPU intensive, so they should be called before any time critical processing has begun. In a sense, these are extensions of the initialization process. Once the setup functions are called for a particular destination buffer, they need not be called again.

The command function is designed to be used during time critical operations, and require minimal CPU time. They can be told to return immediately so that other operations can be performed simultaneously with acquisition. The command function can be called over and over, as many times as needed, to acquire the buffers locked down in the setup functions.

The clean up functions free up any resources allocated in the setup functions, and put the DMA engine in an idle mode.

For example, the basic flow of a program would be:

```
// Initialization

R64AqSetup
Loop
{
   R64AqCommand
}
R64AqCleanup
```

The bulk of the work is done in the R64AqSetup functions. These functions create a scatter gather table based on the virtual memory address, called a relative QTab.

The relative QTab is passed to the kernel driver, where the destination buffer is locked down (so that it cannot be paged to disk) and the physical addresses are determined for each page of the buffer (pages are 4K bytes in 32-bit Windows). These physical addresses are used to build a physical QTab. This physical QTab is then written to the board in preparation scatter gather DMAing.

Finally, the DMA engine is initialized and started. Again, this function need be called only once, for a particular destination buffer.

The R64AqCommand can be called either synchronously or asynchronously. In the synchronous case, the function does not return until the command has completed. In the asynchronous case, the function returns as soon as the command has been issued to the board. If you need to synchronize your process with the acquisition, you can use the R64AqWaitDone function or you can use the signaling system. Signaling is the best way to synchronize to repeated end of frame signals as they do not take any CPU cycles.

37.2 R64AqSetup

Prototype R64RC R64AqSetup(R64 *Board*, PBFVOID *pDest*, BFU32 *DestSize*, BFS32 *Stride*, BFU32 *DestType*)

Description Sets a R64 for acquisition to a host buffer. This function must be called before any acquisition command is issued.

Parameters

Board

Handle to board.

pDest

A void pointer to the destination buffer (already allocated).

DestSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in *bytes*, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

DestType

Type of destination memory:

BFDMADataMem - host memory
BFDMABitmap - display memory

Returns

R64_OK	If successful.
BF_BAD_ALLOC	Resources required for this operation could not be allocated.
R64_BAD_STOP	The function was unable to reset the board.
R64_BAD_CON_PARAM	Bad parameter to control functions.
R64_BAD_CNF	Error extracting information from camera configuration.
R64_BAD_MODEL	Error building QTab from model parameters.
BF_BAD_ROI	Error calculating QTab for ROI

BF_BAD_ARGS	Illegal argument detected.
BF_BAD_SEMAPHORE	Error creating or using semaphore.
R64_AQ_NOT_SETUP	R64AqSetup has not yet been called and the board is not ready for an acquisition command.

Comments

This function sets up the entire R64's acquisition systems for acquisition to host. It lock down the memory, build QTABs, and set up the DMA engine. The QTABs are based on the current camera pointer in the board structure. This function need be called only once, before acquisition begins. It does not need to be called again unless R64AqCleanUp is called. R64AqCleanUp should be called when done acquiring in order to free up resources used by this process. The only reason to call this function again is to acquire into a different host buffer or acquire with a different type of camera. Once this function is called, the function R64AqCommand is used to snap, grab, freeze or abort acquisition.

37.3 R64AqCommand

Prototype R64RC R64AqCommand(R64 *Board*, BFU32 *Command*, BFU32 *Mode*)

Description Once the R64 is set up for acquisition with R64AqSetup, this function issues the actual acquisition command.

Parameters

Board

Handle to board.

Command

Acquisition command to initiate:

BFConGrab - starting at the beginning of the next frame, acquire every frame.
BFConSnap - starting at the beginning of the next frame, acquire one frame.
BFConFreeze - stop acquiring at the end of the current frame. If in between frames, do not acquire any more frames.
BFConAbort - stop acquiring immediately. If in the middle of the frame, the rest of the frame will not be acquired.
BFConReset - reset conditions after an abort or overflow. The board is set up as it was when R64AqSetup was called.

Mode

This function can operate in two modes:

BFConAsync - as soon as the command is issued return.
BFConWait - wait for the current command to complete. For a snap, the function will return when the entire frame has been acquired into memory. For a grab, the function will wait until the first frame has begun to be acquired. For a freeze, the function waits for the current frame to end. All other commands return immediately.

Returns

R64_OK	If successful.
R64_AQ_NOT_SETUP	R64AqSetup has not yet been called and the board is not ready for an acquisition command.
R64_BAD_AQ_CMD	A snap or grab command has already been issued and the board is already acquiring.
R64_BAD_STOP	The function was unable to reset the board.
R64_BAD_CON_PARAM	Bad parameter to control functions.
R64_TIMEOUT	Timeout wait for command.

BF_BAD_SIGNAL	Interrupt signal unknown
BF_SIGNAL_CANCEL	Interrupt signal was cancelled.
BF_WAIT_FAILED	Wait for object failed.
R64_AQSTRT_TIMEOUT	A time-out occurred waiting for acquisition to begin.
R64_AQEND_TIMEOUT	A time-out occurred waiting for acquisition to end.

Comments

This function can only be called after R64AqSetup is called. R64AqSetup need only be called once for any number and combination of calls to R64AqCommand. Basically, you call R64AqSetup once for a given host buffer, then call R64AqCommand as many times as you need to get data into that buffer. Call R64AqCleanUp when you are done acquiring into that buffer. Then the procedure starts over again for the next buffer.

The R64AqXXXX commands handle both DMA and camera acquisition. No other commands are needed to handle the process of acquiring into memory.

If you call this function with *Mode* = BFConWait, it will wait for the acquisition to complete, in the case of a snap or freeze command, or wait for the acquisition to begin, in the case of a grab command. This is an efficient wait that consumes minimal CPU cycles. The function will return when the last pixel has been DMAed into memory. Alternatively, you can call the function with *Mode* = BFConAsync, and the function will return as soon as the command has been issued. You can find out how much data has been DMAed by calling R64AqProgress. You can also just wait for the end of acquisition by calling R64AqWaitDone.

The functions mentioned above use the SDK's signaling system to efficiently wait for events. If you wish to have a higher level of control you can call the R64SignalXXXX functions yourself. These functions use a signaling system that allow processes to be notified of R64 events and interrupts. For acquisition, wait for the BFIntTypeEOD signal. This signal occurs at the end of every frame, in both grab and snap mode. This signal occurs when the last pixel is DMAed into memory.

Calling this function with *Command* = BFConAbort will stop acquisition immediately. The acquisition process can be left anywhere in the frame. You must call this function with *Command* = BFConReset before any more acquire commands can be issued. Alternatively, you can call R64AqCleanUp and start over with R64AqSetup

37.4 R64AqCleanUp

Prototype R64RC R64AqCleanUp(R64 *Board*)

Description Frees all resources used by the acquisition process. Makes sure the board is in a stable state.

Parameters *Board*

Handle to board.

Returns

R64_OK	In all cases.

Comments This function frees all of the resources that were allocated in R64AqSetup. Do not call this function unless you have already called R64AqSetup, and unless you are finished acquiring into the current buffer.

This function does not free the destination buffer passed to R64AqSetup in the *pDest* parameter.

37.5 R64AqWaitDone

Prototype R64RC R64AqWaitDone(R64 *Board*)

Description Waits for the current acquisition to complete.

Parameters *Board*

Handle to board.

Returns

R64_OK	The current acquisition has completed.
R64_AQ_NOT_SETUP	The acquisition process has not been set up yet.
R64_BAD_WAIT	The board is currently in grab mode and acquisition will not end, or there is another acquisition command pending after this one is completed.
R64_AQEND_TIMEOUT	The acquisition time-out expired before the acquisition command completed.
BF_BAD_SIGNAL	Interrupt signal unknown
BF_SIGNAL_CANCEL	Interrupt signal was cancelled.
BF_WAIT_FAILED	Wait for object failed.

Comments This function efficiently waits for the current acquisition to complete. The completion is denoted by the last pixel being DMAed into memory. The function will return with a time-out error if the acquisition has not been completed by the designated acquisition time-out amount. This time is normally set in the camera configuration file, but can be changed in software as well, see R64BrdAqTimeoutSet. This function will return immediately if the acquisition has already completed. This function will also return immediately (with an error code), if the board is in a state where acquisition will not complete without further acquisition commands.

37.6 R64AqProgress

Prototype R64RC R64AqProgress(R64 *Board*, PBFU32 *pCurLine*)

Description Returns the progress, in terms of line number, of the current acquisition.

Parameters

Board

Handle to board.

pCurLine

A pointer to a BFU32. When the function returns this variable, it will contain the line number currently being DMAed.

Returns

R64_OK	If successful.
R64_BAD_CNF	Error extracting data from the current camera configuration.
R64_BAD_CON_PARAM	Error with the destination type.

Comments This function reads the board's registers that indicate the current DMA destination. Based on the current camera configuration and acquisition, the line number is calculated and returned. The line number returned is the line that is currently being DMAed. It is not safe to access the current line in memory, as it may not be completely written into memory. All lines up to the line returned can be safely accessed. However, the function returns an instantaneous value, since the DMA process can move data extremely fast. The value returned from this function, especially for fast cameras, may not be current by the time the function returns.

If acquisition has not yet started, this function will indicate that the current line is line 0. This will only happen right after R64AqSetup has been called. Once the first frame has started, the function will always indicate the current line number, or the last line in the image if acquisition is completed. If an acquisition is pending, this function will still return that the current line is the last line in the image until the next frame starts.

37.7 R64AqFrameSize

Prototype R64RC R64AqFrameSize(R64 *Board*, BFU32 *XSize*, BFU32 *YSize*)

Description This function provides the ability to change the image height and image width.

Parameters

Board

Handle to board.

XSize

The value to change the XSize too.

YSize

The value to change the YSize too.

Returns

R64_OK	If successful.
R64_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R64_CAM_SUPPORT	Cam file being used is not supported by this function.
R64_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R64_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
R64_BAD_CNF_FILE	Could not determine the pixels per clock from the camera file.

Comments This function is used to change the size of the image being acquired, from software. With this function the size of the frame can be changed on the fly, without the use of camera files. This function is limited to use with only free run camera files, and may not work with sophisticated camera files.

This function assumes the CTABs and control registers have already been initialized to a working state by one of the initialization functions (e.g. R64BrdOpen). The function uses the current state to determine how to make the requested modifications. If the current board state is non-functional, this function will fail.

This function can be called before R64AqSetup and the new size will overwrite the size specified by the camera file. To change the size after R64AqSetup has been called R64AqCleanup must be called then R64AqFrameSize and R64AqSetup. The following is an example of the order needed to change the size of the frame after R64AqSetup has been called:

```
// Stop acquisition
```

```
R64AqCleanUp
R64AqFrameSize
R64AqSetup
// Begin acquisition
```

The minimum XSize this functions supports is 8 pixels and a minimum YSize of 1 line. The maximum YSize and XSize is 131,072 lines and pixels. This function will return a R64_BAD_FRM_SIZE error for any problems with the size of the frame. Another precaution to take is that the XSize needs to be a multiple of the pixels per clock. Any XSize value that is not a multiple of the pixels per clock will give a R64_BAD_FRM_SIZE error.

It is left up to the user not to exceed the sensor size of the camera. For example if the user is using a area scan camera with a sensor size of 640x480 and tries and make the frame size 800x600, this function will try to acquire the 800x600 frame size even though the camera can not provide it. The user will end up with a scrambled or unstable image.

37.8 R64AqReengage

Prototype R64RC R64AqReengage(R64 *Board*)

Description Re-engages the QTab into the DMA engine.

Parameters *Board*

Handle to board.

Returns

R64_OK	If successful.
R64_AQ_NOT_SETUP	R64AqSetup has not yet been called and the board is not ready for an acquisition command.

Comments This function is used to engage the QTab. This function only needs to be used if the acquisition or the DMA is aborted in the middle of the frame (for example, when using start-stop triggering).

37.9 R64AqROISet

Prototype

R64RC R64AqROISet(R64 *Board*, BFU32 *XOffset*, BFU32 *YOffset*, BFU32 *XSize*, BFU32 *YSize*)

Description

This function provides the ability to change the region of interest acquired by the camera.

Parameters

Board

Handle to board.

XOffset

The number of pixels to offset in the x-axis.

YOffset

The number of pixels to offset in the y-axis.

XSize

The value to change the XSize too.

YSize

The value to change the YSize too.

Returns

R64_OK	If successful.
R64_BAD_FRM_SIZE	Invalid frame size. The frame can be too big or small, or the XSize is not a multiple of 4.
R64_CAM_SUPPORT	Cam file being used is not supported by this function.
R64_BAD_VCTAB	Couldn't find a valid VStart segment 0.
R64_BAD_HCTAB	Couldn't find a valid HStart segment 0 or 1 and/or HStop.
R64_BAD_CNF_FILE	Could not determine the pixels per clock from the camera file.

Comments

This function is used to change the region of interest (ROI) of the image being acquired from the camera, from software. With this function the ROI of the frame can be changed on the fly, without the use of camera files. This function is limited to use with only free run camera files, and may not work with sophisticated camera files.

This function assumes the CTABs and control registers have already been initialized to a working state by one of the initialization functions (e.g. R64BrdOpen). The function uses the attached camera file to determine how to make the requested modifications. The ROI must stay within the boundaries of the attached camera sensor being use. If the current board state is non-functional, this function will also be non-functional.

This function can be called before R64AqSetup and the new settings will overwrite the settings specified by the camera file. To change the size after R64AqSetup has been called, R64AqCleanup must be called then R64AqROISet and R64AqSetup. The following is an example of the order needed to change the ROI of the frame after R64AqSetup has been called:

```
// Stop acquisition
R64AqCleanUp
R64AqROISet
R64AqSetup
// Begin acquisition
```

The minimum XSize this functions supports is 8 pixels and a minimum YSize of 1 line. The maximum YSize and XSize is 131,072 lines and pixels. This function will return a R64_BAD_FRM_SIZE error for any problems with the size of the frame. Another pre-caution to take is that the XSize needs to be a multiple of the pixels per clock. Any XSize value that is not a multiple of the pixels per clock will give a R64_BAD_FRM_ SIZE error.

It is left up to the user to verify that the ROI dose not exceed the x and y sizes or boundaries in the camera sensor. For example if the user is using a area scan camera with a sensor size of 640x480 and tries and make the frame size 800x600, this function will try to acquire the 800x600 frame size even though the camera can not provide it. The user will end up with a scrambled or unstable image. Another example would be if the same 640x480 camera file is used with an xsize that is less than 640 and a ysize that is less then 480, but the x or y offset puts the ROI beyond the 640x480 borders.

R64 Camera Configuration

Chapter 38

38.1 Introduction

One of the most powerful features of the R64 is the ability for the board to interface to an almost infinite variety of cameras. The knowledge behind these interfaces is stored in the camera configuration files.

The normal way a R64 application works is that the board is initialized to interface to the camera currently attached to the board. The currently attached camera is selected in the SysReg utility program. Normally, an application is written so that it will work with whatever camera is attached. The board is initialized for the currently attached camera when R64BrdOpen is called. If an application is written this way there is no need to call any of the functions in this chapter. However, some users may want to manage what cameras are attached and how the user switches between them using their own software. For this reason, these camera configuration functions are provided.

The normal flow for an application that wants to manage its own camera files is as follows:

```
R64BrdOpen
R64CamOpen
R64BrCamSetCur
// processing and acquisition
R64CamClose
R64BrdClose
```

If using more than one camera:

```
R64BrdOpen
R64CamOpen     // open camera 0
R64CamOpen     // open camera 1
R64BrCamSetCur // configure for camera 0
// processing and acquisition
R64BrCamSetCur // configure for camera 1
// processing and acquisition
R64CamClose    // close camera 0
R64CamClose    // close camera 1
R64BrdClose
```

38.2 R64CamOpen

Prototype R64RC R64CamOpen(R64 *Board*, PCHAR *CamName*, PR64CAM **pCam*)

Description Allocates a camera configuration object, opens a camera configuration file, and loads the file into the object.

Parameters

Board

Handle to board.

CamName

The name of the camera file to open. Do not include the path. The camera file must be in the configuration directory (see the SysReg application). For example: "BitFlow-Synthetic-256x256-E1.r64".

**pCam*

A pointer to a camera object. The memory to hold the object is allocated in this function.

Returns

R64_OK	If successful.
R64_NO_CNFDIR_REG_KEY	The configuration directory entry is missing in the register (run SysReg).
R64_BAD_PATH	Error building the path to the camera file.
R64_BAD_STRUCT	Error calculating the size of the camera structure.
BF_BAD_ALLOC	Cannot allocate memory to perform open.
R64_BAD_CNF_FILE	Error opening or reading configuration file.
BF_WRONG_CAM	Camera file extension is not ".r64", wrong type of cam file.
BF_BAD_HEADER	Error in configuration file header. This could include an error in one or more of the following items: signature (R64 configuration), endian test (endian model is unknown), revision (camera revision is incompatible), size (size of file is not the same as written) and CRC (byte error in file).
BF_BAD_BINR	Error reading configuration item from file.
BF_BAD_CNFA	Error inserting configuration item into camera object.

Comments This function allocates memory to hold a camera configuration object, locates the given camera configuration file in the configuration directory, checks the file for errors, then loads the camera configuration parameters into the camera object. The

camera object is used to tell the system how to set up the board to acquire from a particular camera. Use the programs CamVert and CamEd to edit camera configuration files.

The resulting camera object can be passed to other functions such as R64BrdCamSetCur.

The resources allocated by the function must be freed by calling R64CamClose.

38.3 R64CamInquire

Prototype R64RC R64CamInquire(R64 *Board*, PR64CAM *pCam*, BFU32 *Member*, PBFU32 *pVal*)

Description Returns information about the given camera.

Parameters

Board

Handle to board.

pCam

Camera whose characteristics are requested.

Member

Characteristic to find the value of. The member must be one of:

BFCamInqXSize - width of image in pixels.
BFCamInqYSize0 - height of image in lines.
BFCamInqFormat - image format.
BFCamInqPixBitDepth - depth of pixel in bits, as acquired to host.
BFCamInqPixBitDepthDisplay - depth of pixel in bits, as acquired to display.
BFCamInqBytesPerPix - depth of pixel in bytes, as acquired to host.
BFCamInqBytesPerPixDisplay - depth of pixel in bytes, as acquired to display.
BFCamInqBitsPerSequence - depth of multi-channel pixel in bits, as acquired to host.
BFCamInqBitsPerSequenceDisplay - depth of multi-channel pixel in bits, as acquired to display.
BFCamInqFrameSize0 - total size of image in bytes, as acquired to host.
BFCamInqDisplayFrameSize0 - total size of image in bytes, as acquired to display.
BFCamInqFrameWidth0 - width of image in bytes, as acquired to host.
BFCamInqDisplayFrameWidth - width of image in bytes, as acquired to display.
BFCamInqAqTimeout - number of milliseconds to wait before acquisition command times out.
BFCamInqCamType - camera type.
BFCamInqControlType - type of camera control accessible through API.

pVal

Pointer to value of the characteristic.

Returns

R64_OK	If successful.
R64_BAD_INQ_PARAM	Unknown *Member* parameter.
BF_BAD_CNF_PTR	Invalid configuration pointer.
BF_BAD_ITEM_ID	The ID of configuration item is not in configuration structure.
BF_BAD_ID_EMPTY	The configuration item is empty.
BF_DEST_TO_SMALL	The configuration item is larger than the destination area.
BF_BAD_INDEX	The configuration item has a bad index parameter.

Comments

This function is used to inquire about characteristics of a camera. For 8, 24 and 32-bit cameras, the parameter R64CamInqFrameSize0 is equal to R64CamInqDisplayFrame-Size0. The parameter only differs for cameras with pixels depths of more than 8 bits per channel.

38.4 R64CamClose

Prototype R64RC R64CamClose(R64 *Board*, PR64CAM *pCam*)

Description Frees resources used by a camera object.

Parameters *Board*

Handle to board.

pCam

Camera object.

Returns

R64_OK	In all cases.

Comments This function frees all resources used by a camera object.

38.5 R64CamAqTimeoutSet

Prototype R64RC R64CamAqTimeoutSet(R64 *Board*, PR64CAM *pCam*, BFU32 *Timeout*)

Description Sets the acquisition timeout variable in the given camera configuration.

Parameters

Board

Handle to board.

pCam

Camera whose characteristics are requested.

Timeout

New value for timeout, in milliseconds.

Returns

R64_OK	If successful.
BF_BAD_ITEM_ID	The ID of configuration item is not in configuration structure.
BF_BAD_ID_EMPTY	The configuration item is empty.
BF_BAD_CNF_SIZE	The configuration structure not big enough to accommodate operation.
BF_BAD_INDEX	The configuration item has a bad index parameter.

Comments This function sets the timeout value for an earlier configuration

R64 Interrupt Signals

Chapter 39

39.1 Introduction

The purpose of the Signal Function calls is to make hardware interrupts available to user-level applications in a simple and efficient set of functions. In fact, under all modern Windows operating systems, there is no way for a user-level application to get direct notification of a hardware interrupt. Only kernel-level drivers can contain interrupt service routines (ISR). Most customers do not want to deal with the complications of writing ISRs anyway, so BitFlow has come up with this signaling system.

A signal is a wrapper around a Windows semaphore object. The signal has a state and a queue. Every time an interrupt occurs, the signal's state changes. The nice thing about signals is that you can wait for their state to change, without using any CPU cycles. This is what makes them so efficient. This means that you can have one thread processing images while another is waiting for the next image to be completely DMAed. The thread that is waiting for the signal consumes very little CPU time, thus making most of the CPU available for processing.

The way these functions are used is that you start by creating a signal with the R64SignalCreate. There are a number of different interrupts that the signal can wait for, and it is in this function that you specify the one you want. Once the signal is created, your application waits for the interrupt with either the R64SignalWait or the R64SignalWaitNext function. The difference being the R64SignalWait function uses a signals queue. If an interrupt has occurred before this function is called, then this function will return immediately. It will continue to return immediately until there are no more interrupts in the queue. The R64SignalWaitNext function always waits for the next interrupt after being called, regardless of how many have occurred since it was last called.

Signals can be used in a single thread application, but whenever one of the wait functions is called, execution will be blocked until the interrupt occurs. Because this situation can potentially hang a process, a timeout parameter is provided for all of the wait functions. If you need an application to process data while waiting on an image to be captured, create a separate thread to call the wait function. Meanwhile, another thread can be processing with most of the CPUs cycles. A thread waiting on a signal can be cancelled with the function R64ThreadCancel. This causes the waiting thread to return from the wait function with an error code indicating that it has been cancelled.

The following pseudo-code illustrates how these functions can be called:

```
Int ImageIn = 0
main ()
{
   R64BrdOpen// open board
   R64SignalCreate// create the signal for EOF
   CreateThread(EOFThread)// create a thread
   while (KeepProcessing)// main processing loop
   {
         // here we loop until we have an image
         while (ImageIn !=1)
         {
               // secondary processing
         }

         // now we have an image so process it

         ImageIn = 0             // reset variable

         // primary image processing

         }
   }

   // clean up
   R64SignalCancel// cancel signal kill thread
   R64SignalFree// free signal resources
   R64BrdClose// close board
}

// thread to watch for end of frame
EOFThread()
{
   loop
   {
         rv = R64SignalWait      // wait for signal
         if(rv == CANCELED)      // cancel?
               exit loop         // yes, kill this thread else
         else
               ImageIn = 1       // no, set new image flag
   }
}
```

39.2 R64SignalCreate

Prototype R64RC R64SignalCreate(R64 *Board*, BFU32 *Type*, PR64SIGNAL *pSignal*)

Description Creates a signal that will allow user level thread to be notified of hardware interrupts.

Parameters

Board

Handle to board.

Type

Type of interrupt signal to create. Must be one of the following:

BFIntTypeHW - hardware exception.
BFIntTypeFIFO - video FIFO overflow.
BFIntTypeCTab - interrupt bit in VCTAB is set.
BFIntTypeEOD - End of DMA. Occurs when the last pixel has been DMAed into memory. Users will create this signal most of the time.
BFIntTypeEOF - End of frame from the acquistion.
BFIntTypeSerial - Serial communication.
BFIntTypeTrig - Interrupt on every trigger pulse.

pSignal

Pointer to R64SIGNAL structure.

Returns

R64_OK	If successful.
BF_BAD_SEMAPHORE	Could not get semaphore object from operating system.
BF_BAD_ALLOC	Could not allocate memory for signal.
BF_BAD_ARGS	Invalid type of interrupt.

Comments This function creates a signal object that is used to receive interrupt notifications from the R64. The R64SignalWaitXXXX function takes a signal as a parameter. These functions efficiently wait for an interrupt of the given type to occur. The best way to use a signal is to create a separate thread that calls one of the R64SignalWaitXXXX functions. This thread will consume minimal CPU cycles until the interrupt occurs. When the interrupt occurs, the signal is notified and the R64SignalWaitXXXX functions will return. The thread can then take appropriate action, calling whatever functions are necessary and/or send messages to the main application thread.

This signaling system is the only way to handle R64 interrupts at the user application level.

More than one signal can be created for the same interrupt on the same board. Also, more than one process and/or thread can wait for the same interrupt. When the interrupt occurs, all of the signals will be notified in the order they were created. The signal created by this function receives interrupt notification only from the R64 passed to this function in the *Board* parameter.

The most frequently used signal is *Type* = R64IntTypeEOD. The R64AqSetup function automatically sets the interrupt bit in the last quad in the QTab of the current image. This signal will be notified when the last pixel of the image has been DMAed into memory, and the current acquisition is done in the case of a snap or freeze.

The signal created by this function must be cleaned up by calling R64SignalFree.

39.3 R64SignalWait

Prototype R64RC R64SignalWait(R64 *Board*, PR64SIGNAL *pSignal*, BFU32 *TimeOut*, PBFU32 *pNumInts*)

Description Efficiently waits for an interrupt to occur. Returns immediately if one has occurred since the function was last called.

Parameters

Board

Handle to board.

pSignal

Pointer to R64SIGNAL previously created by R64SignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout.

pNumInts

Pointer to a BFU32. When the function returns, it will contain the number of interrupts (the interrupt queue) that have occurred since this function was last called.

Returns

R64_OK	Interrupt has occurred.
BF_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
BF_SIGNAL_CANCEL	Signal was canceled by another thread (see R64SignalCancel).
BF_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
BF_WAIT_FAILED	Operating system killed the signal.

Comments This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function will return immediately if the interrupt has occurred since the function was last called with this signal. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

When this function returns, the *pNumInts* parameter will contain the number of interrupts that have occurred since this function was last called. This is essentially an interrupt queue. Normally this will be zero. However, if one or more interrupts have occurred, the function will return immediately and this variable will indicate the number that has occurred. This parameter is useful in determining if frames were missed.

This function will continue to return immediately, reducing the number of interrupts in the queue each time until every interrupt that has occurred has been acknowledged, and the queue is empty.

To wait for the next interrupt and ignore any previous interrupts, use R64SignalWait-Next.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

39.4 R64SignalNextWait

Prototype R64RC R64SignalNextWait(R64 *Board*, PR64SIGNAL *pSignal*, BFU32 *TimeOut*)

Description Like R64SignalWait, this function waits efficiently for an interrupt. However, this version always ignores any interrupts that might have occurred since it was called last, and just waits for the next interrupt.

Parameters

Board

Handle to board.

pSignal

Pointer to R64SIGNAL previously created by R64SignalCreate.

TimeOut

Number of milliseconds to wait for the signal to occur before returning with a timeout error. Set to INFINITE to never timeout

Returns

R64_OK	Interrupt has occurred.
BF_SIGNAL_TIMEOUT	Timeout has expired before interrupt occurred.
BF_SIGNAL_CANCEL	Signal was canceled by another thread (see R64SignalCancel).
BF_BAD_SIGNAL	Signal has not been created correctly or was not created for this board.
BF_WAIT_FAILED	Operating system killed the signal.

Comments This function efficiently waits for an interrupt to occur. While the function is waiting, it consumes minimal CPU cycles. This function waits for the next interrupt, regardless of the number of interrupts in the signal's queue. The first time this function is called with a given signal, it will always wait, even if the interrupt has occurred many times in the threads lifetime.

Use R64SignalWait if you need a function that will return immediately if an interrupt has already occurred.

The *TimeOut* parameter is only as accurate as the high-level operating system clock. On Intel platforms this is usually ± 10 milliseconds.

39.5 R64SignalCancel

Prototype R64RC R64SignalCancel(R64 *Board*, PR64SIGNAL *pSignal*)

Description Cancels a signal, any R64SignalWaitXXX function will return with a value of R64_SIGNAL_CANCEL.

Parameters

Board

Handle to board.

pSignal

Pointer to R64SIGNAL to cancel.

Returns

R64_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.

Comments This function will cancel a signal. It is primarily used by multi-threaded applications where one thread is waiting (with one of the R64SignalWaitXXXX functions) for a signal. Another thread can cancel the signal with this function, thereby waking up the waiting thread. When the waiting thread wakes up and the R64SignalWaitXXXX function returns, the return value can be examined. If the return value is BF_SIGNAL_CANCEL, the thread knows that the signal it was waiting for was canceled, and it can take appropriate action.

This function is usually used as a clean way for the main application thread to tell waiting threads to kill themselves.

Canceling a signal with this function will interfere with its internal interrupt counts. Therefore, this function should only be called when synchronization with the interrupt is no longer important and/or the signal is going to be destroyed.

39.6 R64SignalQueueSize

Prototype R64RC R64SignalQueueSize(R64 *Board*, PR64SIGNAL *pSignal*, PBFU32 *pNumInts*)

Description Reports the current number of interrupts in a signal's queue.

Parameters

Board

Handle to board.

pSignal

Pointer to R64SIGNAL whose queue is to be investigated.

pNumInts

Pointer to BFU32. When the function returns *pNumInts*, it will contain the number of interrupts in the signal's queue.

Returns

R64_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.

Comments This function returns the number of interrupts in a signal's queue. This function is useful for testing to see if any interrupts have come in for a given signal, when you do not want to call one of the R64SignalWaitXXX functions. This function can be called any time.

39.7 R64SignalQueueClear

Prototype R64RC R64SignalQueueClear(R64 *Board*, PR64SIGNAL *pSignal*)

Description Clears interrupts from a single queue.

Parameters

Board

Handle to board.

pSignal

Pointer to R64SIGNAL whose queue is to be investigated.

Returns

R64_OK	If successful.
BF_BAD_SIGNAL	Signal does not exist.
BF_WAIT_FAILED	Error clearing queue.

Comments This function clears all of the interrupts for a given signal's queue. This allows a thread to wait for the next interrupt to occur. This function is usually only used to re-synchronize a signal to the current state of acquisition (i.e., ignore any interrupts that have occurred in the past) before calling R64SignalWait. To always wait for the next interrupt, call R64SignalWaitNext.

39.8 R64SignalFree

Prototype R64RC R64SignalFree(R64 *Board*, PR64SIGNAL *pSignal*)

Description Frees all resources used by a signal.

Parameters

Board

Handle to board.

pSignal

Pointer to R64SIGNAL whose queue is to be investigated.

Returns

R64_OK	In all cases.
BF_BAD_SIGNAL	Signal does not exist.

Comments This function frees the resources used by a signal and removes it from the list of signals that get interrupt notification.

R64 Quad Table Functions

Chapter 40

40.1 Introduction

For almost all R64 applications there will be no need to call any of the functions in this chapter. These are considered mid-level functions and are generally only called indirectly by other, high level, functions. These functions are listed here in case some specialized programming of the R64 is required.

Quad Tables (or QTABS) are simple scatter gather DMA tables. A scatter gather DMA table is a list of instructions that tell the R64 how to DMA images to host. The name quad comes from the fact that each DMA instruction consists of four, 32-bit words: DMA source, DMA destination, DMA size, and a pointer to the next DMA instruction.

The R64 API is very similar to many of the other APIs in this manual. However, the QTab create function is one area where they differ. The primary difference is that the other mid-level APIs require two QTab functions be called, one to create relative QTabs and another to create physical QTabs. With the introduction of the R64 API, these two operations have been collapse into a single QTab creation operation, R64QTabCreate.

40.2 R64QTabCreate

Prototype R64RC R64QTabCreate(R64 *Board*, PR64CAM *pCam*, PBFVOID *pDest*, BFU32 *BufferSize*, BFS32 *Stride*, VQTabHeadPtr *pVirtQTabHead*, BFU32 *DestType*, BFU32 *Options*)

Description Builds a QTab, used for acquisition from a given camera type to a host memory buffer.

Parameters

Board

Handle to board.

pCam

Camera object of the type to build the QTab for.

pDest

A void pointer to the destination buffer.

BufferSize

The size (in bytes) of the destination buffer. This should be the size that was used in the allocation of the buffer.

Stride

The line pitch of the destination buffer. The line pitch is the amount, in pixels, a pointer would have to be increased to move to the next line. Normally, this number is equal to the X size of the image. This value can be negative for images that need to be loaded upside down. When acquiring to host memory, this value can be zero, and the function will calculate the *Stride* for you.

pVirtQTabHead

Pointer to an allocated VQTabHeadPtr structure.

DestType

Type of destination memory:

BFDMADataMem - host memory
BFDMABitmap - display memory

Options

Options for building the QTab. Can be one or more of:

0 - No special options.
BFOptPhysicalMemory - The destination address is physical memory and does not need to be locked down.

Returns

R64_OK	If successful.
R64_BAD_CNF	Error extracting information from the camera object.
R64_BAD_MODEL	The camera configuration contains a QTab model or format that is not understood by this version of the SDK. Or, the camera configuration is such that the QTab cannot be built.
R64_BAD_CON_PARAM	One of the parameters is incorrect.
BF_BAD_ALLOC	Cannot allocate enough memory to build QTab.
BF_BAD_CON_PARAM	One of the parameters is incorrect.
BF_BAD_ROI	Error calculating QTab for ROI

Comments

This function builds a QTab for acquisition of a given camera type into a host memory buffer. The QTab is a table of scatter-gather DMA instructions that the R64 uses to continuously (and without host intervention) DMA camera data to the host memory.

This is a mid-level function and should not be called except for custom programming of the R64. The high-level function R64AqSetup will call this function for you.

Depending on the camera, this function may take a moderate amount of time to calculate the QTab. This function should only be called once, for a given camera and destination. The QTab can be used repeatedly to acquire from the same camera type into the same memory buffer.

This function allocates memory to hold the QTab in the users address space. Call R64QTabFree to release this and other resources allocated in this function.

40.3 R64QTabFree

Prototype R64RC R64QTabFree(R64 *Board*, VQTabHeadPtr *pVirtQTabHead*)

Description Frees resources allocated in R64QTabCreate.

Parameters *Board*

Handle to board.

pVertQTabHead

Pointer to VQTabHead structure previously passed to R64QtabCreate.

Returns

R64_OK	In all cases.

Comments This function releases the memory used to hold the QTab and any other resources allocated in R64QTabCreate.

40.4 R64QTabEngage

Prototype R64RC R64QTabEngage(R64 *Board*, VQTabHeadPtr *pVirtQTabHead*)

Description Sets the board up to use the given QTab for the next DMA operation.

Parameters

Board

Handle to board.

pVirtQTabHead

A pointer to a QTab head structure. This should be the QTab for the host memory buffer that will be acquired into when the next acquisition command occurs.

Returns

R64_OK	If successful.
BF_NULL_POINTER	Invalid *pRelQTabHead* pointer.
BF_QUAD_OVERWRITTEN	Attempting to engage a QTab when on has already been engaged.
BF_QUAD_NOT_WRITTEN	QTab has not been written to board
BF_QUAD_GOING	Attempt to engage QTab when board is DMAing.
BF_BAD_CHAIN	Attempting to select a frame number when there is only one QTab.
BF_BAD_FRAME	Requested frame is not in chain.

Comments This function engages the QTab *pRelQTabHead* so that the board will use this QTab for subsequent DMA operations. This is a mid level function which is not need if the high level functions (e.g. R64AqSetup) are being used to set up DMA.

This function is used when building QTABs using the R64QTabCreate functions. The normal order of function calls is as follows

R64QTabCreate
R64QTabEngage
R64ConDMACommand

This function must be called before DMA is started.

40.5 R64QTabChainLink

Prototype R64R64RC R64QTabChainLink(R64 *Board*, RQTabHeadPtrPtr *ChainArray*, BFU32 *NumInChain*)

Description Chains together a number of QTABs for sequential acquisition in host QTab mode.

Parameters

Board

Handle to board.

ChainArray

A array of pointers to QTABs which describe an set of buffers to be acquired into. The buffers will be filled in the order that their QTABs appear in this array.

NumInChain

The total number of QTab headers in the QTab chain array.

Returns

R64_OK	If successful.
Non-zero	If unsuccessful.

Comments This function effectively sets the board up for continuous acquisition into a sequence of host buffers. Each buffer in is DMAed into in turn, when the last buffer in the chain is filled, the board will DMA the next frame into the first buffer. In other words the chain describes a circular buffer.

The parameter *ChainArray* is an array of pointer to QTab headers. The QTab must already be created by calling R64QTabCreate. After this function returns successfully, the chain must be engaged by calling R64QTABChainEngage function. The normal calling sequence for this function would be as follows:

```
R64QTabCreate
R64QTabChainLink
R64QTabChainEngage
R64ConDMACommand
R64ConAqCommand
```

In the scenario above, no data will move until an acquisition command is sent to the board and the camera sends a frame to the board. Once data is flowing, the board will fill each buffer as the data comes in. Once the last buffer in the chain is filled, the board will continue starting with the first buffer. No host interaction is required for this process to work. The board will send a signal every frame to tell your application when a frame is complete (use R64SignalWait).

40.6 R64QTabChainBreak

Prototype R64RC R64QTabChainBreak(R64 *Board*, RQTabHeadPtrPtr *ChainArray*)

Description Release QTABs from a chain so that they can be reused to build a subsequent chain.

Parameters

Board

Handle to board.

ChainArray

A array of pointers to QTABs which has been already passed to R64QTabChainCreate.

Returns

R64_OK	If successful.
Non-zero	If unsuccessful.

Comments This function is used to release the QTABs that are used by a chain. When a chain is built the QTABs that make it up are modified for use in the chain. If these QTABs need to be used again, the chain must first be broken with this function. After this function is called, the individual QTABs can be use to build another chain, presumable in a different order.

There is no need to call this function during cleanup if the individual QTABs are not going to be used again.

40.7 R64QTabChainEngage

Prototype R64RC R64QTabChainEngage(R64 *Board*, RQTabHeadPtrPtr *ChainArray*, BFU32 *FrameNum*)

Description Takes a successfully created chain and sets the board up to use it.

Parameters

Board

Handle to board.

ChainArray

This is an array of pointers to QTABs which describe an set of buffers to be acquired into. This parameter must first be passed to R64QTabChainCreate.

FrameNum

The buffer number of the first frame in the chain to be acquired into.

Returns

R64_OK	If successful.
Non-zero	If unsuccessful.

Comments After a chain is created using R64QTabChainCreate, the chain must be engaged using this function in order for the board to use it. Creating a chain is not a real time operation and should be done off line. If more than one chain is required, they should all be created first, then this function can be used to select which chain will be acquired into first.

See R64QTabChainCreate for more information.

40.8 R64QTabChainProgress

Prototype R64RC R64QTabChainProgress(R64 *Board*, RQTabHeadPtrPtr *ChainArray*, PBFU32 *pFrameNum*, PBFU32 *pLineNum*)

Description Returns the line number and frame number of current image being DMAed.

Parameters

Board

Handle to board.

ChainArray

This is an array of pointers to QTABs which describe an set of buffers to be acquired into. This parameter must first be passed to R64QTabChainCreate.

pFrameNum

Pointer to receive the number of the current frame being DMAed into.

pLineNum

Pointer to receive the number of the current line being DMAed.

Returns

R64_OK	If successful.
R64_BAD_CNF	Error extracting information from camera configuration file.
R64_BAD_CON_PARAM	Invalid function parameter.

Comments This function is used to check the progress of acquisition while the board is acquiring using a chain. The function will return both the line number and the frame number. This function is fairly computationally intensive and should not be called in a tight loop to monitor progress. This function is best used intermittently to check progress, for example, it can be interleaved with processing.

The best way to overlap acquisition and processing is to create a signal that waits for the quad done signal (end of frame interrupt). Once the signal is asserted, the CPU can freely process the entire frame.

If you need to monitor the boards progress using a tight loop, read the VCOUNT register. Reading a register uses much less CPU time. Even in this case, you should put a sleep in your loop to not overwhelm the board with register reads (which take precedence over DMAing). Again is it better to interleave processing and checking VCOUNT.

40.9 R64ChainSIPEnable

Prototype R64RC R64ChainSIPEnable(R64 *Board*, RQTabHeadPtrPtr *ChainArray*)

Description Enables start-stop interrupt processing (SIP).

Parameters *Board*

Handle to board.

ChainArray

Structure holding information about QTab.

Returns

R64_OK	If successful.
Non-zero	On error.

Comments This function enables start-stop interrupt processing (SIP). This processing is used to reset the DMA engine in a kernel interrupt service routine.

When the board is in start-stop mode, the DMA is terminated before the frame is completely acquired. This termination leaves the DMA engine in an unknown state. The DMA engine must be reset and setup for the next host buffer before the next frame starts. Ordinarily this reset is performed by the application at the user level. However, in the case of a multi threaded application, the reset thread may not be able to reset the DMA engine before the beginning of the next frame (because of CPU load and thread priorities). To solve this problem the BitFlow SDK implements a DMA engine reset in the kernel level interrupt service routine. This code has higher priority than any user level threads. The latency and execution time of the SIP reset is minimized thus reducing the required minimum time between frames. This function turns on this functionality.

SIP only works (and is only required) when the board is in start-stop triggering mode (variable size image acquisition) and when a host QTab chain has been created and engaged. This function must be called before acquisition has started but after the QTab chain is created. This function enable the SIP resetting of the DMA engine, you must call R64ChainSIPDisable to turn the SIP off. This SIP is based on the CTAB interrupt (vertical CTAB column IRQ) which must have an interrupt at location zero.

The example application Flow demonstrates usage of this function.

40.10 R64ChainSIPDisable

Prototype BFRC R64ChainSIPDisable(R64 *BoardId,* RQTabHeadPtrPtr *ChainArray*)

Description Disables Start-Stop Interrupt Processing mode.

Parameters *Board*

Board ID.

ChainArray

Structure holding information about QTab.

Returns

BF_OK	Function succeeded.
Non-zero	Function failed.

Comments See R64ChainSIPEnable for details.

R64 Mid-Level Control Functions

Chapter 41

41.1 Introduction

These functions are used to control the board at a lower level than the R64AqCommand function. In general, an application should not need to use these functions unless special circumstances exist. These functions talk directly to the hardware and make no assumptions about how the rest of the board is set up. Generally, it is a bad idea to mix high-level functions and these mid-level functions.

41.2 R64ConAqCommand

Prototype R64RC R64ConAqCommand(R64 *Board*, BFU32 *Command*)

Description Sends an acquisition command to the board.

Parameters *Board*

Handle to board.

Command

Command send to board:

BFConSnap - snap one frame.
BFConGrab - start continuous acquisition.
BFConFreeze - stop continuous acquisition at the end of the current frame.
BFConAbort - stop acquisition immediately.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	Unknown *Command* parameter.

Comments This function sends an acquisition command directly to the hardware. This is a low-level function and makes no assumptions about the state of the rest of the board.

This command returns immediately.

41.3 R64ConAqStatus

Prototype R64RC R64ConAqStatus(R64 *Board*, PBFU32 *pStatus*)

Description Gets the current acquisition state of the board.

Parameters *Board*

Handle to board.

pStatus

Pointer to BFU32. When this function returns it contains the status of the board. The status will be one of the following:

BFConFreeze - the board is not acquiring.
BFConSnap - the board is currently acquiring one frame.
BFConGrab - the board is currently in continuous acquisition mode.

Returns

R64_OK	In all cases

Comments This function returns the current acquisition status of the board.

41.4 R64ConAqMode

Prototype R64RC R64ConAqMode(R64 *Board*, BFU32 *DestType*)

Description For a given destination type, this function sets the board's acquisition registers, based on the current camera type.

Parameters *Board*

Handle to board.

DestType

Type of acquisition to prepare for:

BFDMABitmap - the destination buffer to be used for display.
BFDMADataMem - the destination buffer needs to contain raw data.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	Unknown *DestType* parameter.

Comments This function sets up the R64's front end acquisition paths for acquiring to a display buffer or a raw data buffer. A display buffer is one that will be used for display on a monitor, and is 8, 24 or 32 bits deep. A raw data buffer is one that the data has the same bit depth as the camera.

This function has no effect for 8, 24 or 32-bit cameras.

This function is normally called automatically by R64AqSetup, and does not need to be called explicitly by an application. An unpredictable result will occur if this function is called while the board is acquiring.

41.5 R64ConInt

Prototype R64RC R64ConInt(R64 *Board*, BFU32 *IntType*, BFU32 *Action*)

Description Disables or enables individual hardware interrupts.

Parameters

Board

Handle to board.

IntType

Type of interrupt:

BFIntTypeHW - hardware exception.
BFIntTypeFIFO - video FIFO overflow.
BFIntTypeCTab - interrupt bit in VCTAB is set.
BFIntTypeEOD - End of DMA. Occurs when the last pixel has been DMAed into memory. Users will create this signal ninety percent of the time.
BFIntTypeEOF - End of frame from the acquistion.
BFIntTypeSerial - Serial communication.
BFIntTypeTrig - Interrupt on every trigger pulse.

Action

Indicates whether to enable or disable the interrupt:

BFConEnable - enable the interrupt.
BFConDisable - disable the interrupt.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	Either the parameter *IntType* or *Action* is unknown.

Comments This function enables or disables the specified hardware interrupt for being invoked on the PCI bus. The driver always has an interrupt service (ISR) routine ready to handle any interrupts that come in. The driver's ISR will automatically reset the appropriate interrupt bits on the board when an interrupt occurs.

To receive notification of interrupts at the user application level, use the signaling system (see the R64SignalXXXX functions). These functions automatically enable the appropriate interrupt when the signal is created, so you do not have to call this function to use an interrupt with the signaling system. However, you can use this function to enable and disable interrupts, based on your application needs, without creating and destroying signals. As a general rule, you should disable any interrupts that you are not using. Every interrupt uses a certain amount of CPU time, even if no application is waiting for it.

When the board is initialized, by default, all interrupts are turned off.

41.6 R64ConDMACommand

Prototype R64RC R64ConDMACommand(R64 *Board*, BFU32 *Command*, BFU32 *Mode*)

Description Issues a DMA command to the board.

Parameters

Board

Handle to board.

Command

DMA command to issue:

BFConDMAGo - start the DMA engine.
BFConDMAAbort - immediately abort the current DMA operation.

Mode

Behavior of this function once the command is issued:

BFConWait - wait for current command to be implemented.
BFConAsync - return as soon as command is issued.

Returns

R64_OK	If successful
R64_AQ_NOT_SETUP	R64AqSetup has not yet been called and the board is not ready for an acquisition command.
R64_BAD_CON_PARAM	Unknown command.
R64_TIMEOUT	Timeout waiting for command to complete. This is only possible if *Mode* = R64ConWait.

Comments This function sends a DMA command to the board. If the *Command* = R64ConDMAGo, this will tell the board to start DMAing. No data will actually be moved until an acquisition command has been issued. The best way to use the R64's DMA engine is to start the DMA and leave it on all the time. Then, control the time when data gets moved to the host by using the acquisition commands.

The command R64ConDMAAbort will stop DMA immediately. This is actually a faster way to stop moving data than aborting acquisition. If acquisition is aborted the board will still DMA until the DPM is empty. Call this function with *Command* = R64ConDMAGo, when ready to start DMAing again.

This function is automatically called by R64AqSetup, and does not normally need to be called by the applications.

If this function is called with *Mode* = R64ConWait, the function will not return until the command has been implemented. Table 41-1 lists what the function will wait for.

Table 41-1 Function Waiting

Command	Waits for…
R64ConDMAGo	DMA engine to get ready for DMA (this is a negligible amount of time).
R64ConDMAAbort	DMA engine to abort current transfer.

When *Mode* = R64ConWait, this function does not efficiently wait, it polls the DMA registers for completion. This is necessary since none of the above conditions causes an interrupt. If the command has not completed before the DMA timeout has expired, the function will return with a timeout error. This DMA timeout is set using the SysReg utility.

41.7 R64DMAProgress

Prototype R64RC R64DMAProgress(R64 *Board*, RQTabHeadPtr *pRelQTabHead*, PBFU32 *pBytesAqed*)

Description Returns the instantaneous number of bytes that have been DMAed so far in the current image.

Parameters

Board

Handle to board.

pRelQTabHead

Pointer to QTABHEAD structure already filled out.

pBytesAqed

Pointer to BFU32. When the function returns it will contain the number of bytes that have been DMAed.

Returns

R64_OK	If successful.
Non-zero	If unsuccessful.

Comments This function returns the number of bytes of the current image that have been DMAed so far. The returned value is an instantaneous value that is accurate at the moment the board was checked. Since DMA can occur very quickly, the returned value may not be accurate for a very long. The value returned is also approximate, the granularity depends on the number of bytes transferred per quad (individual DMA instruction), which can vary from quad to quad. As a rule of thumb, this function usually is accurate to plus or minus one line's worth of bytes.

Calling this function in a loop is not a very efficient way to wait for a frame to be acquired. Use the signaling system instead. This function can be used to check the progress of the DMA and to find out how much new data is in the host memory.

41.8 R64Shutdown

Prototype R64RC R64Shutdown(R64 *Board*)

Description Aborts all DMA activity and acquisition on the board.

Parameters *Board*

Handle to board.

Returns

R64_OK	In all cases.

Comments This functions aborts all activity on the board. DMA is aborted. Acquisition is aborted. The board stops what it is currently doing and gets it ready for more acquisition. Normally this function does not need to be called.

41.9 R64ConIntModeSet

Prototype R64RC R64ConIntModeSet(R64 *Board*, BFU32 *Mode*)

Description Sets the interrupt mode for the board.

Parameters

Board

Handle to board.

Mode

The R64 has two interrupt modes:

BFIntModeDefault - Interrupts happen continually.

BFIntModeEOFAq - End of frame interrupts happen only when acquisition has started. The will be no interrupts during a freeze of acquisition.

Returns

R64_OK	In all cases.

Comments This function puts the board in a mode that is most useful when the board is in start-stop mode. When the board is in start-stop mode end of frame (EOF) interrupts are raised every time the trigger is de-asserted, even if the board is not currently acquiring. This can cause problems in applications that are tracking this interrupt. The solution to this problem is to use this function to put the board into BFInModeEOFAq. In this mode, the board only issues the EOF interrupt when the trigger de-asserts and the board is in grab mode.

41.10 R64ConIntModeGet

Prototype R64RC R64ConIntModeGet(R64 *Board*, PBFU32 *Mode*)

Description Gets the interrupt mode for the board.

Parameters

Board

Handle to board.

Mode

The interrupt mode returned can be one of the following:

BFIntModeDefault - Interrupts happen continually.

BFIntModeEOFAq - End of frame interrupts happen only when acquisition has started. The will be no interrupts during a freeze of acquisition.

Returns

R64_OK	In all cases.

Comments See the function R64ConIntModeSet for more information.

41.11 R64LutPeek

Prototype BFU32 R64LutPeek(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*)

Description Reads a single LUT value.

Parameters

Board

R64 board ID.

Mode

This parameter is ignored.

Bank

LUT bank:

BFLutBank0 - peek LUT bank 0.
BFLutBank1 - peek LUT bank 1.
BFLutBank2 - peek LUT bank 2.
BFLutBank3 - peek LUT bank 3.

Lane

One or more LUT lanes ORed together:

BFLutLane0 - peek LUT lane 0. Returned value will be 8 bits.
BFLutLane1- peek LUT lane 1. Returned value will be 8 bits.
BFLutLane2 - peek LUT lane 2. Returned value will be 8 bits.
BFLutLane3 - peek LUT lane 3. Returned value will be 8 bits.
BFLutTwoLanes - peek LUT lanes 0 and 2. Returned value will be 32 bits.
BFLutFourLanes - peek LUT lanes 0, 1, 2 and 3. Returned value will be 32 bits.

Addr

LUT address to read. Must be in the range 0 to 256.

Returns The LUT value. Bit depth of the returned value will depend on the *Lane* parameter.

Comments In order to use this function special firmware is needed to implement the look up table on the board. The LUT on the R64 has two banks, each consists of four lanes of 8-in/8-out look-up-tables.

41.12 R64LutPoke

Prototype R64RC R64LutPoke(RdRn *Board*, BFU8 *Mode*, BFU8 *Bank*, BFU8 *Lane*, BFU32 *Addr*, BFU32 *Value*)

Description Writes a single LUT value to one or more LUT lanes.

Parameters

Board

R64 board ID.

Mode

This parameter is ignored.

Bank

LUT bank:

BFLutBank0 - poke LUT bank 0.
BFLutBank1 - poke LUT bank 1.

Lane

One or more LUT lanes ORed together:

BFLutLane0 - poke LUT lane 0.
BFLutLane1- poke LUT lane 1.
BFLutLane2 - poke LUT lane 2.
BFLutLane3 - poke LUT lane 3.
BFLutTwoLanes - poke LUT lanes 0 and 2.
BFLutFourLanes - poke LUT lanes 0, 1, 2 and 3.

Addr

LUT address to write. Must be in the range 0 to 256.

Value

Value to write to LUT location *Addr*. Only the 8 MSBs of *Value* will be used.

Returns

R64_OK	Function succeeded.

Comments In order to use this function special firmware is needed to implement the look up table on the board. The LUT on the R64 has two banks, each consists of four lanes of 8-in/8-out look-up-tables.

41.13 R64ConGPOutSet

Prototype R64RC R64ConGPOutSet(R64 *Board*, BFU32 *GPout*, BFU32 *Level*)

Description Sets the bits on the General Purpose Output registers.

Parameters

Board

Handle to board.

GPout

Type of GPOut (GPOuts may be or'ed together - BFGPOut0|BFGPOut1):

BFGPOut0 - Set the the value of GPOut0.
BFGPOut1 - Set the the value of GPOut1.
BFGPOut2 - Set the the value of GPOut2.
BFGPOut3 - Set the the value of GPOut3.
BFGPOut4 - Set the the value of GPOut4.
BFGPOut5 - Set the the value of GPOut5.
BFGPOut6 - Set the the value of GPOut6.

Level

The level to set the bit(s) too. This value can be either a 0 or 1.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	*Level* was not a 0 or a 1.
R64_BAD_GPOUT	A invalid *GPout* value was passed into the function.
R64_CON_GPOUT_BAD	The setting of the register value failed.

Comments

41.14 R64ConGPOutGet

Prototype R64RC R64ConGPOutGet(R64 *Board*, PBFU32 *Value*)

Description Returns the value of the all the general purpose output bits.

Parameters

Board

Handle to board.

Value

A pointer to the value of the GPOut bits.

Returns

R64_OK	In all cases.

Comments Each digit in *Value* represents a GPOut, GPOut0 being the right most digit. For example, if *Value* has a value of 0x0000007e all the GPOuts have a value of 1 except GPOut0 which has a value of 0.

R64 Control Functions

Chapter 42

42.1 Introduction

These functions control how the R64 interfaces to the camera. In general, the registers on the board are set up with the camera configuration file, and these functions need not be called. However, if an application needs to make minor changes to the boards setup, it is often easier to call these functions than to switch between camera modes.

These functions are essentially wrappers around register reads and writes. Some users may find it easier to write directly to the registers as it more closely imitates modifying the camera configuration files.

42.2 R64ConVTrigModeSet

Prototype R64RC R64ConVTrigModeSet(R64 *Board*, BFU32 *TrigMode*, BFU32 TrigSelect, BFU32 *TrigPolarity*)

Description Sets the trigger mode and polarity for the acquisition engine.

Parameters *Board*

Handle to board.

TrigMode

Trigger mode for acquisition the engine:

BFTrigFreeRun – no trigger is used, board free runs.
BFTrigOneShot – one shot mode, for asynchronously resettable cameras.
BFTrigOneShotSelfTriggered – self triggering one shot mode.
BFTrigAqCmd - acquisition command is latched by trigger.
BFTrigSnapQualified - trigger causes snap command to be issue, application issues grab command to turn on this mode, freeze command turns off this mode.
BFTrigContinuousData – for continuous data sources.
BFTrigContinuousDataQualified – for continuous data sources with separate date qualifier.
BFTrigOneShotStartAStopA – variable length image acquisition controlled by trigger, one frame acquired per trigger assert.
BFTrigOneShotStartAStopALevel – variable length image acquisition controlled by trigger, capture continues as long as trigger stays asserted.

TrigSelect

Controls which type of trigger is to be used:

For Karbon-CL/Alta/Neon/R64

BFTrigDiff - Differential trigger
BFTrigTTL - TLL trigger
BFTrigOpto - Opto-isolated trigger
BFTrigFVAL - Trigger is the FVAL signal on the CL cable
BFTrigNTG - NTG is trigger source
BFIgnore - No change to trigger selection

For Karbon-CXP

BFTrigDiff - Differential trigger
BFTrigTTL - TLL trigger
BFTrigNTG - NTG is trigger source
BFTrigVFG0TrigSel - Use same trigger as VFG 0
BFTrigButton - Trigger is button

BFTrigCXPTriggerIn - Trigger is CXP trigger (from camera)
BFTrigSWTrigger - Trigger is software trigger
BFTrigScanStep - Trigger comes from quad. encoder circuit in scan step mode
BFTrigNTGVFG0 - Trigger is NTG of VFG 0
BFIgnore - No change to trigger selection

TrigPolarity

Polarity for trigger:

BFTrigAssertedHigh – TRIGGER is asserted on rising edge.
BFTrigAssertedLow – TRIGGER is asserted on falling edge.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	One of the parameters is not valid

Comments

This function works in conjunction with the camera configuration files. It is important to understand that not all cameras support all triggering modes. Usually a particular camera will only support one or two triggering modes. Furthermore, a different camera configuration file is usually needed for each triggering mode. For example, a camera will almost always have a free running configuration file, useful for set up and offline testing. A camera may also have a one shot file, which would be used in time-critical applications. You cannot usually put the board, set up by the free running file, into one shot mode because the latter mode requires special triggering signals to be sent to the camera. However, you can put the board, set up by a one shot file, into self triggering one shot mode. This is useful for camera set up and system debugging.

The exception to the paragraph above is the triggered acquire command mode, which will work with all cameras. This mode is really no different than just issuing an acquisition command at a specific point in time in the future. When the board is in this mode, an acquisition command is written by the host but not latched. Basically, the board is armed but does not acquire any data. When the trigger is asserted the command latches. Once the command is latched, it acts as it normally does, that is, the board starts acquiring data at the start of the next frame from the camera. The only acquisition commands that are affected are snap and grab. The freeze and abort commands work normally, and do not need a trigger to be latched. The disadvantage of this mode is that it can add up to a frame time of latency to any trigger, because the camera's timing is not being reset.

For more information on the acquisition and trigger on the R64, see the Hardware Reference Manual.

If you want to find out what mode the board is in, call the function R64ConVTrigModeGet.

Note: This function only controls how the board is vertically triggered. Vertical triggers cause the board to acquire a whole frame from an area camera or a number of lines from a line scan camera. You must enable the connection of the external trigger with

the acquisition engines with the function R64ConExTrigConnect. The software triggers are always available. Not all combinations of TrigMode and TrigAssignments are possible.

42.3 R64ConVTrigModeGet

Prototype R64RC R64ConVTrigModeGet(R64 *Board*, PBFU32 *TrigMode*, PBFU32 *TrigSelect*, PBFU32 *TrigPolarity*)

Description Gets the current trigger mode and polarities for both acquisition engines.

Parameters *Board*

Handle to board.

TrigMode

Returns the current trigger mode for the acquisition engine:

BFTrigFreeRun – no trigger is used, board free runs.
BFTrigOneShot – one shot mode, for asynchronously resettable cameras.
BFTrigOneShotSelfTriggered – self triggering one shot mode.
BFTrigAqCmd - acquisition command is latched by trigger.
BFTrigSnapQualified - trigger causes snap command to be issue, application issues grab command to turn on this mode, freeze command turns off this mode.
BFTrigContinuousData – for continuous data sources.
BFTrigContinuousDataQualified – for continuous data sources with separate date qualifier.
BFTrigOneShotStartAStopA – variable length image acquisition controlled by trigger, one frame acquired per trigger assert.
BFTrigOneShotStartAStopALevel – variable length image acquisition controlled by trigger, capture continues as long as trigger stays asserted.

TrigSelect

Returns the current type of trigger being used by the acquisition engine:

For Karbon-CL/Alta/Neon/R64

BFTrigDiff - Differential trigger
BFTrigTTL - TLL trigger
BFTrigOpto - Opto-isolated trigger
BFTrigFVAL - Trigger is the FVAL signal on the CL cable
BFTrigNTG - NTG is trigger source

For Karbon-CXP

BFTrigDiff - Differential trigger
BFTrigTTL - TLL trigger
BFTrigNTG - NTG is trigger source
BFTrigVFG0TrigSel - Use same trigger as VFG 0
BFTrigButton - Trigger is button
BFTrigCXPTriggerIn - Trigger is CXP trigger (from camera)

BFTrigSWTrigger - Trigger is software trigger
BFTrigScanStep - Trigger comes from quad. encoder circuit in scan step mode
BFTrigNTGVFG0 - Trigger is NTG of VFG 0

TrigPolarity

Returns the current polarity for the trigger:

BFTrigAssertedHigh - trigger A is asserted on rising edge.
BFTrigAssertedLow - trigger A is asserted on falling edge.

Returns

R64_OK	In all cases.

Comments

This function returns the current state of the trigger circuitry for the acquisition engine. See the function R64ConVTrigModeSet for a complete description of the modes.

42.4 R64ConHTrigModeSet

Prototype BFRC R64ConHTrigModeSet(Bd *Board*, BFU32 *EncMode*, BFU32 *EncPolarity*, BFU32 *EncSelect*)

Description Sets the horizontal trigger mode and polarities for the acquisition engine.

Parameters

Board

Handle to board.

EncMode

The horizontal triggering mode:

BFEncFreeRun – no line trigger is used, board free runs.
BFEncOneShot – horizontal one shot mode, every line needs a line trigger.
BFEncOneShotSelfTriggered – self triggering one shot mode..

EncPolarity

Polarity for all line triggers:

BFEncAssertedHigh - line triggers are asserted on rising edge.
BFEncAssertedLow - line triggers are asserted on falling edge.

EncSelect

Type of encoder:

For Karbon-CL/R64/Neon:

BFEncTTL - Single ended TTL level encoder
BFEncDiff - Differential (LVDS) encoder
BFEncOpto - Optocoupled encoder

For Karbon-CXP:

BFEncTTL - Single ended TTL level encoder
BFEncDiff - Differential (LVDS) encoder
BFEncVFG0EncASel - Selected encoder on VFG0
BFEncNTG - NTG is encoder
BFEncButton - The boards buttong is the encoder
BFEncCXPTriggerIn - CXP trigger is the encoder (from camera)
BFEncSWEncoderA - Software encocder A is the encoder
BFEncNTGVFG0 - NTG from VFG0

Returns

R64_OK	If successful.

R64_BAD_CON_PARAM	One of the parameters is not valid or the particular combination of parameters is not possible.

Comments

42.5 R64ConHTrigModeGet

Prototype BFRC R64ConHTrigModeGet(Bd *Board*, PBFU32 *EndMode*, PBFU32 *EncPolarity*, PBFU32 *EncSelect*)

Description Gets the current horizontal encoder mode and polarity of the encoder.

Parameters

Board

Handle to board.

EncMode

Returns the current encoder mode:

BFEncFreeRun – no line trigger is used, board free runs.
BFEncOneShot – horizontal one shot mode, every line needs a line trigger.

EncPolarity

Returns the current polarity for the encoder:

BFEncAssertedHigh - trigger A is asserted on rising edge.
BFEncAssertedLow - trigger A is asserted on falling edge.

EncSelect

Returns the encoder input type:

For Karbon-CL/R64/Neon:

BFEncTTL - Single ended TTL level encoder
BFEncDiff - Differential (LVDS) encoder
BFEncOpto - Optocoupled encoder

For Karbon-CXP:

BFEncTTL - Single ended TTL level encoder
BFEncDiff - Differential (LVDS) encoder
BFEncVFG0EncASel - Selected encoder on VFG0
BFEncNTG - NTG is encoder
BFEncButton - The boards buttong is the encoder
BFEncCXPTriggerIn - CXP trigger is the encoder (from camera)
BFEncSWEncoderA - Software encocder A is the encoder
BFEncNTGVFG0 - NTG from VFG0

Returns

R64_OK	In all cases.

Comments

42.6 R64ConSwTrig

Prototype R64RC R64ConSwTrig(R64 *Board*, BFU32 AssertType)

Description Trips software trigger.

Parameters *Board*

Handle to board.

AssertType

BFTrigAssert - Sets the trigger high.
BFTrigDeassert - Sets the trigger low.

Returns

R64_OK	Function succeeded.
R64_BAD_CON_PARAM	Invalid AssertType.

Comments Register REG_SW_TRIG is modified by this function.

42.7 R64ConSwTrigStat

Prototype R64RC R64ConSwTrigStat(R64 *Board*, PBFU32 *Status*)

Description Returns the status of the software trigger.

Parameters *Board*

Handle to board.

Status

The status of the software trigger can be:

BFTrigLow - Trigger is low.
BFTrigHigh - Trigger is high.

Returns

R64_OK	In all cases.

Comments This function returns the status of the software trigger at the moment that the function is called.

42.8 R64ConHwTrigStat

Prototype R64RC R64ConHWTrigStat(R64 *Board*, PBFU32 *Status*)

Description Returns the status of the hardware trigger.

Parameters

Board

Handle to board.

Status

The status of the hardware trigger can be:

BiTrigLow - Trigger is low.

BiTrigHigh - Trigger is high.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	Couldn't determine the trigger type being used.

Comments This function returns the status of the hardware trigger at the moment that the function is called.

42.9 R64ConExTrigConnect

Prototype R64RC R64ConExTrigConnectt(R64 *Board*, BFU32 *Mode*)

Description Enables or disables the external trigger.

Parameters *Board*

Handle to board.

Mode

The external trigger can be:

BFExTrigConnect - Enables the external trigger.

BFExTrigDisconnect - Disables the external trigger.

Returns

R64_OK	If successful.
R64_BAD_CON_PARAM	Invalid mode parameter.

Comments This function enables or disables the external trigger based on the mode parameter.

42.10 R64ConExTrigStatus

Prototype R64RC R64ConExTrigStatus(R64 *Board*, PBFU32 *Mode*)

Description Returns the enabled status of the external trigger.

Parameters

Board

Handle to board.

Mode

The mode of the external trigger can be:

BFExTrigConnect - The external trigger is enabled.

BFExTrigDisconnect - The external trigger is disabled.

Returns

R64_OK	In all cases.

Comments This function reads the EN_TRIGGER register to determine if the external trigger is enabled or disabled. If the register is set to a one, the external trigger is enabled. Otherwise the external trigger is disabled.

To enable or disable the external trigger see function R64ConExTrigConnect.

42.11 R64ConFreqSet

Prototype R64RC R64ConFreqSet(R64 *Board*, BFU8 *Freq*)

Description Sets the R64 internal clock generator frequency.

Parameters *Board*

Handle to board.

Freq

Internal clock frequency:

R64Freq000 - set to 0.0 MHz
R64Freq037 - set to 3.75 MHz
R64Freq075 - set to 7.5 MHz
R64Freq150 - set to 15.0 MHz
R64Freq240 - set to 24.0 MHz
R64Freq300 - set to 30.0 MHz
R64Freq480 - set to 48.0 MHz
R64Freq600 - set to 60.0 MHz

Returns

R64_OK	Function succeeded.
R64_BAD_FREQ	Illegal clock frequency.
R64_CON_FREQ_ERR	Frequency switch failed.

Comments R64 clock frequencies are declared in R64Def.h.

This clock does not affect board operation, it is only provided for cameras that need an external master clock.

Register REG_CFREQ is the only register that may be modified by this function.

42.12 R64ConGPOutSet

Prototype R64RC R64ConGPOutSet(R64 *Board*, BFU8 *Value*, BFU32 *Level*)

Description Sets the level of the GPOUT outputs.

Parameters *Board*

Handle to board.

Value

One or more outputs ORed together:

R64GPOut0 - general output 0
R64GPOut1 - general output 1
R64GPOut2 - general output 2
R64GPOut3 - general output 3
R64GPOut4 - general output 4
R64GPOut5 - general output 5
R64GPOut6 - general output 6
R64GPOut7 - general output 7
R64GPOut8 - general output 8
R64GPOut9 - general output 9
R64GPOut10 - general output 10
R64GPOut11 - general output 11

Level

0 - Sets the GPOUT(s) to zero (low).
1 - Sets the GPOUT(s) to one (high).

Returns

R64_OK	Function succeeded.
R64_BAD_GPOUT	Illegal general purpose output pin number.
R64_CON_GPOUT_ERR	Output pin set failed.

Comments This function sets the level of one or more GPOUTs.

42.13 R64ConGPOutGet

Prototype R64RC R64ConGPOutSet(R64 *Board*, BFU8 *Value*)

Description Gets the current state of the GPOUTs

Parameters *Board*

Handle to board.

Value

One or more outputs ORed together who state is current set (i.e. 1):

R64GPOut0 - general output 0
R64GPOut1 - general output 1
R64GPOut2 - general output 2
R64GPOut3 - general output 3
R64GPOut4 - general output 4
R64GPOut5 - general output 5
R64GPOut6 - general output 6
R64GPOut7 - general output 7
R64GPOut8 - general output 8
R64GPOut9 - general output 9
R64GPOut10 - general output 10
R64GPOut11 - general output 11

Returns

R64_OK	Function succeeded.

Comments This function returns all of the GPOUTs whose current state is 1.

42.14 R64LastLine

Prototype R64RC R64LastLine(R64 *Board*, PBFU32 *pCurLine*)

Description Returns the line number of the last line in the frame.

Parameters

Board

Handle to board.

pCurLine

Pointer to the last line number.

Returns

R64_OK	In all cases.

Comments This functions returns the line number of the last line in the frame. The returned value is actually the Vertical CTAB counter value for the last line. If the camera being used is a line scan camera then this value will be equivalent to the line number. However, for area scan camera the start of the vertical active region will have to be subtracted from the returned value (usually the vertical active region starts at 0x1000).

This function is most useful when acquiring variable sized images and thus the frame size is unknown. This function will return the value from the last frame up until the end of the following frame. In other words, the value of the last line stays constant for the entire duration of the next frame. Once the next frame ends, then the last line is the value for that frame.

42.15 R64ConExposureControlSet

Prototype BFRC R64ConExposureControlSet(Bd *Board*, BFDOUBLE *ExposurePeriod*, BFDOUBLE *LineFramePeriod*, BFU32 *TriggerMode*, BFBOOL *AssertedHigh*, BFU32 *OutputSignal*)

Description Programs the New Timing Generator (NTG), used to create waveforms to control the line/frame rate and exposure time of cameras.

Parameters

Board

Handle to board.

ExposurePeriod

The desired exposure period in milliseconds

Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)

LineFramePeriod

The desire line/frame rate period in milliseconds.

Note: This parameter is floating point and you can pass in non-whole number values (e.g. 10.523)

TriggerMode

The triggering mode for the timing generator. Must be one of the following:

BFNTGModeFreeRun - Timing generator is free running.
BFNTGModeOneShotTrigger - Timing generator is in one-shot mode, triggered by the board's trigger input.
BFNTGModeOneShotEncoder - Timing generator is in one-shot mode, triggered by the board's encoder input.

AssertedHigh

The level of the timing generator's output waveform. Must be:

TRUE - Waveform is asserted high.
FALSE - Waveform is asseted low.

OutputSignal

The output pins that the waveform will be output on. Can be one more more of the following ORed together (signal will be output on all pins selected by this parameter):

The outputs that the waveform will be output on. Can be one or more of the following ORed together (signal will be output on all pins selected by this parameter):

For the Karbon-CL/Neon/Alta:

BFNTGOuputCC1 - Output on the CC1 signal on CL connector.
BFNTGOuputCC2 - Output on the CC2 signal on CL connector.
BFNTGOuputCC3 - Output on the CC3 signal on CL connector.
BFNTGOuputCC4 - Output on the CC4 signal on CL connector.
BFNTGOutputGP0 - Output on GPOUT0 on the I/O connector.
BFNTGOutputGP1 - Output on GPOUT1 on the I/O connector.
BFNTGOutputGP2 - Output on GPOUT2 on the I/O connector.
BFNTGOutputGP3 - Output on GPOUT3 on the I/O connector.
BFNTGInputTrig - Output goes to Trigger input.
BFNTGInputEncA - Output goes to Encoder A input.

For the Karbon-CXP

BFNTGOuputCC1 - Output on the CC1 signal on CL connector.
BFNTGOuputCC2 - Output on the CC2 signal on CL connector.
BFNTGOuputCC3 - Output on the CC3 signal on CL connector.
BFNTGOuputCC4 - Output on the CC4 signal on CL connector.
BFNTGInputTrig - Output goes to Trigger input.
BFNTGInputEncA - Output goes to Encoder A input.
BFNTGInputEncB - Output goes to Encoder B input.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The current board family does not support the NTG.
R64_NTG_NOT_SUPPORTED	The installed frame grabber does not support the NTG or the current firmware does not currently support the NTG (contact BitFlow for more informatino).
R64_NTG_EXP_OUT_OF_ RANGE	The requested values are out of range of the timing generator
R64_NTG_EXP_GT_LF	The requested exposure time is longer than the requested line/frame period.
R64_NTG_UNKNOWN_MODE	The requested mode triggering mode is not known.

Comments

This function is used to program the New Timing Generator (NTG) avialable on modern BitFlow boards (Karbon, Neon, Alta and newer). The timing generator is used to control the line/frame rate and exposure time of attached cameras.

The Exposure time is controlled by the *ExposurePeriod* parameter. This parameter takes a floating point value in units of milliseconds. The line/frame rate is controlled by the *LineFrameRate* parameter. This parameter is also floating point and the units are in milliseconds. Note that although this parameter controls the line/frame rate, it is not in units of Hertz, which it would be if this parameter was the line/frame frequency. Instead this parameters controls the line/frame period, units of time. Refer to the hardware manual of your frame grabber to see the range for these parameters.

The triggering of the NTG is independent of the triggering configuration of the rest of the frame grabber. The NTG is fully independent all other components of the frame grabber, and runs completely on its own timing. The NTG can be triggered either by the currently selected trigger input or the currently selected encoder input.

The waveform of the NTG can be routed to one or more outputs. The parameter *OutputSignal* controls which outputs get the waveform. This parameter can take one or more of the defined outputs ORed together. The waveform will appear on all outputs simultaneously selected by this parameter.

The current status of the NTG can be retrieved using the R64ConExposureControlGet function.

Please refer to the hardware manual of your board for more detailed information on how this timing generator works.

42.16 R64ConExposureControlGet

Prototype BFRC R64ConExposureControlSet(Bd *Board*, PBFDOUBLE *pExposurePeriod*, PBFDOUBLE *pLineFramePeriod*, PBFU32 *pTriggerMode*, PBFBOOL *pAssertedHigh*, PBFU32 *pOutputSignal*)

Description Retrieve the current parameters of the New Timing Generator (NTG).

Parameters

Board

Handle to board.

pExposurePeriod

Pointer to a double, returns the current exposure period in milliseconds

Note: This parameter is floating point and can be a in non-whole number values (e.g. 10.523)

pLineFramePeriod

Pointer to a double, returns the current line/frame rate period in milliseconds.

Note: This parameter is floating point and can be in non-whole number values (e.g. 10.523)

pTriggerMode

Pointer to a BFU32, returns the current triggering mode for the timing generator. Will be one of the following:

BFNTGModeFreeRun - Timing generator is free running.
BFNTGModeOneShotTrigger - Timing generator is in one-shot mode, triggered by the board's trigger input.
BFNTGModeOneShotEncoder - Timing generator is in one-shot mode, triggered by the board's encoder input.

pAssertedHigh

Pointer to a BFU32, returns the current the current level of the timing generator's output waveform. Will be:

TRUE - Waveform is asserted high.
FALSE - Waveform is asseted low.

pOutputSignal

Pointer to a BFU32, returns the current output pins that the waveform will be output on. Will be one more more of the following ORed together:

The outputs that the waveform will be output on. Can be one or more of the following ORed together (signal will be output on all pins selected by this parameter):

For the Karbon-CL/Neon/Alta:

BFNTGOuputCC1 - Output on the CC1 signal on CL connector.
BFNTGOuputCC2 - Output on the CC2 signal on CL connector.
BFNTGOuputCC3 - Output on the CC3 signal on CL connector.
BFNTGOuputCC4 - Output on the CC4 signal on CL connector.
BFNTGOutputGP0 - Output on GPOUT0 on the I/O connector.
BFNTGOutputGP1 - Output on GPOUT1 on the I/O connector.
BFNTGOutputGP2 - Output on GPOUT2 on the I/O connector.
BFNTGOutputGP3 - Output on GPOUT3 on the I/O connector.
BFNTGInputTrig - Output goes to Trigger input.
BFNTGInputEncA - Output goes to Encoder A input.

For the Karbon-CXP

BFNTGOuputCC1 - Output on the CC1 signal on CL connector.
BFNTGOuputCC2 - Output on the CC2 signal on CL connector.
BFNTGOuputCC3 - Output on the CC3 signal on CL connector.
BFNTGOuputCC4 - Output on the CC4 signal on CL connector.
BFNTGInputTrig - Output goes to Trigger input.
BFNTGInputEncA - Output goes to Encoder A input.
BFNTGInputEncB - Output goes to Encoder B input.

Returns

CI_OK	If successful.
CISYS_ERROR_BAD_ BOARDPTR	An invalid board handle was passed to the function.
CISYS_ERROR_NOTSUP-PORTED	The current board family does not support the NTG.
R64_NTG_NOT_SUPPORTED	The installed frame grabber does not support the NTG or the current firmware does not currently support the NTG (contact BitFlow for more informatino).

Comments

This function is retrieves the current status of the New Timing Generator (NTG) avialable on modern BitFlow boards (Karbon, Neon, Alta and newer). The timing generator is used to control the line/frame rate and exposure time of attached cameras.

The NTG can be programmed using the R64ConExposureControlSet function.

R64 Control Tables

Chapter 43

43.1 Introduction

These functions allow an application to write directly to the control tables (CTAB) on the R64. Normally the CTABs are initialized from a camera configuration file. However, because the CTABs control things like frame rate and exposure time, an application may want to modify them on-the-fly.

43.2 Modifying CTABS from Software

It is often necessary for an application to modify the CTABs dynamically, based on user input or as the result of a calculation on image data. The SDK provides functionality for modifying the CTABs from an application. The primary function used to program the CTAB is "R64CtabFill()". This function writes a masked value to the R64's camera control table. The following sections illustrate the use of "R64CtabFill()" to modify CTABs.

43.3 Example Code Showing Modifying The CTabs From Software

The code below is a function to program the CTabs for a color line scan camera that has four independent exposure signals. One signal controls the overall line rate, and the other three control the exposure for the Red, Green and Blue channels. The function take four parameters which are used to control these four signals. The units of the parameters are in CTab clocks, which is equivalent to pixel clock divided by eight.

For the sake of readability, the function below was coded very simply. The downside is that it is not very efficient in execution. The R64 CTabs are quite large, and clearing them as is done four times below can take a long time. A more efficient approach would be to retain the value of the parameters in static variables, and when this function is called, only program CTabs with what has changed from the previous call.

```
BFU32 R64TVISetExposures(Bd hBoard, BFU32 NewFrame, BFU32
RedCommon, BFU32 Green, BFU32 Blue)
{
   BFU32 NewFrameStart,NewFrameSize;
   BFU32 RStart, RSize;
   BFU32 GStart, GSize;
   BFU32 BStart, BSize;

   /*
   *  Calculate starts and sizes
   */

   NewFrameStart = NewFrame;
   NewFrameSize = 1;

   RStart = NewFrame - RedCommon + 1;
   RSize = RedCommon + Safety;

   GStart = NewFrame - Green  + 1;
   GSize = Green + Safety;

   BStart = NewFrame - Blue + 1;
   BSize = Blue + Safety;

   /*
   * Program the CTABs
   */

   // stop CTABs
   BFRegPoke(hBoard,REG_CTABHOLD,1);

   // program GPHO (controls CT0 = NewFrame)
   R64CTabFill(hBoard,0,R64HCTABSIZE,R64HCTabGPH0,0xffff);
   R64CTabFill(hBoard,0,R64VCTABSIZE,R64VCTabGPV0,0xffff);
   R64CTabFill(hBoard,NewFrameStart,NewFrameSize,R64HCTabGPH0,0)
;
```

```
    // program GPH1 (controls CT1 = Red Integration)
    R64CTabFill(hBoard,0,R64HCTABSIZE,R64HCTabGPH1,0);
    R64CTabFill(hBoard,0,R64VCTABSIZE,R64VCTabGPV1,0xffff);
    R64CTabFill(hBoard,RStart,RSize,R64HCTabGPH1,0xffff);

    // program GPH2 (controls CT2 = Green Integration)
    R64CTabFill(hBoard,0,R64HCTABSIZE,R64HCTabGPH2,0);
    R64CTabFill(hBoard,0,R64VCTABSIZE,R64VCTabGPV2,0xffff);
    R64CTabFill(hBoard,GStart,GSize,R64HCTabGPH2,0xffff);

    // program GPH3 (controls CT3 = Blue Integration)
    R64CTabFill(hBoard,0,R64HCTABSIZE,R64HCTabGPH3,0);
    R64CTabFill(hBoard,0,R64VCTABSIZE,R64VCTabGPV3,0xffff);
    R64CTabFill(hBoard,BStart,BSize,R64HCTabGPH3,0xffff);

    // allow CTABs to run
    BFRegPoke(hBoard,REG_CTABHOLD,0);

    return 0;
}
```

43.4 R64CTabPeek

Prototype BFU16 R64CTabPeek(R64 *Board*, BFU32 *Index*, BFU16 *Mask*)

Description Reads a single masked value from the R64 Camera Control Table.

Parameters *Board*

R64 board ID.

Index

CTAB table offset.

0 - 0x8000 for horizontal CTABs
0 - 0x20000 for vertical CTABs

Mask

CTAB bit extraction mask.

R64CTab
R64HCTab
R64VCTab
R64HCTabHStart
R64HCTabHReset
R64HCTabENHLoad
R64HCTabReserved
R64HCTabGPH0
R64HCTabGPH1
R64HCTabGPH2
R64HCTabGPH3
R64VCTabVStart
R64VCTabVReset
R64VCTabENVLoad
R64VCTabIRQ
R64VCTabGPV0
R64VCTabGPV1
R64VCTabGPV2
R64VCTabGPV3

Returns A single masked CTAB entry.

Comments CTAB bit masks and other definitions are declared in “R64Def.h”.

Example

Check for a horizontal start bit in the horizontal CTAB at the horizontal load point location 0x2000.

```
BFU32  HStartBit
HStartBit = R64CTabPeek(Board, 0x2000, R64HCTabHStart)
```

43.5 R64CTabPoke

Prototype R64RC R64CTabPoke(R64 *Board*, BFU32 *Index*, BFU16 *Mask*, BFU16 *Value*)

Description Writes a single masked value from the R64 Camera Control Table.

Parameters

Board

R64 board ID.

Index

CTAB table offset.

Mask

CTAB bit extraction mask (see R64CtabPeek).

Value

CTAB value.

Returns

R64_OK	Function succeeded.
R64_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R64_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R64_CTAB_POKE_ERR	CTAB poke failed.

Comments CTAB bit masks and other definitions are declared in "R64Def.h".

Example

Write a vertical reset bit in the vertical CTAB after 0x1000 lines.

```
R64CTabPoke(Board, 0x1000, R64VCTabVReset, 0xffff)
```

43.6 R64CTabRead

Prototype R64RC R64CTabRead(R64 *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, PBFVOID *pDest*)

Description Reads masked CTAB values from the R64 Camera Control Table.

Parameters

Board

R64 board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to read.

Mask

CTAB bit extraction mask (see R64CtabPeek).

pDest

Pointer to CTAB table storage (32 bits per entry).

Returns

R64_OK	Function succeeded.
R64_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R64_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R64_CTAB_READ_ERR	CTAB read failed.

Comments CTAB bit masks and other definitions are declared in "R64Def.h".

Example

Read 0x100 values from the vertical CTAB starting at the zero.

```
BFU32  VTab[0x100]
R64CTabRead(Board, 0, 0x100, R64VCTab, &VTab[0])
```

43.7 R64CTabWrite

Prototype R64RC R64CTabWrite(R64 *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, PBFVOID *pSource*)

Description Writes masked CTAB values from the R64 Camera Control Table.

Parameters

Board

R64 board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to read.

Mask

CTAB bit extraction mask (see R64CtabPeek).

pSource

CTAB entries to write (32 bits per entry).

Returns

R64_OK	Function succeeded.
R64_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R64_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R64_CTAB_WRITE_ERR	CTAB write failed.

Comments CTAB bit masks and other definitions are declared in “R64Def.h”.

Example

Write an entire horizontal CTAB to the R64.

```
BFU32  HTab[R64HCTABSIZE]
R64CTabWrite(Board, 0, R64HCTABSIZE, R64HCTab, &HTab[0])
```

43.8 R64CTabFill

Prototype R64RC R64CTabFill(R64 *Board*, BFU32 *Index*, BFU32 *NumEntries*, BFU16 *Mask*, BFU16 *Value*)

Description Writes a masked CTAB fill value from the R64 Camera Control Table.

Parameters

Board

R64 board ID.

Index

CTAB table offset.

NumEntries

Number of CTAB values to write.

Mask

CTAB bit extraction mask (see R64CtabPeek).

Value

CTAB fill value to write.

Returns

R64_OK	Function succeeded.
R64_BAD_HCTAB_ADDR	Illegal horizontal CTAB address.
R64_BAD_VCTAB_ADDR	Illegal vertical CTAB address.
R64_CTAB_FILL_ERR	CTAB fill failed.

Comments CTAB bit masks and other definitions are declared in "R64Def.h".

Example

Clear the horizontal and vertical CTAB tables.

```
R64CTabFill(Board, 0, R64HCTABSIZE, R64HCTab, 0x0000)
R64CTabFill(Board, 0, R64VCTABSIZE, R64VCTab, 0x0000)
```

R64 Dual Port Memory

Chapter 44

44.1 Introduction

These functions allow an application to write and read directly to the dual port memory (DPM) on the R64. These function are only used in special circumstances and are not needed for ordinary acquisition.

44.2 R64DPMPeek

Prototype BFU32 R64DPMPeek(R64 *Board*, BFU32 *Offset*)

Description Read a single 32-bit value from the R64 DPM.

Parameters *Board*

R64 board ID.

Offset

Offset into DPM.

Returns A single 32-bit DPM entry.

Comments

44.3 R64DPMPoke

Prototype R64RC R64DPMPoke(R64 *Board*, BFU32 *Offset*, BFU32 *Value*)

Description Write a single 32-bit value to the R64 DPM.

Parameters

Board

R64 board ID.

Offset

Offset into DPM.

Value

The value to place into the DPM.

Returns

R64_OK	Function succeeded.
Non-zero	On error.

Comments

44.4 R64DPMRead

Prototype R64RC R64DPMRead(R64 *Board*, BFU32 *Offset*, BFU32 *NumEntries*, PBFVOID *pDest*)

Description Read 32-bit DPM values from the R64 DPM.

Parameters

Board

R64 board ID.

Offset

Offset into DPM.

NumEntries

Number of DPM values to read.

pDest

Pointer to storage for the DPM values (32 bits per entry).

Returns

R64_OK	Function succeeded.
Non-zero	On error.

Comments

44.5 R64DPMWrite

Prototype R64RC R64DPMWrite(R64 *Board*, BFU32 *Offset*, BFU32 *NumEntries*, PBFVOID *pSource*)

Description Write 32-bit DPM values to the R64 DPM.

Parameters

Board

R64 board ID.

Offset

Offset into DPM.

NumEntries

Number of DPM values to write.

pSource

DPM entries to write (32 bits per entry).

Returns

R64_OK	Function succeeded.
Non-zero	On error.

Comments

44.6 R64DPMFill

Prototype R64RC R64DPMFill(R64 *Board*, BFU32 *Offset*, BFU32 *NumEntries*, BFU32 *Value*)

Description Write a 32-bit DPM fill value to the R64 DPM.

Parameters

Board

R64 board ID.

Offset

Offset into DPM

NumEntries

Number of DPM values to write.

Value

DPM fill value to write.

Returns

R64_OK	Function succeeded.
Non-zero	On error.

Comments

44.7 R64DPMRamp

Prototype R64RC R64DPMRamp(R64 *Board*, BFU32 *StartOffset*, BFU32 *EndOffset*, BFU32 *StartVal*, BFU32 *EndVal*)

Description Write a 32-bit ramp to the R64 DPM.

Parameters

Board

R64 board ID.

StartOffset

DPM start offset.

EndOffset

DPM end offset.

StartVal

DPM start value.

EndVal

DPM end value.

Returns

R64_OK	Function succeeded.
Non-zero	On error.

Comments

44.8 R64DPMReadDMA

Prototype R64RC R64DPMReadDMA(R64 *Board*, BFU32 *Offset*, BFU32 *NumEntries*, PBFVOID *pDest*)

Description Read 32-bit DPM values from the R64 DPM using DMA.

Parameters

Board

R64 board ID.

Offset

Offset into DPM

NumEntries

Number of DPM values to read.

Value

Pointer to storage for the DPM values (32 bits per entry).

Returns

R64_OK	Function succeeded.
Non-zero	On error.

Comments

Camera Link Specification Serial Interface

Chapter 45

45.1 Introduction

The functions described in this chapter can be used with any BitFlow board with a camera link interface (Road Runner CL, R3-CL, R64-CL, Neon-CL, Karbon-CL and Axion-CL). These functions are used to send and receive commands to and from the camera, over the serial communications portion of the Camera Link cable. These functions directly control the UART (serial communications chip) on the BitFlow board. They cannot be used to communicate with the standard COM ports in the computer.

The functions in this chapter are defined by the Camera Link Specification revision 1.1. BitFlow has added some non-standard functions which are more convenient to use in certain circumstances. These functions start with the pre-fix clBF.

Normal program flow would use the functions in the following order to communicate with the camera without acquiring data from the camera. First the serial device needs to be initilized. Once clSerialInit returns successfully the read and write functions may be called. When communication has ended the serial device will need to be closed. The following is a simple example:

```
// declare serial reference instance
void serialRefPtr;
// initialize the first serial device
clSerialInit(0, &serialRefPtr);
// Write data to camera
clSerialWrite(serialRefPtr, transmit_buf, numofchar, 100);
// Read data from the camera
clSerialRead(serialRefPtr, receive_buf, &BufReadSize,100);
// Close the serial device
clSerialClose(serialRefPtr);
```

For a more complete example of how to use the CL API please refer to the BFCom example that comes with BitFlow's SDK.

45.1.1 BitFlow Specific Serial Functions

In addition to the functions required by the Camera Link specification, BitFlow has added a few extra helper functions. The functions start with the prefix "clBF". These function can only be used with BitFlow frame grabbers. These functions require a separate BitFlow serial reference pointer, which is a different reference pointer from the regular SL serial functions (the reason involves the middleware DLL which obfuscates serial reference handles). There are two ways to get this handle, either from the port number (as is used in clSerialInit()) or from a board handle use for the rest of the BitFlow API. The following shows how these functions are used.

For the first example, the board is opened and initialize, then the serial handles are retrieved based on the board handle:

```
void* serialRefPtr; // CL serial handle
void* serialBFRefPtr; // BitFlow serial handle

// Open and initialize the board
CiBrdOpen(&entry, &hBoard, BFSysInitialize);
// Get the CL serial handle from the board handle
clBFSerialInitFromBoardHandle(hBoard, &serialRefPtr);
// Get the BitFlow Serial handle from the board handle
clBFGetSerialRefFromBoardHandle(hBoard, &serialBFRefPtr);

// use the serial functions

// Write data using the standard CL function
clSerialWrite(serialRefPtr, Message, &Size, 100);

// Read whatever characters have come in using BF function
clBFSerialRead(serialBFRefPtr, Buf, &BufSize);
```

In the next example, the board is not open nor initialized, in this case we just want to open the serial port and do some serial I/O.

```
void* serialRefPtr; // CL serial handle
void* serialBFRefPtr; // BitFlow serial handle

// Open the CL serial port and get a CL serial handle
clSerialInit(SerNum, &serialRefPtr);
// Get the CL serial handle from the board handle
clBFGetSerialRef(SerNum, &serialBFRefPtr);

// use the serial functions

// Write data using the standard CL function
clSerialWrite(serialRefPtr, Message, &Size, 100);

// Read whatever characters have come in using BF function
clBFSerialRead(serialBFRefPtr, Buf, &BufSize);
```

45.2 clFlushPort

Prototype CLINT32 clFlushPort(hSerRef *serialRef*)

Description This function discards any bytes that are available in the input buffer.

Parameters *serialRef*

The value obtained by the clSerialInit function that describes the port to be flushed.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
BFCL_ERROR_FLUSH_PORT	An error occurred trying to flush the port.

Comments

45.3 clGetErrorText

Prototype CLINT32 clGetErrorText(const CLINT8* *manuName*, CLINT32 *errorCode*, CLINT8* *errorText*, CLUINT32* *errorTextSize*)

Description This function converts an error code to error text which can be displayed in a dialog box or in the standard I/O window.

Parameters

manuName

The manufacturer name in a NULL-terminated buffer. Manufacturer name is returned from clGetPortInfo.

errorCode

The error code used to look up the appropriate error text. This code can be returned from any function in this library.

errorText

A caller allocated buffer which will contain a NULL terminated error description on return.

errorTextSize

As an input, this value is the size, in bytes, of the errorText buffer that is passed in. On success, this value is the number of bytes that have been written into the buffer, including the null termination character. On CL_ERR_BUFFER_TOO_SMALL error, this value is the size of the buffer required to write the data text.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_MANU_DOES_NOT_EXIST	The requested manufactures's dll does not exist on your system.
CL_ERR_BUFFER_TOO_SMALL	User buffer not large enough to hold data.
CL_ERR_ERROR_NOT_FOUND	Could not find the error description for this error code.
BFCL_ERROR_NULLPTR	*errorText* and/or *errorTextSize* are null pointers.

Comments This function first looks up the error code in clserial.dll to determine whether it is a standard Camera Link error. If it is a non-standard error, this function passes the error code to clserbit.dll, which returns the manufacturer specific error code.

45.4 clGetNumPorts

Prototype CLINT32 clGetNumPorts(CLUINT32* *numPorts*)

Description This function returns the number of Camera Link serial ports installed in the computer that are supported by clallserial.dll.

Parameters ***numPorts***

The number of Camera Link serial ports installed in the computer.

Returns

CL_ERR_NO_ERR	Function was successful.
BFCL_ERROR_BRDOPEN	Error opening board to determine if it is a Camera Link board.

Comments

45.5 clGetNumBytesAvail

Prototype CLINT32 clGetNumBytesAvail(hSerRef *serialRef*, CLUINT32* *numBytes*)

Description This function outputs the number of bytes that are received, but not yet read out.

Parameters *serialRef*

The serial reference returned by clSerialInit.

numBytes

The number of bytes currently available to be read from the port.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
BFCL_ERROR_BYTES_AVAIL	Error getting the number of bytes available.

Comments

45.6 clGetPortInfo

Prototype CLINT32 clGetPortInfo(CLUINT32 *serialIndex*, CLINT8* *manufacturerName*, CLUINT32* *nameBytes*, CLINT8* *portID*, CLUINT32* *IDBytes*, CLUINT32* *version*)

Description This function provides information about the port specified by *serialIndex*.

Parameters

serialIndex

Zero based index of the serial port you are finding the name for. Use clGetNumSerialPorts to determine the valid range of this parameter. This range will be 0 to numSerialPorts - 1.

***manufacuterName**

Pointer to a user allocated buffer into which the function copies the manufacturer name. The returned name is NULL terminated.

***nameBytes**

As an input parameter, this value is the size of the manufacturerName buffer, including the NULL termination. As an output parameter, this parameter is the number of bytes written into the name buffer. If the provided name buffer is not large enough, this value is the number of required bytes.

***portID**

The identifier for the port.

***IDBytes**

As an input parameter, this value is the size of the *portID* buffer, including the NULL termination. As an output parameter, this value is the number of bytes written into the portID buffer. If the provided portID buffer is not large enough, this value is the number of required bytes.

***version**

The version of the Camera Link specifications with which the framegrabber complies.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_BUFFER_TOO_SMALL	User buffer not large enough to hold data.
CL_ERR_INVALID_INDEX	*serialIndex* was not a valid index.

Comments

45.7 clGetSupportedBaudRates

Prototype CLINT32 clGetSupportedBaudRates(hSerRef *serialRef*, CLUINT32* *baudRates*)

Description This function returns the valid baud rates that the framegrabber supports for serial communication.

Parameters *serialRef*

The serial reference returned by clSerialInit.

baudRates

Indicates which baud rates are supported by the framegrabber. This is represented as a bitfield with the following constants:

CL_BAUDRATE_9600 - 9600 baud rate. Value = 1.
CL_BAUDRATE_19200 - 19200 baud rate. Value = 2.
CL_BAUDRATE_38400 - 38400 baud rate. Value = 4
CL_BAUDRATE_57600 - 57600 baud rate. Value = 8.
CL_BAUDRATE_115200 - 115200 baud rate. Value = 16
CL_BAUDRATE_230400 - 230400 baud rate. Value = 32.
CL_BAUDRATE_460800 - 460800 baud rate. Value = 64.
CL_BAUDRATE_921600 - 921600 baud rate. Value = 128.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_FUNCTION_NOT_FOUND	Function does not exist in the manufacturer's library.
BFCL_ERROR_NULLPTR	The *baudRate* parameter has a NULL pointer.

Comments

45.8 clSerialClose

Prototype void clSerialClose(hSerRef *serialRef*)

Description Closes the serial device and cleans up the resources associated with serialRef.

Parameters *serialRef*

The value obtained from the clSerialInit function.

Returns None.

Comments

45.9 clSerialInit

Prototype CLINT32 clSerialInit(CLUNT32 *serialIndex*, hSerRef* *serialRefPtr*)

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

Parameters *serialIndex*

The number of the serial device in the system to initialize. This number is a zero-based index value. This n number of serial devices in the system, the serialIndex has a range 0 to (n-1).

serialRefPtr

Points to a value that contains, on a successful call, a pointer to the vendor-specific reference to the current session.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_PORT_IN_USE	Port is valid but cannot be opened because it is in use.
CL_ERR_INVALID_INDEX	Not a valid index.
BFCL_ERROR_SERNOTFOUND	The serial device specified by serialIndex was not found.
BFCL_ERROR_BRDNOTFOUND	There where no RoadRunner boards found.
BFCL_ERROR_BRDOPEN	Error opening board.
BFCL_ERROR_NOSIGNAL	The interrupt signal could not be created.
BFCL_ERROR_NOSTRUC	Memory for struction could not be allocated.
BFCL_ERROR_THRE	The transmitter holding register was not empty after initialization.
BFCL_ERROR_TEMT	The transmitter buffer was not empty after initialization.
BFCL_ERROR_FIFO_EN	Initialization failed to enable the FIFOs.
BFCL_ERROR_RCVRFIFO	Error with the receiver FIFO during initialization.

Comments This function initializes the serial device referred to by serialIndex. The serial device is the UART on the boards with Camera Link interfaces. If, for example, two R3-CL boards are installed in a system there are two serial devices available, serial device 0 on board 0 and serial device 1 on board 1. If there is, for example, a RoadRunner and a RoadRunnerCL installed in the system, there is only one serial device available.

45.10 clSerialRead - Deprecated as of CL 2.1

Prototype CLINT32 clSerialRead(hSerRef *serialRef*, CLINT8* *buffer*, CLUINT32* *numBytes*, CLUINT32 *serialTimeout*)

Description This function reads the serial device referenced by serialRef.

Parameters ***serialRef***

The value obtained from the clSerialInit function.

buffer

Points to a user-allocated buffer. Upon a successful call, buffer contains the data read from the serial device. If there is an error or timeout, the buffer will be returned empty.

bufferSize

The number of bytes requested by the caller.

serialTimeout

Indicates the time-out in milliseconds.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_TIMEOUT	The timeout has elapsed.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
BFCL_ERROR_NULLPTR	A NULL pointer was passed into the function.
BFCL_ERROR_RCVRFIFO	Error with the receiver FIFO during initialization.
BFCL_WARN_BUFFULL	The buffer was full before the read was complete.

Comments This function reads the serial device referenced by *serialRef*. This function fills the host buffer, *buffer*, until *bufferSize* bytes have been received from the camera. If *bufferSize* bytes have not been received by *serialTimeout* milliseconds, the function will return a time out error, the buffer will be returned empty and *bufferSize* will be set to zero.

45.11 clSerialReadEx

Prototype CLINT32 clSerialReadEx(hSerRef *serialRef*, CLINT8* *buffer*, CLUINT32* *numBytes*, CLUINT32 *serialTimeout*)

Description This function reads the serial device referenced by serialRef.

Parameters

serialRef

The value obtained from the clSerialInit function.

buffer

Points to a user-allocated buffer. Upon a successful call, buffer contains the data read from the serial device.

bufferSize

The size of the buffer in bytes. Upon a successful call contains the number of bytes read from the device.

serialTimeout

Indicates the time-out in milliseconds.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_TIMEOUT	The timeout has elapsed.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
BFCL_ERROR_NULLPTR	A NULL pointer was passed into the function.
BFCL_ERROR_RCVRFIFO	Error with the receiver FIFO during initialization.
BFCL_WARN_BUFFULL	The buffer was full before the read was complete.

Comments This function reads the serial device referenced by *serialRef*. This function fills the host buffer, *buffer*, until *bufferSize* bytes have been received from the camera or the timeout has expired. When the function returns, *numBytes* will contain the number of bytes read from the device.

Note: This function was added in to the Camera Link specifiaction in revision 2.1. The BitFlow SDK 6.4 and later support the Camera Link revision 2.1.

Note: This function differs in subtle but important wasy from the function clSerialRead. Please see the CL 2.1 specification for more details.

45.12 clSerialWrite

Prototype CLINT32 clSerialWrite(hSerRef *serialRef*, CLINT8* *buffer*, CLUINT32* *bufferSize*, CLUINT32 *serialTimeout*)

Description Writes the data in the buffer to the serial device referenced by serialRef.

Parameters

serialRef

The value obtained from the clSerialInit function.

buffer

Contains data to write to the serial port.

bufferSize

Contains the buffer size indicating the maximum number of bytes to be written. Upon a successful call, *bufferSize* contains the number of bytes written to the serial device.

serialTimeout

Indicates the time-out in milliseconds.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_TIMEOUT	Timed out waiting to write data.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
BFCL_ERROR_NULLPTR	A NULL pointer was passed into the function.

Comments This function will try and write the data in *buffer* to the serial device for the number of milliseconds specified by *serialTimeout*. If the data could not be written within that time, the CL_ERR_TIMEOUT error will be returned.

45.13 clSetBaudRate

Prototype CLINT32 clSetBaudRate(hSerRef *serialRef*, CLUINT32 *baudRate*)

Description This function sets the baud rate for the serial port on the framegrabber.

Parameters *serialRef*

The value obtained from the clSerialInit function.

baudRate

The baud rate to be set on the serial port. Can be one of the following:

CL_BAUDRATE_9600 - 9600 baud rate.
CL_BAUDRATE_19200 - 19200 baud rate.
CL_BAUDRATE_38400 - 38400 baud rate.
CL_BAUDRATE_57600 - 57600 baud rate.
CL_BAUDRATE_115200 - 115200 baud rate.
CL_BAUDRATE_230400 - 230400 baud rate.
CL_BAUDRATE_460800 - 460800 baud rate.
CL_BAUDRATE_921600 - 921600 baud rate.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The serial reference is not valid.
CL_ERR_BAUD_RATE_NOT_SUPPORTED	The requested baud rate is not supported.

Comments Use the clGetSupportBaudRates function to determine supported baud rates.

45.14 clBFSerialSettings

Prototype int clBFSerialSettings(void* *serBFRef*, unsigned int *baudRate*, unsigned int *dataBits*, unsigned int *parity*, unsigned int *stopBits*)

Description Changes the serial device settings.

Parameters ***serBFRef***

The BitFlow serial handle.

baudRate

Specifies the baud rate setting. The baud rate can be set to one of the following:

CL_BAUDRATE_9600 - 9600 baud rate.
CL_BAUDRATE_19200 - 19200 baud rate.
CL_BAUDRATE_38400 - 38400 baud rate.
CL_BAUDRATE_57600 - 57600 baud rate.
CL_BAUDRATE_115200 - 115200 baud rate.
CL_BAUDRATE_230400 - 230400 baud rate.
CL_BAUDRATE_460800 - 460800 baud rate.
CL_BAUDRATE_921600 - 921600 baud rate.

dataBits

Specifies the number of data bits in each transmitted or received in each serial character. The number of data bits can be set to one of the following:

DataBits_5 - Five bit serial characters
DataBits_6 - Six bit serial charaters
DataBits_7 - Seven bit serial charaters
DataBits_8 - Eight bit serial characters

parity

Specifies the parity to be used if any. The parity can be set to one of the following:

ParityEven - Even parity.
ParityOdd - Odd parity.
ParityNone - No parity.

stopBits

Specifies the number of stop bits transmitted and received in each serial character. The number of stop bits can be set to one of the following:

StopBits_1 - One stop bit.
StopBits_15 - One and a half stop bits.
StopBits_2 - Two stop bits.

Returns

CL_ERR_NO_ERR	Function was successful.
BFCL_ERROR_BAUDRATE	An invalid baud rate was specified.
BFCL_ERROR_DATABITS	An invalid number of data bits was specified.
BFCL_ERROR_PARITY	An invalid parity parameter was specified.
BFCL_ERROR_15STOP5-DATA	To use 1.5 number of stop bits, 5 bit data must be used.
BFCL_ERROR_2STOP5-DATA	Two stop bits can not be used with five data bits.
BFCL_ERROR_STOPBITS	An invalid number of stop bits was specified.

Comments

This is a non-standard function which allows for changing of serial communications parameters that are not supported by the normal CL serial API.

45.15 clBFSerialRead

Prototype int clBFSerialRead(void* *serBFRef*, char* *buffer*, unsigned int* *bufferSize*)

Description This function reads the serial device referenced by serBFRef.

Parameters ***serBFRef***

The BitFlow serial handle.

buffer

Points to a user-allocated buffer. Upon a successful call, buffer contains the data read from the serial device

bufferSize

As an input parameter, this contains the maximum number of bytes that the buffer can accommodate. Upon a successful call, bufferSize is the number of bytes that were read successfully from the serial device.

Returns

CL_ERR_NO_ERR	Function was successful.
BFCL_WARN_SIG_CANCEL	The function was force to return by the clBFSerialCancelRead function.
BFCL_ERROR_RCVRFIFO	An error occurred waiting for data on the port.

Comments This function is similar to clSerialRead function except that without a timeout value this function will wait efficiently for data to be read from the serial port. Once there is data at the port this function will read the data out and return it through the *buffer* parameter. The *bufferSize* paramerter will be returned with the number of bytes read into *buffer*.

In order to force this function to return will waiting for data on the serial port, use the clBFSerialCancelRead function.

If the user knows how many bytes are to be received they can use the clSerialRead function. This function is useful when the user is unsure how many bytes are being received. In the case of not knowing how many bytes are being received, the user can call this function in a loop, and every time data is received this function will return the data until there is nothing left to return. This is how this function is used in the example application BFCom, that comes with the BitFlow SDK. In BFCom, there is a read thread that is started when the application starts up. The read thread calls this function in a loop until the application is closed. While the application is opened, the read thread takes any returned data from this function and displays it to screen.

45.16 clBFSerialCancelRead

Prototype int clBFSerialCancelRead(void* *serBFRef*)

Description This function cancels the serial interrupt signal that clBFSerialRead waits for, forcing the clBFSerialRead function to return.

Parameters ***serBFRef***

The BitFlow serial handle.

Returns

CL_ERR_NO_ERR	Function was successful.

Comments This function is for use in conjunction with the function clBFSerialRead, which does not time out. You will need to call clBFSerialCancelRead from another thread in order to get clBFSerialRead to return. When it does return, it will indicate that the wait was cancelled (rather than new bytes coming from the camera).

45.17 clBFGetBaudRate

Prototype int clBFGetBaudRate(void* *serBFRef*, unsigned int* *baudRate*)

Description This function returns the current baud rate of the serial port.

Parameters *serBFRef*

The BitFlow serial handle.

buadRate

The returned baud rate for the serial port. The returned rate can be one of the following:

CL_BAUDRATE_9600 - 9600 baud rate.
CL_BAUDRATE_19200 - 19200 baud rate.
CL_BAUDRATE_38400 - 38400 baud rate.
CL_BAUDRATE_57600 - 57600 baud rate.
CL_BAUDRATE_115200 - 115200 baud rate.
CL_BAUDRATE_230400 - 230400 baud rate.
CL_BAUDRATE_460800 - 460800 baud rate.
CL_BAUDRATE_921600 - 921600 baud rate.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_BAUD_RATE_NOT_SUPPORTED	Could not determine the current baud rate.

Comments The standard CL serial API does not have a function to get the current baud rate, this BitFlow CL API function fills this need.

45.18 clBFGetSerialRef

Prototype int clBFGetSerialRef(unsigned int *portNum*, void** serBFRefPtr)

Description This function is used to get the BitFlow serial handle for use with all clBF functions for a given port number.

Parameters *portNum*

The number of the port for which the serial reference is requested.

serBFRefPtr

A pointer to the returned BitFlow serial handle. This reference should be used for any calls to clBFSerialSetting, clBFSerialRead, clBFSerialCancelRead and clBFGetBaudRate functions.

Returns

CL_ERR_NO_ERR	Function was successful.
BFCL_ERROR_SERNOTFOUND	Could not determine the serial reference.

Comments This function returns a BitFlow serial handle for a given port number. The BitFlow serial handle is for use with the BitFlow specific serial functions (listed in this chapter). The port number provided to this function should be the same as the port number provided clSerialInit.

Note: The BitFlow serial handle returned from this function is different from the CL serial handle returned from clSerialInit. The former is for use with the BitFlow serial functions and the latter is for use with the CL serial functions. This is due to the obsufacation that takes place in the middleware DLL, "clallserial.dll".

45.19 clBFGetSerialRefFromBoardHandle

Prototype int clBFGetSerialRefFromBoardHandle(Bd *hBoard*, void** serBFRefPtr)

Description This function is used to get the BitFlow serial handle for use with all clBF functions for a given board handle.

Parameters *hBoard*

A handle to a BitFlow board

serBFRefPtr

A pointer to the returned BitFlow serial handle. This reference should be used for any calls to clBFSerialSetting, clBFSerialRead, clBFSerialCancelRead and clBFGetBaudRate functions.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The board handle is invalid.
BFCL_ERROR_SERNOTFOUND	Could not determine the serial reference.
CL_XXXX	Can also return all the values that clSerialInit can return.

Comments This function returns a BitFlow serial handle for a board handle. The BitFlow serial handle is for use with the BitFlow specific serial functions (listed in this chapter). The board handle provided to this function must be the handle to a board already opened by one of the board open functions.

Note: The BitFlow serial handle returned from this function is different from the CL serial handle returned from clSerialInit. The former is for use with the BitFlow serial functions and the latter is for use with the CL serial functions. This is due to the obsufacation that takes place in the middleware DLL, "clallserial.dll".

45.20 clBFSerialInitFromBoardHandle

Prototype int clBFSerialInitFromBoardHandle(Bd *hBoard*, void** *serialRefPtr*)

Description This function is used to get the CL serial handle for use with all CL functions for a given board handle.

Parameters *hBoard*

A handle to a BitFlow board

serialRefPtr

A pointer to the returned CL serial reference pointer. This reference should be used for any calls to CL serial API.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The board handle is invalid.
CL_XXXX	Can also return all the values that clSerialInit can return.

Comments This function returns a CL serial handle for a board handle. The CL serial handle is for use with the CL serial functions (listed in this chapter). This is the same type of handle as is returned from clSerialInit and can be used in the same way. The board handle provided to this function must be the handle to a board already opened by one of the board open functions.

This function also opens and initialized the serial port, just like clSerialInit. The difference is that this function opens the serial port based on a board handle, while clSerialInit open the serial port based on a port number.

45.21 clBFSerNumtFromBoardHandle

Prototype int clBFSerNumtFromBoardHandle(Bd *hBoard*, PBFU32 *pSerNum*)

Description This function is used to get the CL serial port number for the given board.

Parameters *hBoard*

A handle to a BitFlow board

pSerNum

A pointer to the returned port number.

Returns

CL_ERR_NO_ERR	Function was successful.
CL_ERR_INVALID_REFERENCE	The board handle is invalid.

Comments This function returns can be use to get the CL port number for the given board handle. Not that board number will equal port number if all the frame grabbers in the system are Camera Link. However, this relationship does not hold if there are non-CL board in the system.

Display Functions

Chapter 46

46.1 Introduction

The BitFlow SDK includes a simple set of functions for displaying live video in a window. The functions exists in a stand alone DLL and have their own API. They are independent of any of the other frame grabber access and control functions and can be used to display images from any source.

The display surface window is separate and independent from the main calling applications window. The display surface window can be moved anywhere on the desktop and can be independently resized, panned and scrolled.

To use these functions, first create a display surface. An application can create as many display surfaces as it needs. Next, get the bitmap that backs the display surface and pass it to any of the acquisition setup functions (e.g. CiAqSetup). Finally, program the LUTs on the board to match the VGA palette, if necessary, and your application is ready to display images.

The display surface window can be closed by the user, in which case the display surface still exists, it is just not visible. You can detect this condition from your application by calling one of the IsOpen functions. The display surface itself is deleted by calling one of the display surface close functions. You must make sure that the board is not DMAing to the surface's bitmap buffer when you delete the surface. Always make sure the board is frozen before calling the close function. If you call the close function when the window is still visible, the window will be automatically closed before the display surface is deleted.

46.2 DispSurfCreate

Prototype

BOOL DispSurfCreate(PBFS32 *DspsurfHandle*, BFU32 *Dx*, BFU32 *Dy*, BFS32 *PixDepth*, HWND *HwndOwner*)

Description

Creates a display surface (video window) on the desktop, above the given parent window.

Parameters

DspsurfHandle

Pointer to a BFS32. Should be -1 when the function is first called. When the function returns, the value will be a handle used to refer to the surface in subsequent function calls.

Dx

Horizontal size of the destination bitmap in pixels. This is not the size of the window but the size of the bitmap, and should be the same as the X size of the camera.

Dy

Vertical size of the destination bitmap in lines. This is not the size of the window but the size of the bitmap, and should be the same as the Y size of the camera.

PixDepth

Number of bits per pixel. Currently this value must = 8, 24, or 32.

HwndOwner

Window's handle to the parent application's window. This forces the resulting display surface to be a child of the application's window. If this parameter is NULL, the display surface will be independent.

Returns

TRUE	If successful
FALSE	Error creating bitmap.

Comments

This function creates a display surface on the Windows desktop. A display surface is just a window that can display images. The window can be resized (up to the size of the camera's image), panned, scrolled, and moved like any other window. Creating a display surface also creates a bitmap.

This bitmap can be the destination of DMA operations from the board. Call the function DispSurfGetBitmap to get a pointer to the resulting bitmap's memory buffer. This pointer can be passed to any of the acquisition setup functions.

The window on the display is not automatically updated. The application must call DispSurfBlit to get new data in the bitmap on to the actual display.

46.3 DispSurfGetBitmap

Prototype BOOL DispSurfGetBitmap(BFS32 *DspsurfHandle*, PBFVOID **pBitmap*)

Description Gets a pointer to the memory buffer of the bitmap behind a display surface.

Parameters *DspsurfHandle*

Handle to display surface.

pBitmap

Pointer to a PBFVOID. When this function returns, the pointer will point to the bitmap's memory buffer.

Returns

TRUE	If successful
FALSE	Cannot get the bitmap or display surface, handle is invalid.

Comments This function gets a pointer to the actual memory buffer of a bitmap that is used by a display surface. Like any other memory, this pointer can be read or written to directly. It can also be the destination of DMA operations. This pointer can be passed as the destination parameter when calling the CiAqSetup() function.

The pointer points to valid memory as long as the display surface exists.

46.4 DispSurfTop

Prototype BOOL DispSurfTop(BFS32 *DspsurfHandle*)

Description Puts the display surface window at the top of the Z order.

Parameters *DspsurfHandle*

Handle to display surface.

Returns

TRUE	If successful
FALSE	Error moving window to top or display surface does not exist.

Comments This function moves the display surface window to the top of the Z order. This means the window will be on top of all other windows.

46.5 DispSurfBlit

Prototype BOOL DispSurfBlit(BFS32 *DspsurfHandle*)

Description Updates the display surface window with the data currently in the display surface's bitmap.

Parameters *DspsurfHandle*

Handle to bitmap.

Returns

TRUE	If successful
FALSE	Error blitting or display surface does not exist.

Comments This function updates the display surface window (the image you see on the VGA) with the data currently in the bitmap. Changing the data in a bitmap does not automatically change the data display window. This function must be called every time the bitmap is updated. A simple way to keep a display surface window updated is to create a thread that waits for a end of DMA signal. Whenever the signal occurs, call this function to update the display.

This function can also be called if you have written your own data into the bitmap and want it to be displayed.

46.6 DispSurfChangeSize

Prototype BOOL DispSurfChangeSize(BFS32 *DspsurfHandle*, BFU32 *Dx*, BFU32 *Dy*, BFU32 *PixDepth*)

Description Changes the size of a display surface bitmap. Does not change the window size.

Parameters

DspsurfHandle

Handle to bitmap.

Dx

New horizontal size of the destination bitmap in pixels. This is not the size of the window but the size of the frame, and should be the same as the X size of the camera.

Dy

New vertical size of the destination bitmap in lines. This is not the size of the window but the size of the frame, and should be the same as the Y size of the camera.

PixDepth

Number of bits per pixel, currently this value must = 8, 24, 32.

Returns

TRUE	If successful
FALSE	Error changing size or display surface does not exist.

Comments This function allows you to change the size of a bitmap without destroying and recreating the display surface. Essentially, this function destroys the current bitmap and creates a new one at the given size. The only purpose of this function is to handle on-the-fly changes of acquisition size (as might happen when changing the current camera). This does not change the size of the window, and can only be done through the GUI.

46.7 DispSurfGetLut

Prototype BOOL DispSurfGetLut(BFS32 *DspsurfHandle*, PBFU8 *pLut*)

Description Returns a LUT that will match the palette that has been created in the display surface.

Parameters

DspsurfHandle

Handle to bitmap.

pLut

Pointer to an already allocated byte array of 256 bytes.

Returns

TRUE	If successful
FALSE	Error getting LUT or display surface does not exist.

Comments In order to optimize display speed, the palette of the bitmap in a display surface and the VGA's palette must be matched. When the display surface is created, the palettes are automatically matched for the current VGA color mode. However, the image must also be mapped to match this palette (the palette matching does not guarantee that all colors will be available). The board's LUTs can perform this matching. Use this function to get the correct values to program the LUT with.

This function returns an array of bytes which can be passed to the LUT programming function. Use 8-bit mode and write this array to all four 8-bit lanes.

Note: This function is only required if your VGA is in 8-bit (256) color mode.

Note: This function is only useful if the board being used has LUTs.

46.8 DispSurfClose

Prototype BOOL DispSurfClose(BFS32 *DspsurfHandle)*

Description Closes a display surface.

Parameters *DspsurfHandle*

Handle to display surface.

Returns

TRUE	If successful
FALSE	Error getting LUT or display surface does not exist.

Comments This function closes and destroys a display surface. The bitmap associated with the surface will be destroyed. The window on the desktop will be closed.

The display surface can also be closed from the GUI by the user.

46.9 DispSurfIsOpen

Prototype BOOL DispSurfIsOpen(BFS32 *DspsurfHandle)*

Description Check to see if a display surface is open.

Parameters *DspsurfHandle*

Handle to display surface.

Returns

TRUE	If the surface exists and its window is open on the desktop.
FALSE	If the surface does not exist or the window has been closed.

Comments These functions provide a simple way to check if a display surface has already been opened, and thus the handle can be used for subsequent commands. It also is a good way to check if the user has closed the window.

46.10 DispSurfOffset

Prototype BOOL DDrawSurfOffset(BFS32 *DspsurfHandle*, BFS32 *DX*, BFS32 *DY*)

Description Moves the display surface window.

Parameters *DspsurfHandle*

Handle to display surface.

DX

Amount to offset display surface horizontally (in screen pixels).

DY

Amount to offset display surface vertically (in screen pixels).

Returns

TRUE	If successful
FALSE	On error.

Comments These functions move the display surface window on the VGA. The parameters *DX* and *DY* are the amount to move the surface relative to its current position. The units are in terms of screen pixels.

46.11 DispSurfSetWindow

Prototype

BOOL DispSurfSetWindow(BFS32 *DspsurfHandle* BFU32 *Left*,BFU32 *Top*, BFU32 *Width*, BFU32 *Height*)

Description

Sets the location and the size of the display surface window.

Parameters

DspsurfHandle

Handle to display surface.

Left

Sets the X coordinate of the upper left corner of the window (in screen pixels).

Top

Sets the Y coordinate of the upper left corner of the window (in screen pixels).

Width

Sets the width of the window (in screen pixels).

Top

Sets the height of the window (in screen pixels).

Returns

TRUE	If successful
FALSE	On error.

Comments

These functions set the location and the size of the display surface window on the VGA. The parameters *Left* and *Top* set the location of the window. The upper left corner of the workspace is (0,0). The parameters *Width* and *Height* set the size of the window. If the size you set the size of the window bigger than the display surface's bitmap, the new sizes will be reduced to the size of the bitmap. The units are in terms of screen pixels.

46.12 DispSurfGetWindow

Prototype

BOOL DispSurfGetWindow(BFS32 *DspsurfHandle* PBFU32 *pLeft*,PBFU32 *pTop*, PBFU32 *pWidth*, PBFU32 *pHeight*)

Description

Gets the location and the size of the display surface window.

Parameters

DspsurfHandle

Handle to display surface.

pLeft

Contains the current value of the X coordinate of the upper left corner of the window (in screen pixels).

pTop

Contains the current value of the Y coordinate of the upper left corner of the window (in screen pixels).

pWidth

Contains the current value of the width of the window (in screen pixels).

pTop

Contains the current value of the height of the window (in screen pixels).

Returns

TRUE	If successful
FALSE	On error.

Comments

These functions gets the location and the size of the display surface window. The parameters *pLeft* and *pTop* contain the location of the window. The upper left corner of the workspace is (0,0). The parameters *pWidth* and *pHeight* return the size of the window. The units are in terms of screen pixels.

46.13 DispSurfTitle

Prototype BOOL DispSurfTitle(BFS32 *DspsurfHandle*, PBFCHAR *Title*)

Description Changes the title on the display surface window.

Parameters *DspsurfHandle*

Handle to display surface.

Title

Character string to make the title bar of the window to.

Returns

TRUE	If successful
FALSE	On error.

Comments These functions give the user the ability to customize the title bar of display surface windows.

46.14 DispSurfDisableClose

Prototype BOOL DispSurfDisableClose(BFS32 *DspsurfHandle*, BFBOOL *Enabled*)

Description Enables or disables the ability of the user to close display surface window.

Parameters *DspsurfHandle*

Handle to display surface.

Enabled

Used to disable or enable the ability of the user to close the display surface:

TRUE - The user will not be able to close the display surface.
FALSE - The user will be able to close the display surface.

Returns

TRUE	If successful
FALSE	On error.

Comments This function provides the ability to prevent the user of an application from closing the display surface. This is useful where an application opens a display surface and would like that display surface to remain open until the application is closed.

To use this function properly, create the display surface with the DispSurfCreate function. After creation of the display surface, call this function with the Enabled parameter set to TRUE to prevent the display surface from being closed.

By default, if the display surface is created and this function is not called, the display surface will be able to be closed by the user of the application.

46.15 DispSurfFormatBlit

Prototype

BOOL DispSurfFormatBlit(BFS32 *DspSurfHandle*, PBFU32 *pBuffer*, BFU32 *PixDepth*, BFU32 *Options*)

Description

Formats and updates the display surface window with the image data pointed to by *pBuffer*.

Parameters

DspSurfHandle

Handle to bitmap.

pBuffer

A pointer to the image data stored in memory.

PixDepth

The pixel depth of the image data in bits.

Options

Options for formatting the image data is as follows:

BFDISP_FORMAT_NORMAL - Format the data to display the most significant 8-bits.
BFDISP_FORMAT_PACKED - Use when capturing packed data that is greater than 8 bits per pixels. Data is byte swapped.
BFDISP_FORMAT_PACKED_NOSWAP - Used when capturing packed data that is greater than 8 bits per pixels. Data is not byte swapped.
BFDISP_FORMAT_RGB_SWAP - Use when capturing 24-bit RGB data that needs the R and G swapped for correct display in Windows.

Returns

TRUE	If successful
FALSE	Error blitting or display surface does not exist.

Comments

This function is identical to the DispSurfBlit function except that this function will format the image data to be displayed correctly on the display surface. After the image data is formatted for display, the display surface window is updated.

The formatting of the image data is based on the pixel depth of the image and the format options. Under normal formatting, the upper 8 most significant bits are displayed.

46.16 DispSurfSetZoom

Prototype BOOL DispSurfSetZoom(BFS32 *DspsurfHandle*, BFU32 *Zoom*)

Description Sets the zoom value of the display surface window.

Parameters *DspsurfHandle*

Handle to display surface.

Zoom

The amount of zoom to set the window to. Permitted values are:

10 - 10%
25 - 25%
50 - 50%
100 - 100% (no zoom)
200 - 200%
400 - 400%
800 - 800%

Returns

TRUE	If successful
FALSE	On error.

Comments This functions sets the zoom value for the display surface. This function has the same affect s changing the zoom using the display surface menu. Only the values like above are supported.

46.17 DispSurfGetZoom

Prototype BOOL DispSurfGetZoom(BFS32 *DspsurfHandle*, PBFU32 *pZoom*)

Description Gets the current zoom value of the display surface window.

Parameters *DspsurfHandle*

Handle to display surface.

pZoom

Returns the current amount of zoom. It will be one of the following:

10 - 10%
25 - 25%
50 - 50%
100 - 100% (no zoom)
200 - 200%
400 - 400%
800 - 800%

Returns

TRUE	If successful
FALSE	On error.

Comments This functions gets the zoom value for the display surface.

BitFlow Common Functions Introduction

Chapter 47

47.1 Overview

The functions in the following chapters belong to the BF API. This API is the foundation for all the other APIs in this manual. Essentially these functions provide low level access to the hardware. The BF functions work on all board families, provided the memory being accesses is actually mounted on the target board. In general, there is rarely a need to call these functions, as the higher level APIs provide easy access to all of the board functionality. However, occasionally some customization of a function or the board's mode is needed, and the BF functions can be used to get right down to the hardware.

CoaXPress specific functions

Chapter 48

48.1 Introduction

The functions described in this chapter can be used with any BitFlow board with a CoaXPress (CXP) interface. The CXP boards generally work the same as the Camera Link boards with regards to acquisition, triggering, DMA etc. However, the CXP boards have some special features that require special function for managing the CXP portions of the boards. These CXP specific functions are listed in this chapter.

48.1.1 CoaXPress Camera Control

The CoaXPress specification (unlike Camera Link specification) explicitly dictates the protocol to be used to control CXP devices (e.g. a camera) from a CXP host (e.g. a frame grabber). The protocol is based on GenICam, which is a well developed specification used by many camera/frame grabber interconnect standards. GenICam consists of a few different components. Generally they can be grouped in to a low level specification for how a host controls a device, and a set of high level interfaces which provide APIs for applications to control devices and acquire data from devices. It is beyond the scope of this manual to describe GenICam in detail. For more information on GenICam please see www.genicam.org.

At the lowest level, GenICam specifies that all devices (cameras) should be controlled by "register based" access. In this case, "register based" means that all camera functions are controlled by read/writing from/to various addresses in the device. Some addresses are the same for all devices, and are dictated by the CoaXPress specification, other addresses can be used however the manufacturer sees fit. GenICam provides a standardized mechanism for the camera maker to provide a list of the addresses and the features controlled by each address.

BitFlow provides full GenICam support. Included is an GenICam application which provides a GUI for full control of all of your camera's features. There is also a GenICam API which provides access into the GenICam system for your own application. However, we also provide a very simple low level register read/write API which is easy to use, and some time the best option when only minor changes are needed in the camera configuration. These functions are documented in the following section.

48.1.2 Example Usage

To see examples how these functions are used, refer to the source code installed by the SDK: "\BitFlow SDK X.XX\Examples\CXPRegTool".

48.2 BFCXPReadReg

Prototype BFCAPI BFCXPReadReg(Bd *Board*, BFU32 *Link*, BFU32 *Address*, PBFU32 *pValue*)

Description Read a register from a CoaXPress device.

Parameters

Board

Handle to board.

Link

Link number (0 based). In general any VFG can read/write from/to any CXP link. However, it usually only makes sense to communicate with the camera the VFG is acquiring from. If you are not sure, set this value to 0xff.

Address

Address to read from.

pValue

Pointer to a 32-bit integer which will contain the requested value when this function returns.

Returns

BF_OK	Function was successful.
BF_BRD_NOT_CXP	*Board* is not a handle to a CXP board.
BF_BRD_CXP_COM_TIMEOUT	Timeout waiting for acknowledgement.
BF_BRD_CXP_COM_BAD_ACK	Receive FIFO empty
BF_BRD_CXP_COM_BAD_ACK	Wrong number of bytes received

Comments This function is used to read a single 32-bit register from the CXP device attached to link number *link* on the VFG whose handle is *Board*. If the link number is not known, just set *link* equal to 0xff. The function will return with the 32-bit value in the parameter *pValue* unless there is a problem, then it will return an error.

With the exception of the CXP boot strap registers, each camera's address space is different. Please contact the camera manufacturer for an address map of your camera.

48.3 BFCXPWriteReg

Prototype BFCAPI BFCXPWriteReg(Bd *Board*, BFU32 *Link*, BFU32 *Address*, BFU32 *Value*)

Description Write a register to a CoaXPress device.

Parameters

Board

Handle to board.

Link

Link number (0 based). In general any VFG can read/write from/to any CXP link. However, it usually only makes sense to communicate with the camera the VFG is acquiring from. If you are not sure, set this value to 0xff.

Address

Address to write to.

Value

A 32-bit integer value which will be written to register at address *Address*.

Returns

BF_OK	Function was successful.
BF_BRD_NOT_CXP	*Board* is not a handle to a CXP board.
BF_BRD_CXP_COM_TIMEOUT	Timeout waiting for acknowledgement.
BF_BRD_CXP_COM_BAD_ACK	Error code in camera acknowledgement.

Comments This function is used to write a single 32-bit register to the CXP device attached to link number *link* on the VFG whose handle is *Board*. If the link number is not known, just set *link* equal to 0xff. The function will write the 32-bit value in the parameter *Value* unless there is a problem, then it will return an error.

With the exception of the CXP boot strap registers, each camera's address space is different. Please contact the camera manufacturer for an address map of your camera.

48.4 BFCXPReadData

Prototype BFCAPI BFCXPReadData(Bd *Board*, BFU32 *Link*, BFU32 *Address*, BFU32 *ReadSizeRequested*, PBFU32 *pReadSizeActual*, PBFVOID *pBuffer*, BFSIZET *BufferSize*)

Description Read a block of memory from a CoaXPress device.

Parameters

Board

Handle to board.

Link

Link number (0 based). In general any VFG can read/write from/to any CXP link. However, it usually only makes sense to communicate with the camera the VFG is acquiring from. If you are not sure, set this value to 0xff.

Address

Start address of block of memory to read from.

ReadSizeRequested

Number of bytes to read.

pReadSizeActual

Number of bytes actually read.

pBuffer

Pointer to a buffer which will hold the block of memory when this function returns.

BufferSize

Allocated size of *pBuffer* (in bytes).

Returns

BF_OK	Function was successful.
BF_NULL_POINTER	*pBuffer* is NULL.
BF_BRD_NOT_CXP	*Board* is not a handle to a CXP board.
BF_BRD_CXP_LINK_ERROR	Can not determine the device's block size.
BF_BRD_CXP_COM_TIMEOUT	Timeout waiting for acknowledgement.
BF_BRD_CXP_COM_BAD_ACK	Error code in camera acknowledgement.
BF_CXP_COM_READ_ERROR	Wrong number of bytes received.

Comments

This function is used to read a contiguous block of memory from the device into a host buffer. The block is defined by a starting address and block size. The size of the block must be smaller or equal to the size of the host buffer used to receive the data.

With the exception of the CXP boot strap registers, each camera's address space is different. Please contact the camera manufacturer for an address map of your camera.

48.5 BFCXPWriteData

Prototype BFCAPI BFCXPWriteData(Bd *Board*, BFU32 *Link*, BFU32 *Address*, PBFVOID *pBuffer*, BFU32 *NumBytesToWrite*)

Description Write a block of memory to a CoaXPress device.

Parameters

Board

Handle to board.

Link

Link number (0 based). In general any VFG can read/write from/to any CXP link. However, it usually only makes sense to communicate with the camera the VFG is acquiring from. If you are not sure, set this value to 0xff.

Address

Start address of block of memory to write to.

pBuffer

Pointer to a buffer containing the data to write to the device.

NumBytesToWrite

Number of bytes to write to the device.

Returns

BF_OK	Function was successful.
BF_NULL_POINTER	*pBuffer* is NULL.
BF_BRD_NOT_CXP	*Board* is not a handle to a CXP board.
BF_BRD_CXP_LINK_ERROR	Can not determine the device's block size.
BF_BRD_CXP_COM_TIMEOUT	Timeout waiting for acknowledgement.
BF_BRD_CXP_COM_BAD_ACK	Error code in camera acknowledgement.
BF_CXP_COM_WRITE_ERROR	The write packet is not the correct size or the response packet is not the correct size..

Comments This function is used to write to a contiguous block of memory in the device from a host buffer. The block is defined by a starting address and block size. The size to write must be smaller or equal to the size of the host buffer.

With the exception of the CXP boot strap registers, each camera's address space is different. Please contact the camera manufacturer for an address map of your camera.

48.6 BFCXPConfigureLinkSpeed

Prototype

BFCAPI BFCXPConfigureLinkSpeed(Bd *Board*, BFU32 *Link*, BFU32 *NumCamLinks*, BFU32 *LinkSpeed*, BFBOOL *ProgCam*)

Description

Write a block of memory to a CoaXPress device.

Parameters

Board

Handle to board.

Link

Link number (0 based). In general any VFG can read/write from/to any CXP link. However, it usually only makes sense to communicate with the camera the VFG is acquiring from. If you are not sure, set this value to 0xff.

NumCamLinks

The number of links (on the current camera) that need to be programmed.

LinkSpeed

The new link speed to program the links to.

CXPSpeed125 - 1.25 Gb/S
CXPSpeed250 - 2.50 Gb/S
CXPSpeed3125 - 3.125 Gb/S
CXPSpeed500 - 5.00 Gb/S
CXPSpeed625 - 6.25 Gb/S

ProgCam

Must be set to TRUE.

Returns

BF_OK	Function was successful.
BF_BRD_NOT_CXP	*Board* is not a handle to a CXP board.
BF_BRD_CXP_LINK_ERROR	Can not determine the device's block size.

Comments

This function programs the camera to the given speed. The board will automatically adjust to the camera's speed. The *NumCamLinks* parameter should be set to the number of links that are currently connecting the camera to the frame grabber.

48.7 BFCXPFindMasterLink

Prototype BFCAPI BFCXPFindMasterLink(Bd *Board*, PBFU32 *pLink*)

Description Find which frame grabber link is connected to the master link on the CoaXPress camera.

Parameters

Board

Handle to board.

Link

Returns the link number of frame grabber link that is connected to the camera's master link.

Returns

BF_OK	Function was successful.
BF_BRD_NOT_CXP	*Board* is not a handle to a CXP board.
BF_BRD_CXP_LINK_ERROR	Can not determine the device's block size.

Comments This function search all of the board's connected CXP links looking for the link whose DeviceConnectionID register is 0. This link is the camera's master link which is used for sending commands to the camera.

48.8 BFCXPIsPowerUp

Prototype BFCAPI BFCXPIsPowerUp(Bd *Board*, BFU32 *Link*)

Description Determine is a CXP link is powered up or not.

Parameters *Board*

Handle to board.

Link

Link number (0 based). In general any VFG can read/write from/to any CXP link. However, it usually only makes sense to communicate with the camera the VFG is acquiring from. If you are not sure, set this value of 0xff.

Returns

TRUE	The link is powered up
FALSE	The link is not powered

Comments This function can be used to determine if a link is powered up or not. The hardware automatically powers up links if the cameras indicates that it needs power. This function can be used to determine if the camera is powered up or not, also which links of a multiple link camera are using power and which are not.

BitFlow Error Handling

Chapter 49

49.1 Introduction

All of the BitFlow SDK functions will return error codes if the function does not execute correctly. High-level functions call mid-level functions which in turn call low-level functions, which in turn call kernel functions. Because this chain can be so long and some of the errors may happen in a low-level call, the return code of the top level function may not be very useful in debugging the root cause of the problem.

For these reasons, the BitFlow SDK uses an error stack. As errors occur, they are put on the stack. The stack can be walked and the errors can be examined, one by one, to determine the root cause of the malfunction.

Errors can be sent to a number of destinations; each destination is independent of all other destinations. The possible destinations are the event viewer, a pop-up dialog, a debugger (if there is one running), and a file. Also, an error can cause the application to abort at the user level. Each destination can be turned on or off independently. Each destination can be programmed to selectively filter out one or more particular errors.

49.2 BFErrorXXXXXX

Prototype

BFRC BFErrorEnableEvent(Bd *Board*, BFU32 *Filter*) - enables event viewer errors

BFRC BFErrorDisableEvent(Bd *Board*, BFU32 *Filter*) - disable event viewer errors

BFRC BFErrorEnableDebugger(Bd *Board*, BFU32 *Filter*) - enables debugger errors

BFRC BFErrorDisableDebugger(Bd *Board*, BFU32 *Filter*) - disables debugger errors

BFRC BFErrorEnableDialog(Bd *Board*, BFU32 *Filter*) - enables dialog errors

BFRC BFErrorDisableDialog(Bd *Board*, BFU32 *Filter*) - disables dialog errors

BFRC BFErrorEnableFile(Bd *Board*, BFU32 *Filter*) - enables log file errors

BFRC BFErrorDisableFile(Bd *Board*, BFU32 *Filter*) - disables log file errors

BFRC BFErrorEnableMonitor(Bd *Board*, BFU32 *Filter*) - enables debug monitor errors

BFRC BFErrorDisableMonitor(Bd *Board*, BFU32 *Filter*) - disables debug monitor errors

BFRC BFErrorEnableBreakUser(Bd *Board*, BFU32 *Filter*) - enables user level break after error

BFRC BFErrorDisableBreakUser(Bd *Board*, BFU32 *Filter*) - disables user level break after error

BFRC BFErrorEnableAll(Bd *Board*, BFU32 *Filter*) - enables errors for all error devices

BFRC BFErrorDisableAll(Bd *Board*, BFU32 *Filter*) - disable errors for all error devices

Description

These functions controls which error messages are active for each error destination.

Parameters

Board

Board to set error destination(s) on.

Filter

Error(s) to enable or disable.

Returns

RV_OK	If successful.
Non-zero	On error.

Comments

Each of the functions are independent. Each error destination can have its own list of

enable and disable errors independent of all other destinations. If an error destination gets conflicting instructions (e.g., calling BFErrorEnable(board,ErrorBug) followed by BFErrorDisable(board,ErrorAll)), the last error function that is called takes precedence.

Filter can be one of the following options:

A single error number (see header files "BFTabError.h", “R2TabError.h” and "R64-TabError.h").

One or more error types ORed together. The error types are:

ErrorBug - outright bug in the code that must be fixed.
ErrorFatal - deep, deep trouble. Program execution cannot continue.
ErrorWarn - something is wrong but recovery is possible.
ErrorInfo - informational messages.

The error type that represents none of the errors.

ErrorNone

The error type that represents all errors.

ErrorAll

49.3 BFErrorShow

Prototype BFRC BFErrorShow(Bd *BoardId*)

Description Displays errors on the error stack

Parameters *Board*

Board ID.

Returns

BF_OK	If no errors on stack.
Error number	Top error on error stack.

Comments This function pops errors off the error stack. For each error on the stack, a dialog will pop up displaying the error information. The user can click the Next button to walk through all of the errors on the stack. The user also has the option of aborting the application, or ignoring the rest of the error on the error stack. Regardless of what button the user clicks (other than abort), the error stack is cleared before this function returns.

Note: If a particular error has been disabled through a call to BFErrorDisableDialog, then it will not show up when calling this function.

49.4 BFErrorCheck

Prototype BFRC BFErrorCheck(Bd *BoardId*)

Description Returns the top error on the error stack.

Parameters *Board*

Board ID.

Returns

BF_OK	If no errors on stack.
Error number	Top error on error stack.

Comments This function returns the top error on the error stack. This function does not pop errors off the error stack. Use BFErrorClearLast to pop errors off the stack.

49.5 BFErrorClearAll

Prototype BFRC BFErrorClearAll(Bd *BoardId*)

Description Clears the error stack.

Parameters *Board*

Board ID.

Returns

BF_OK	Function succeeded.
Non-zero	Function failed.

Comments This function clears all error from the error stack.

49.6 BFErrorGetLast

Prototype BFRC BFErrorGetLast (Bd *BoardId,* PBFU32 *pLastError*)

Description Returns the top error on the error stack.

Parameters *Board*

Board ID.

pLastError

Points to top error on error stack

Returns

BF_OK	If no errors on stack.
Non-zero	Error getting top error.

Comments This function returns the top error on the error stack. This function does not pop errors off the error stack. Use BFErrorClearLast() to pop errors off the stack.

49.7 BFErrorClearLast

Prototype BFRC BFErrorClearLast (Bd *BoardId*)

Description Removes the top error on the error stack.

Parameters *Board*

Board ID.

Returns

BF_OK	If no errors on stack.
Non-zero	Error getting top error.

Comments This function removes the top error on the error stack.

49.8 BFErrorDefaults

Prototype BFRC BFErrorDefaults (Bd *BoardId*)

Description Puts all error destinations in their default mode.

Parameters *Board*

Board ID.

Returns

BF_OK	If no errors on stack.
Non-zero	Error getting top error.

Comments Enable or Disable all error destinations so that they are set their default mode.

49.9 BFErrorGetMes

Prototype BFRC BFErrorGetMes(Bd *Board*, BFRC *ErrorRC*, PBFCHAR *Message*, BFU32 *MessageBufSize*);

Description Retrieves a string containing the error text for the given error.

Parameters

Board

Board ID.

ErrorRC

The error number to retrieve the error text of.

Message

A pointer to a character array which will contain the error string when returned.

MessageBufSize

The size of the buffer pointed to by *Message*.

Returns

BF_OK	If no errors on stack.
Non-zero	Error getting error text.

Comments This function can be used to retrieve text of error messages as a string. This text will be the same as is displayed in the error dialogs.

BitFlow Register Access

Chapter 50

50.1 Introduction

These functions allow an application to read and write directly to every bit on the any of the BitFlow boards. For a full description of the individual bit names and their functions, refer to the corresponding Hardware Reference Manual.

The most important functions in this section are BFRegPoke and BFRegPeek. The other functions are for more esoteric uses that treat registers as generic objects. Generally, these other functions are not useful in customer applications.

50.2 BFRegPeek

Prototype BFRC BFRegPeek(Bd(Bd *Board*, BFU32 *RegId*)

Description Reads a bit field out of a full 32-bit register.

Parameters *Board*

Board ID.

RegId

Register ID.

Returns The bit field value.

Comments Register IDs for partial registers (bit fields) and for full registers (32-bits) are intermingled into the same table. Function BFRegFlags may be used to distinguish the two register types (BFF_NSET for bit field registers, BFF_WSET for full registers).

Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

50.3 BFRegPeekWait

Prototype BFRC BFRegPeekWait(Bd *Board*, BFU32 *RegId*, BFU32 *WaitValue*, BFU32 *WaitMilliseconds*)

Description Waits for a register value to match a desired value or return after a time-out.

Parameters

Board

Board ID.

RegId

Register ID.

WaitValue

Register value to wait on.

WaitMilliseconds

Wait time-out in milliseconds.

Returns The desired register value or the register contents at time-out.

Comments The register value is immediately tested against the *WaitValue* and will return if the values match.

BFRegPeekWait spins on the register contents during the current threads entire time slice. Processor time is not shared back to other threads.

The minimum wait time will always be at least as long as the specified wait regardless of clock granularity.

The actual maximum wait time is shown in Table 50-1.

Table 50-1 Maximum Wait Time

Algorithm	Method
T + 2G - T% - 1	when T%G != 0
T + G – 1	when T%G == 0

Where:

T = WaitMilliseconds
G = The clock granularity in milliseconds

50.4 BFRegPoke

Prototype BFRC BFRegPoke(Bd *Board*, BFU32 *RegId*, BFU32 *RegValue*)

Description Writes a value to a register.

Parameters

Board

Board ID.

RegId

Register ID.

RegValue

Value to write into the register.

Returns

BF_OK	If successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

Full (32-bit) registers are written directly. A read-modify-write is used to set bit field registers within a full register.

50.5 BFRegRMW

Prototype BFRC BFRegRMW(Bd *Board*, BFU32 *RegId*, BFU32 *RegValue*, BFU32 *RegMask*)

Description Read-modify-write to a masked area within a register.

Parameters

Board

Board ID.

RegId

Register ID.

RegValue

Value to write into the register.

RegMask

Mask bits defining register bits to be modified.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

The *RegValue* is masked with *RegMask* and the result is masked into the target register. Normally, you can use the function BFRegPoke as it automatically does a read-modify-write operation for predefined bit fields.

50.6 BFRegName

Prototype BFRC BFRegName(Bd *Board*, BFU32 *RegId*, LPSTR *pRegName*, BFU32 *Size*)

Description Gets a register's name.

Parameters

Board

Board ID.

RegId

Register ID.

pRegName

Pointer to register name storage.

Size

Register name array size.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

There are RegCount register IDs from ID 0 to ID RegCount - 1.

The register name will be truncated to fit the provided name buffer.

50.7 BFRegFlags

Prototype BFRC BFRegFlags(Bd *Board*, BFU32 *RegId*, PBFU32 *FlagsPtr*)

Description Gets a register's type flags.

Parameters

Board

Board ID.

RegId

Register ID.

FlagsPtr

Pointer to register flag storage.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

There are RegCount register IDs from ID 0 to ID RegCount - 1.

50.8 BFRegShift

Prototype BFRC BFRegShift(Bd *Board*, BFU32 *RegId*, PBFU32 *ShiftPtr*)

Description Gets a register's bit field shift count.

Parameters

Board

Board ID.

RegId

Register ID.

ShiftPtr

Pointer to shift count storage.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments The register shift field count is the number of bit positions a register value must be shifted to the left to fall within the register's bit field.

Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

Example

The following code uses BFRegObjectId, BFRegShift and BFRegMask to extract bit field REG_MUXC out of the 32-bit register, REG_CON0. The same result may be obtained by calling BFRegPeek(Board, REG_MUXC).

```
BFU32 ParentValue, ChildValue, Parent, Shift, Mask
BFRegObjectId(Board, REG_MUXC, &Parent)
BFRegShift(Board, REG_MUXC, &Shift)
BFRegMask(Board, REG_MUXC, &Mask)
ParentValue = BFRegPeek(Board, Parent)
ChildValue = (ParentValue & Mask) >> Shift
```

50.9 BFRegMask

Prototype BFRC BFRegMask(Bd *Board*, BFU32 *RegId*, PBFU32 *MaskPtr*)

Description Gets a register's bit field mask.

Parameters

Board

Board ID.

RegId

Register ID.

MaskPtr

Pointer to storage for the register's bit field mask.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments A bit field mask is used to access bit field data stored in a full 32-bit register.

Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

Example

The following code uses BFRegObjectId, BFRegShift and BFRegMask to extract bit field REG_MUXC out of the 32-bit register, REG_CON0. The same result may be obtained by calling BFRegPeek(Board, REG_MUXC).

```
BFU32 ParentValue, ChildValue, Parent, Shift, Mask
BFRegObjectId(Board, REG_MUXC, &Parent)
BFRegShift(Board, REG_MUXC, &Shift)
BFRegMask(Board, REG_MUXC, &Mask)
ParentValue = BFRegPeek(Board, Parent)
ChildValue = (ParentValue & Mask) >> Shift
```

50.10 BFRegObjectId

Prototype BFRC BFRegObjectId(Bd *Board*, BFU32 *RegId*, PBFU32 *ObjectIdPtr*)

Description Gets a register's wide register object ID.

Parameters

Board

Board ID.

RegId

Register ID.

ObjectIDPtr

Pointer to wide register object ID storage.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments A bit field register's parent ID is used with a register's bit field mask and bit field count to directly access bit fields within a full 32-bit register.

Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

Example

The following code uses BFRegObjectId, BFRegShift and BFRegMask to extract bit field REG_MUXC out of the 32-bit register, REG_CON0. The same result may be obtained by calling BFRegPeek(Board, REG_MUXC).

```
BFU32 ParentValue, ChildValue, Parent, Shift, Mask
BFRegObjectId(Board, REG_MUXC, &Parent)
BFRegShift(Board, REG_MUXC, &Shift)
BFRegMask(Board, REG_MUXC, &Mask)
ParentValue = BFRegPeek(Board, Parent)
ChildValue = (ParentValue & Mask) >> Shift
```

50.11 BFRegSupported

Prototype BFBOOL BFRegSupported(Bd *Board*, BFU32 *RegId*)

Description Returns whether a register is support on the board or not.

Parameters

Board

Board ID.

RegId

Register ID.

Returns

TRUE	Register is on board.
FALSE	Register is not on board.

Comments This function can be called to determine if a register is on a board. A register identified by *RegId* may be on more than one type of board, or just on one board. This function will help identify if the register is on the board.

Register IDs are declared in "BFTabRegister.h". See the corresponding Hardware Reference Manual for a description of each bit.

50.12 BFRegAddr

Prototype BFBOOL BFRegAddr(Bd *Board*, BFU32 *RegId*, PBU32 *AddrPtr*)

Description Returns wide register ID of the given bitfield

Parameters

Board

Board ID.

RegId

Register ID.

AddrPtr

Wide register ID that contains the bitfield RegID.

Returns

BF_OK	Function successful.
BF_BAD_BIT_ID	Illegal register ID.

Comments This function can be used to find the wide (i.e. 32-bit) register ID that a give bitfiled belongs to.

BitFlow Version Control Functions

Chapter 51

51.1 Introduction

These functions are provided so that applications can cross check the version of the DLLs that is on the runtime machine with the version of the SDK they were built with. The variables BF_SDK_VERSION_MAJOR and BF_SDK_VERSION_MINOR are defined in the header files and should match the values returned from these function at runtime. If there is a miss match then this usually means the DLLs on the runtime machine are not compatible with the application.

51.2 BFDriverVersion, R2DVersion, BFDVersion, BFErVersion, DispSurfVersion, DDrawSurfVersion, BitDirectSurfVersion, CiDVersion, R64DVersion

Prototype

BFRC BFDVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC R2DVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC CiDVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC BFDriverVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC BFErVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC DispSurfVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC DDrawSurfVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC BitDirectSurfVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)
BFRC R64DVersion(PBFU32 *pMajorVersion*, PBFU32 *pMinorVersion*)

Description

Returns the current version of their corresponding DLL.

Parameters

pMajorVersion

Pointer to a BFU32. When the function returns, it contains the major version number. If the highest order bit is set, the DLL is a debug version.

pMinorVersion

Pointer to a BFU32. When the function returns, it contains the minor version number.

Returns

BF_OK	In all cases.

Comments

These functions return the version of their corresponding DLLs. These function can be used to make sure application are working with the correct DLLs. The variables BF_SDK_VERSION_MAJOR and BF_SDK_VERSION_MINOR are always defined in the header files of the SDK. These variables can be used in an application to determine what version of the SDK the application was built with. These variables can also be compared to the values returned from the above functions to maintain version consistency between the application and the current DLLs.

The highest order bit (bit 31) will be set in the *pMajorVersion* parameter upon return if the current DLL is a debug version.

51.3 BFBuildNumber

Prototype BFRC BFBuildNumber(Bd *Board*, PBFU32 *pBuildNumber*)

Description Returns the build number of the current installation of the SDK and BitFlow Driver.

Parameters

Board

Board ID.

pBuildNumber

Returns the current build number.

Returns

BF_OK	In all cases.

Comments Even though each SDK release has a unique version number, during the release process the final release is built a number of times before the final release candidate is released. Each time the release is build the build number is incremented. It is sometime helpful to know exactly which build is currently installed, though this information is most useful during Beta and release candidate testing.

51.4 BFReadHWRevision

Prototype BFRC BFReadHWRevision(Bd *Board*, PBFCHAR *HWRevision*, BFU32 *HWRevsion-Size*)

Description Returns the boards hardware revision information.

Parameters *Board*

Board ID.

HWRevsion

Returns a string containing the board's hardware revision information.

HWRevisionSize

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *HWRevsion*.

Returns

BF_OK	In all cases.

Comments This function can be used to determine the hardware revision of certain models of BitFlow frame grabbers. Not all models have hardware revision information.

51.5 BFReadFWRevision

Prototype

BFRC BFReadFWRevision(Bd *Board*, PBFCHAR *HWRevision*, BFU32 *HWRevsion-Size*)

Description

Returns the boards firmware revision information on boards that do not download firmware on power up.

Parameters

Board

Board ID.

HWRevsion

Returns a string containing the board's hardware revision information.

HWRevisionSize

This parameter should contain the size of the buffer (in bytes) pointed to by the parameter *HWRevsion*.

Returns

BF_OK	In all cases.

Comments

This function can be used to determine the hardware revision of certain models of BitFlow frame grabbers, primarily the Karbon-CXP family. Not all models have firmware revision information.

BitFlow Miscellaneous Functions

Chapter 52

52.1 Introduction

This chapter contains functions that do not fit into any of the previous categories.

52.2 BFQTabModeRequest

Prototype BFRC BFQTabModeRequest(Bd *BoardId,* BFU32 *ModeRequested,* PBFU32 *ModeImplemented*)

Description Put the board in to a particular DMA mode: host QTABs or board QTABs

Parameters *Board*

Board ID.

ModeRequested

The desired DMA mode. Must be one of:

BFQTabModeHost - host QTABs mode is requested
BFQTabModeBoard - board QTABs mode is request

ModeImplemented

Returned variable containing the mode that was actually implemented. This will be one of:

BFQTabModeHost - host QTABs were implemented
BFQTabModeBoard - board QTABs were implemented

Returns

BF_OK	If no errors on stack.
Error number	Top error on error stack.

Comments Traditional DMA operations are by performed programming a DMA engine with a set of instructions telling the engine to where to write the data and how much data to write. Once instruction is executed, the DMA engine must be reprogrammed from the host CPU for the next block of data to be written. This process becomes a burden in any modern operating system that uses virtual memory. These systems break physical memory in to small pages (typically 4K bytes), requiring a separate DMA operation for each page. Even small image buffers require frequent reprogramming by the CPU. This reprogramming takes CPU cycles away from other tasks such as image processing. Scatter Gather DMA solves this problem by building a list of DMA instructions for the entire image buffer before during setup time. Once the DMA operation has begun, the DMA engine fetches and performs each DMA instruction in turn. This technique requires no CPU activity to DMA to even the largest image buffer or sequence of buffers. The scatter gather DMA instruction can be stored on the board in SRAM, which can limit the total size of the destination buffer, or in host memory, where the size of the destination buffer (or buffers) is limited only by the amount of memory in the PC.

This function is used to select where the scatter gather instructions should be stored, either in host memory or on the board. Some BitFlow boards support both methods, some boards support just one method or the other, and some boards support both but require a firmware change. For this reason the returned value *ModeImplemented* should always be examined when this function returns to make sure the desired mode was implemented.

All of the high level functions in the BitFlow SDK work identically regardless of the mode the board is in. The only difference will be when extremely large images are being acquired, in which case functions may fail in board QTab mode, or when building QTab chains, which only works in host QTab mode.

Both DMA methods are identical in terms of PCI bandwidth performance.

Note: Not this function is only for use with RoadRunner models.

52.3 BFChainSIPEnable

Prototype BFRC BFChainSIPEnable(Bd *BoardId*)

Description Enables Start-Stop Interrupt Processing mode.

Parameters *Board*

Board ID.

Returns

BF_OK	If no errors on stack.
Error number	Top error on error stack.

Comments This function enables Start-Stop Interrupt Processing (SIP). This processing is used to reset the DMA engine in a kernel interrupt service routine.

When the board is in start-stop mode, the DMA is terminated before the frame is completely acquired. This termination leaves the DMA engine in an unknown state. The DMA engine must be reset and setup for the next host buffer before the next frame starts. Ordinarily this reset is performed by the application at the user level. However, in the case of a multi threaded application, the reset thread may not be able to reset the DMA engine before the beginning of the next frame (because of CPU load and thread priorities). To solve this problem the BitFlow SDK implements a DMA engine reset in the kernel level interrupt service routine. This code has higher priority than any user level threads. The latency and execution time of the SIP reset is minimized thus reducing the required minimum time between frames. This function turns on this functionality.

SIP only works (and is only required) when the board is in start-stop triggering mode (variable size image acquisition) and when a host QTab chain has been created and engaged. This function must be called before acquisition has started but after the QTab chain is created. This function enable the SIP resetting of the DMA engine, you must call BFChainSIPDisable to turn the SIP off. This SIP is based on an interrupt which occurs at the end of the variable length frame. On the Road Runner/R3 the CTAB column, IRQ, interrupt (put an entry at location zero) is used. On the R64 the EOF interrupt is used.

The example application Flow demonstrates usage of this function.

52.4 BFChainSIPDisable

Prototype BFRC BFChainSIPDisable(Bd *BoardId*)

Description Disables Start-Stop Interrupt Processing mode.

Parameters *Board*

Board ID.

Returns

BF_OK	Function succeeded.
Non-zero	Function failed.

Comments See BFChainSIPEnable for details.

52.5 BFStructItemGet

Prototype BFRC BFStructItemGet(Bd *Board*, PBFCNF *pBase*, BFU32 *ID*, BFU32 *Indx*, PBFVOID *pVal1*, PBFVOID *pVal2*, PBFU32 *pDisp*, BFU32 *DestSize*, PBFU32 *pASize*)

Description Gets a configuration value from a configuration structure.

Parameters

Board

Board ID.

pBase

Pointer to a configuration structure.

ID

Token identifying the configuration item. See R64Entry.h or R2Entry.h for a list of tokens.

Indx

If the configuration item is a list, this is in number in the list to get.

pVal1

A pointer that will contain the value when the function returns.

pVal2

If the item has two values, this parameter will contain the second item when the function returns.

pDisp

If the item is a list, this will contain the disposition of the list search (by incrementing *Indx*) when the function returns. This value will be one of:

BFCNF_ENDOFLIST - the list number *Indx* does not exist.
BFCNF_OK - the list number Indx is valid, and it's values are return in *pVal1* and *pVal2*.

DestSize

Total size of the memory (in bytes) pointed to by both *pVal1* and *pVal2*. This is mainly used when the item is a string of unknown length.

pASize

The total size of the memory (in bytes) return in both the variables *pVal1* and *pVal2*.

Returns

BF_OK	If no errors on stack.
Non-zero	Error getting top error.

Comments

This function is used to extract a configuration value from a configuration structure in memory. The configuration information is stored in a compact binary structure of variable length. The information can on be extracted using this function. The parameter ID is identifies the item to be extracted. There are many different types of configuration items, this function supports extracting all of them. To get all the item in list, call this function can be called in a loop, incrementing *Indx* each time, until *pDisp* returns BFCNF_ENDOFLIST.

Typically this function is used to get a parameter out of a camera configuration file that has been loaded into memory. In general the board and camera inquiry functions are much easier to use, this function is only needed for parameters that are not available from the inquiry functions.

52.6 BFStructItemSet

Prototype BFRC BFStructItemSet(Bd *Board*, PBFCNF *pBase*, BFU32 *ID*, BFU32 *Indx*, PBFVOID *pVal1*, PBFVOID *pVal2*, BFU32 *SourceSize*)

Description Inserts a configuration value into a configuration structure.

Parameters

Board

Board ID.

pBase

Pointer to a configuration structure.

ID

Token identifying the configuration item. See R64Entry.h or R2Entry.h for a list of tokens.

Indx

If the configuration item is a list, this is in number in the list to insert.

pVal1

A pointer to the value to insert.

pVal2

If the item has two values, this parameter is a pointer to the second value to insert.

SourceSize

Total size of the memory (in bytes) pointed to by both *pVal1* and *pVal2*.

Returns

BF_OK	If no errors on stack.
Non-zero	Error getting top error.

Comments This function is used to insert a value into a configuration structure. This function is the only way to update a configuration structure. Inserting values in a configuration structure only makes sense if the structure is then used by some other high level function for setting up acquisition. Generally is it easier to make changes directly to the board.

52.7 BFTick

Prototype BFTickPtr BFTick(BFTickPtr *TickPtr*)

Description Gets the current time in units of ticks.

Parameters *TickPtr*

Pointer to a tick structure. When the function returns, this contains the current tick count.

Returns The current tick count.

Comments This function gets the current tick count. Generally this function is called twice, once before an event and once after. The two tick counts are then passed to the function BFTickDelta to calculate the elapsed time.

52.8 BFTickRate

Prototype BFU32 BFTickRate(void)

Description Gets the current tick rate.

Parameters

Returns The current tick rate in ticks per second.

Comments This function returns the operating systems tick rate in units of ticks per second.

52.9 BFTickDelta

Prototype BFU32 BFTickDelta(BFTickPtr *t0*, BFTickPtr *t1*)

Description Returns the amount of time that has elapse between to tick values.

Parameters *t0*

Pointer to a tick structure, containing the start time of an event.

t1

Pointer to a tick structure, containing the end time of an event.

Returns The time between the two events, in milliseconds.

Comments This function calculates the time between to events. General BFTIck is call twice, once before an event and once after. The two tick counts are then passed to this function to calculate the elapsed time.

52.10 BFFine

Prototype BFU64 BFFine(Bd *Board*)

Description Returns the current fine grain tick value (CPU clock counter).

Parameters *Board*

Handle to board.

Returns The current fine grain tick count.

Comments This function gets the fine grain current tick count. Generally this function is called twice, once before an event and once after. The two tick counts are then passed to the function BFFineDelta to calculate the elapsed time.

The value return from this function can be converted to milliseconds by dividing the value return from this function by the value returned from BFFineRate

52.11 BFFineRate

Prototype BFU64 BFTickRate(Bd *Board*)

Description Gets the current fine grain tick rate.

Parameters *Board*

Handle to board.

Returns The current fine grain tick rate in ticks per second.

Comments This function returns the CPU clock tick rate in units of ticks per second. The value returned from this function should only be used the other BFFineXXX functions.

52.12 BFFineDelta

Prototype BFU64 BFFineDelta(Bd *Board,* BFU64 *t0,* BFU64 *t1,* BFBOOL *AbsValue*)

Description Returns the amount of time that has elapse between to fine grain tick values.

Parameters

Board

Handle to board.

t0

A fine grain tick count, containing the start time of an event.

t1

A fine grain tick count, containing the end time of an event.

AbsValue

Set to TRUE to have the function return the absolute value of the difference between t0 and t1..

Returns The time between the two events, microseconds.

Comments This function calculates the time between to events. General BFFine is called twice, once before an event and once after. The two tick counts are then passed to this function to calculate the elapsed time.

52.13 BFFineWait

Prototype void BFTickRate(Bd *Board,* BFU32 *Microseconds*)

Description Wait the given number of microseconds.

Parameters *Board*

Handle to board.

Microseconds

Number of microseconds to wait.

Comments This function delays execution the given number of microseconds. The function does consume CPU cycles.

52.14 BFDrvReady

Prototype BFRC BFDrvReady(BFU32 *TimeoutSeconds*)

Description Waits for the driver to start.

Parameters *TimeoutSeconds*

Number of seconds to wait for the driver to start before returning.

Returns

BF_OK	Driver is running.
BFSYS_ERROR_ NOTREADY	Time-out has elapsed and the driver is not ready.

Comments This function waits for the kernel level driver to start. This function is useful if an application is automatically run when a system is booted. The start order of drivers is not predictable, and the BitFlow driver may not yet be started by the time and application is started. This function can be called to make sure the driver is started before making any calls to the BF SDK.

52.15 BFIsCL

Prototype BFBOOL BFIsCL(Bd *Board*)

Description Returns type of board

Parameters *Board*

Board ID.

Returns

TRUE	The board is a camera link board.
FALSE	The board is not a camera link board.

Comments This function can be called to determine if the board is a camera link board or not. This function is usually on required in special situations requiring low level access to the board.

52.16 BFIsR3

Prototype BFBOOL BFIsR3(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an R3 board.
FALSE	The board is not an R3 board.

Comments This function can be called to determine if the board is a R3 or a Road Runner. This function is usually on required in special situations requiring low level access to the board.

52.17 BFIsR2

Prototype BFBOOL BFIsR2(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an R2 board.
FALSE	The board is not an R2 board.

Comments This function can be called to determine if the board is a Road Runner. This function is usually required in special situations requiring low level access to the board.

52.18 BFIsRv

Prototype BFBOOL BFIsRv(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Raven board.
FALSE	The board is not a Raven board.

Comments This function can be called to determine if the board is a Raven. This function is usually on required in special situations requiring low level access to the board.

52.19 BFIsR64Board

Prototype BFBOOL BFIsR64Board(Bd *Board*)

Description Returns exact board type

Parameters *Board*

Board ID.

Returns

TRUE	The board is an R64/R64e board.
FALSE	The board is not an R64/R64e board.

Comments This function can be called to determine if the board is a R64 or R64e. This function is usually on required in special situations requiring low level access to the board.

52.20 BFIsR64

Prototype BFBOOL BFIsR64(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board has the R64 architecture.
FALSE	The board does not have the R64 architecture.

Comments This function can be called to determine if the current board has the R64 architecture. Current this means that this function will return true if the board is an R64, R64e, Karbon, Neon or Alta. It will return false for a Road Runner, R3 or Raven.

52.21 BFIsPMC

Prototype BFBOOL BFIsPMC(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is a PMC board.
FALSE	The board is not a PMC board.

Comments This function can be called to determine if the board is a PMC. This function is usually on required in special situations requiring low level access to the board.

52.22 BFIsPLDA

Prototype BFBOOL BFIsPLDA(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board uses the PLDA engine.
FALSE	The board does not use the PLDA engine/

Comments This function is usually on required in special situations requiring low level access to the board.

52.23 BFIsKbn

Prototype BFBOOL BFIsKbn(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Karbon board.
FALSE	The board is not an Karbon board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.24 BFIsKbn4

Prototype BFBOOL BFIsKbn4(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Karbon 4 board.
FALSE	The board is not an Karbon 4 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.25 BFIsKbn2

Prototype BFBOOL BFIsKbn2(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Karbon 2 board.
FALSE	The board is not an Karbon 2 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.26 BFIsKbnBase

Prototype BFBOOL BFIsKbnBase(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an base Karbon board.
FALSE	The board is not an base Karbon board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.27 BFIsKbnFull

Prototype BFBOOL BFIsKbnFull(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is a full Karbon board.
FALSE	The board is not a full Karbon board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.28 BFIsKbnCXP

Prototype BFBOOL BFIsKbnCXP(Bd *Board*)

Description Returns true if the board is a Karbon-CXP family board

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Karbon-CXP board.
FALSE	The board is not a Karbon-CXP board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.29 BFIsKbnCXP1

Prototype BFBOOL BFIsKbnCXP1(Bd *Board*)

Description Returns true if the board is a Karbon-CXP1 board

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Karbon-CXP1 board.
FALSE	The board is not a Karbon-CXP1 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.30 BFIsKbnCXP2

Prototype BFBOOL BFIsKbnCXP2(Bd *Board*)

Description Returns true if the board is a Karbon-CXP2 board

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Karbon-CXP2 board.
FALSE	The board is not a Karbon-CXP2 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.31 BFIsKbnCXP4

Prototype BFBOOL BFIsKbnCXP4(Bd *Board*)

Description Returns true if the board is a Karbon-CXP4 board

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Karbon-CXP4 board.
FALSE	The board is not a Karbon-CXP4 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.32 BFIsNeonBase

Prototype BFBOOL BFIsNeonBase(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is a base Neon board.
FALSE	The board is not an base Neon board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.33 BFIsNeonD

Prototype BFBOOL BFIsNeonD(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Neon-CLD.
FALSE	The board is not a Neon-CLD.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.34 BFIsNeonQ

Prototype BFBOOL BFIsNeonQ(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Neon-CLQ.
FALSE	The board is not a Neon-CLQ.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.35 BFIsNeonDif

Prototype BFBOOL BFIsNeonDif(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Neon-DIF.
FALSE	The board is not a Neon-DIF.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.36 BFIsAlta

Prototype BFBOOL BFIsAlta(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Alta board.
FALSE	The board is not an Alta board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.37 BFIsAlta1

Prototype BFBOOL BFIsAlta1(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Alta 1 board.
FALSE	The board is not an Alta 1 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.38 BFIsAlta2

Prototype BFBOOL BFIsAlta2(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Alta 2 board.
FALSE	The board is not an Alta 2 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.39 BFIsAlta4

Prototype BFBOOL BFIsAlta4(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Alta 4 board.
FALSE	The board is not an Alta 4 board.

Comments This function is usually on required in special situations requiring low level access to the board.

52.40 BFIsSlave

Prototype BFBOOL BFIsSlave(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is a slave on a multi-VFG board.
FALSE	The board is not a slave on a multi-VFG board.

Comments Multi-VFG boards have one master VFG and one or more slave VFGs. Some operations can only be performed on the master VFG. This function is usually on required in special situations requiring low level access to the board.

52.41 BFIsAxn

Prototype BFBOOL BFIsAxn(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Axion
FALSE	The board is not an Axion

Comments This function is usually on required in special situations requiring low level access to the board.

52.42 BFIsAxn1xE

Prototype BFBOOL BFIsAxn1xE(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Axion-1xE
FALSE	The board is not an Axion-1xE

Comments This function is usually on required in special situations requiring low level access to the board.

52.43 BFIsAxn2xE

Prototype BFBOOL BFIsAxn2xE(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Axion-2xE
FALSE	The board is not an Axion-2xE

Comments This function is usually on required in special situations requiring low level access to the board.

52.44 BFIsAxn2xB

Prototype BFBOOL BFIsAxn2xB(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Axion-2xB
FALSE	The board is not an Axion-2xB

Comments This function is usually on required in special situations requiring low level access to the board.

52.45 BFIsAxn4xB

Prototype BFBOOL BFIsAxn4xB(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Axion-4xB
FALSE	The board is not an Axion-4xB

Comments This function is usually on required in special situations requiring low level access to the board.

52.46 BFIsMaster

Prototype BFBOOL BFIsMaster(Bd *Board*)

Description Returns board family

Parameters *Board*

Board ID.

Returns

TRUE	The board is a master on a multi-VFG board.
FALSE	The board is not a master on a multi-VFG board.

Comments Multi-VFG boards have one master VFG and one or more slave VFGs. Some operations can only be performed on the master VFG. This function is usually on required in special situations requiring low level access to the board.

52.47 BFIsAltaAN

Prototype BFBOOL BFIsAltaAN(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Alta-AN.
FALSE	The board is not a Alta-AN.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.48 BFIsAltaCO

Prototype BFBOOL BFIsAltaCO(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Alta-CO.
FALSE	The board is not a Alta-CO.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.49 BFIsAltaYPC

Prototype BFBOOL BFIsAltaYPC(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Alta-YPC.
FALSE	The board is not a Alta-YPC.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.50 BFIsEncDiv

Prototype BFBOOL BFIsEncDiv(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board supports the encoder divider function.
FALSE	The board does not support the encoder divider function.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.51 BFIsNTG

Prototype BFBOOL BFIsNTG(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board supports the New Timing Generator (NTG) function.
FALSE	The board does not support the NTG.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.52 BFIsGn2

Prototype BFBOOL BFIsGn2Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is PCIe Gen2 architecture board.
FALSE	This board is a previous architecture.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.53 BFIsCtn

Prototype BFBOOL BFIsCtn(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Cyton.
FALSE	The board is not a Cyton.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.54 BFIsCXP

Prototype BFBOOL BFIsCXP(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is for use with CoaXPress cameras.
FALSE	The board is not for use with CoaXPress cameras.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.55 BFIsCXP2

Prototype BFBOOL BFIsCXP2(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board has two CoaXPress inputs
FALSE	The board has less than or more than two CoaXPress inputs

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.56 BFIsCXP4

Prototype BFBOOL BFIsCXP4(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board has four CoaXPress inputs
FALSE	The board has less than or more than four CoaXPress inputs

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.57 BFIsAon

Prototype BFBOOL BFIsAon(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is in the Aon family
FALSE	The board is not in the Aon family

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.58 BFIsAonCXP1

Prototype BFBOOL BFIsAonCXP1(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Aon-CXP1 model
FALSE	The board is not an Aon-CXP1 model

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.59 BFIsAxnII

Prototype BFBOOL BFIsAxnII(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is an Axion Mark II model
FALSE	The board is not an Axion Mark II model

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.60 BFIsCtnII

Prototype BFBOOL BFIsCtnII(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Cyton MarkII model
FALSE	The board is not in a Cyton Mark II model

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.61 BFIsClx

Prototype BFBOOL BFIsClx(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is Claxon model
FALSE	The board is not a Claxon model

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.62 BFIsClxCXP2

Prototype BFBOOL BFIsClxCXP2(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Claxon CXP 2 model
FALSE	The board is not a Claxon CXP 4 model

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.63 BFIsClxCXP4

Prototype BFBOOL BFIsAonCXP1(Bd *Board*)

Description Returns status of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board is a Claxon CXP4 model
FALSE	The board is not in Claxon CXP4 model

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.64 BFIsSynthetic

Prototype BFBOOL BFIsSynthetic(Bd *Board*)

Description Returns true if the board is configured with a synthetic camera configuration file.

Parameters *Board*

Board ID.

Returns

TRUE	The board is configured to acquire from its internal synthetic image generator
FALSE	The board is configured to acquire from an external camera

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.65 BFHasSerialPort

Prototype BFBOOL BFHasSerialPort(Bd *Board*)

Description Returns availability of a serial port of current board.

Parameters *Board*

Board ID.

Returns

TRUE	The board has a CL serial port.
FALSE	The board does not have a CL serial port.

Comments This function can be used to quickly determine if the board is of the given type or has the given feature.

52.66 BFCurrentTimeGet

Prototype BFU32 BFICurrentTimeGet(Bd *Board*, PBFTimeStruct *Time*)

Description Returns a time structure filled with the system time at the point the function was called.

Parameters

Board

Board ID.

Time

Pointer to the structure (allocated by the user) to hold the time structure.

Returns

BF_OK	In all cases.

Comments The BFTimeStruct structure contains the year (20XX), mon (1..12), yday (1..366), day (1..31), hour (0..23), min (0..59), sec (0..59), and msec (0..999). When this function is called, the time structure is filled with the current system time. The time structure will remain unchanged until this function is called again, at which point the time structure will be overwritten with the new time. Calling function BFTimeStructInit will reset all the information in the time structure to a value of zero.

52.67 BFTimeStructInit

Prototype BFU32 BFTimeStructInit(Bd *Board*, PBFTimeStruct *Time*)

Description Sets all the information in the BitFlow time structure to zero.

Parameters *Board*

Board ID.

Time

Pointer to the structure (allocated by the user) to be initialized.

Returns

BF_OK	In all cases.

Comments

52.68 BFHiResTimeStampInit

Prototype BFU32 BFTimeStructInit(PBFTime *pStartTime*)

Description Returns the initialized base time that is required for the BFHiResTimeStamp function.

Parameters *pStartTime*

The returned base time.

Returns

BF_OK	If the function was successful.
BF_HIGH_RES_TIMER_ERR	Could not retrieve the high-resolution counter.

Comments This function returns the base time that will be required for the BFHiResTimeStamp function. The base time may drift from actual time when used for long periods of time. (days, weeks, months, or years) Because of this it may be necessary to re-initialize the base time to be accurate with the actual time. Because the high-resolution time stamp uses the CPU clock to calculate the time, the drift will vary between systems. The user will need to experiment with the how often to re-initialize the base time.

52.69 BFHiResTimeStamp

Prototype BFU32 BFHiResTimeStamp(BFTime *StartTime*, PBFTime *pCurTime*)

Description Returns the system time at the moment the function is called with a high resolution timer.

Parameters

StartTime

The base time returned from the BFHiResTimeStampInit function.

pCurTime

The returned high resolution time stamp.

Returns

BF_OK	If the function was successful.
BF_HIGH_RES_TIMER_ERR	Could not retrieve the high-resolution counter.

Comments This time stamp uses the CPU clock to determine time. Hence, the faster the CPU the more accurate the time stamp. Using a modern day CPU the time stamp should be accurate to at least +/- 1mS. It is recommended that the user benchmark the time stamp for their particular system.

52.70 BFHiResTimeStampEx

Prototype BFU32 BFHiResTimeStampEx(BFTime *StartTime*, BFU64 *HiResStampTime*, PBFTime *pCurTime*)

Description Converts a high resolution time stamp into a BFTime struct..

Parameters

StartTime

The base time returned from the BFHiResTimeStampInit function.

HiResStampTime

The high resolution time stamp to convert.

pCurTime

The returned high resolution time stamp.

Returns

BF_OK	If the function was successful.
BF_HIGH_RES_TIMER_ERR	Could not retrieve the high-resolution counter.

Comments The function BFHiRestTimeStamp returns the current time (as an offset from the start time). However this function returns the time when a high resolution time stamp was taken.

52.71 DoBrdOpenDialog

Prototype BFUPTR DoBrdOpenDialog(BFU32 *Options*, PBFU32 *FamilyFilter*, PBFU32 *pFamily*, PBFU32 *pBrdNum*, PBFU32 *pDoInit*, PBFU32 *pSerPortNum*)

Description Opens a dialog to prompt the user which board to open.

Parameters *Options*

Options for what the dialog board will look like and what boards to present to be opened:

BOD_NONE - Both the "Open Initalized" and "Just Open" buttons will be part of the dialog.
BOD_HIDEJUSTOPEN - Hides the "Just Open" button on the dialog.
BOD_CLONLY - Only shows CL boards to be opened.
BOD_HIDEOPENINIT - Hides the "Open Initalized" button on the dialog.

The options can be ORed together to customize the dialog.

FamilyFilter

Option for which family of board or boards to present to be opened:

FF_ROADRUNNER - Only shows RoadRunner and R3 boards to be opened.
FF_R64 - Only shows R64 boards to be opened.
FF_BITFLOW_MODERN - Shows all supported BitFlow boards to be opened.

pFamily

Returns the family type of the board that was asked to be opened. The family type can be one of the following:

FF_ROADRUNNER - A RoadRunner or R3 to be opened.
FF_R64 - A R64 board to be opened.

pBrdNum

Returns the board number of the board that is to be opened.

pDoInit

Returns whether or not the board is to be opened initalized or not. The returned value can be one of the following:

0 - Board will open normally but not initialized.
BFSysInitialize - Board will opened and initialized.

pSerPortNum

Returns the serial port number if the board is a CL board. If the board is not a CL board, the returned value will be 0xFFFFFFFF.

Returns

BRD_DLG_OK	A board was selected and the user clicked the Init or Just open button, or there is only one board installed and the dialog was not displayed.
BRD_DLG_NO_BOARDS	There are no boards installed in the system.
BRD_DLG_USER_CANCEL	The user clicked the Cancel button. This is backwards compatible with the old return value of IDCANCEL.
BRD_DLG_MALLOC_ ERROR	There was an error allocating resources for the dialog.
BRD_DLG_DRIVER_ERROR	There was an error communicating with the driver.
BRD_DLG_OTHER_ERROR	There was an internal error creating the dialog.

Comments

This function provides a dialog box to prompt the user to which board they would like to open. If there is only one board installed in the system, the dialog will not be displayed and the function returns with the board information. If there is more than one board in the system the dialog allows the user to choose which board to open.

When the user chooses a board, the function returns with information about the board that can then be passed on to functions that do the actual work of opening the board.

Here are two example on how to use this function for the Ci and Bi APIs:

```
BFU32 Type, Num, Init, SerNum;

DoBrdOpenDialog(TRUE, FF_BITFLOW_MODERN, &Type, &Num, &Init,
&SerNum)       // prompt user for the board to open
CiSysBrdFind(Type, Num, &entry)      // find board
CiBrdOpen(&entry, &hBoard, Init)      // open board


DoBrdOpenDialog(TRUE, FF_BITFLOW_MODERN, &Type, &Num, &Init,
&SerNum)       // prompt user for the board to open
BiBrdOpen(Type, Num, &m_hBoard)  // open board
```

52.72 WaitDialogOpen

Prototype BFBOOL WaitDialogOpen(PBFCHAR *Msg*, PBFU32 *pHandle*)

Description A generic dialog to display different wait messages.

Parameters

Msg

The message to be displayed in the dialog box.

pHandle

A handle to the dialog box.

Returns

TRUE	If successful.
FALSE	Could not create the dialog box.

Comments This function is used primarily to give the user some feedback when there is a event happening that may require some time, such are opening a board or saving a sequence of images to disk.

This function works in conjunction with the WaitDialogClose function, to close the dialog when the event we're waiting for completes.

52.73 WaitDialogClose

Prototype BFBOOL WaitDialogClose(BFU32 *pHandle*)

Description Closes the dialog opened by WaitDialogOpen.

Parameters *pHandle*

Handle to the dialog to be closed.

Returns

TRUE	If successful.
FALSE	Function failed.

Comments This function will close the dialog box opened by the WaitDialogOpen function. Pass this function the pHandle that was returned by the WaitDialogOpen function.

52.74 WaitDialogClose

Prototype BFBOOL WaitDialogClose(BFU32 *pHandle*)

Description Closes the dialog opened by WaitDialogOpen.

Parameters *pHandle*

Handle to the dialog to be closed

Returns

TRUE	If successful.
FALSE	Function failed.

Comments This function will close the dialog box opened by the WaitDialogOpen function. Pass this function the pHandle that was returned by the WaitDialogOpen function.

52.75 ChoiceDialog

Prototype BFBOOL ChoiceDialog(PBFCHAR **ChoiceList*, BFU32 *NumChoices*, BFU32 *CurrentChoiceIndex*, PBFCHAR *Choice*, BFSIZET *ChoiceSize*)

Description Pops up a simple dialog offering the user a choice of options

Parameters

ChoiceList

An array of strings containing the list of choices

NumChoices

The number of choices in the list

CurrentChoiceIndex

The choice in the list that should be highlighted when the dialog pops up

Choice

The string corresponding to the choice that the user selected

ChoiceSize

The size of the buffer pointed to by the destination string *Choice*

Returns

TRUE	The user made a choice and hit "OK"
FALSE	The user chose "Cancel"

Comments This function is just a simple wrapper around a Win32 native dialog.

52.76 BFGetCurrentFimwareName

Prototype BFCRC BFGetCurrentFimwareName(Bd *Board*, PBFCHAR *FWRoot*, PBFCHAR *FWFileName*, BFU32 *FWFileNameSize*)

Description Returns the name of the firmware file currently download to the board.

Parameters

Board

Board ID.

FWRoot

Root or token of FPGA whose firmware file name is to be retrieved. For all modern frame grabbers this should be set to “MUX”.

FWFileName

Name of firmware file used to source the firmware

FWFileNameSize

Size of *FWFileName* (bytes)

Returns

BF_OK	Success
BF_NOT_SUPPORTED	Current board is not supported by this function
BF_REGISTRY_ERROR	Error accessing the Registry

Comments This function returns information about the firmware currently loaded on the board.

52.77 BFGetVFGNum

Prototype BFU32 BFGetVFGNum(Bd *Board*)

Description Get the VFG Number of the given board.

Parameters *Board*

Board ID.

Returns

VFG Number	In all cases.

Comments This function returns the VFG number of the give board. Some BitFlow frame grabbers have more that one Virtual Frame Grabber (VFG). They are physically a single board, but the look like multiple devices to software. For boards that support more than one VFG this function can be used to determine with VFG number the current board is.

52.78 BFReadSerialNumberString

Prototype void BFReadSerialNumberString(Bd *Board*, PBFCHAR *pSerialNumberString*, BFSIZET *SerialNumberStringSize*)

Description Returns the serial number string of the given board.

Parameters

Board

Board ID.

SerialNumberString

String to receive the serial number string. If the board does not support serial numbers or the board has not been programmed with a serial number the parameter will be returned empty, *SerialNumberString*[0] = 0.

SerialNumberStringSize

Size in bytes of the string pointed to by *SerialNumberString*.

Comments This function returns the serial number string stored on the given board. Note that this function is not supported for all models. The board does not support a serial number string then the *SerialNumberString*[0] = 0.

52.79 BFOutputDebugString

Prototype void BFOutputDebugString(PBFCHAR *OutputString)*

Description Sends a message to the BitFlow logging utility, BFLog.

Parameters *OutputString*

String to sent to logger.

Comments This function sends a message the BitFlow logging utility, called BFLog. All messages are time stamped and displayed as they are received. Other components of the BitFlow driver and DLLs will also send messages to BFLog, some message types are color coded. Message logs can be saved to disk.

BitFlow Disk I/O Functions

Chapter 53

53.1 Introduction

This chapter contains functions that are associated with reading and writing to disk.

53.2 BFIOWriteSingle

Prototype BFU32 BFIOWriteSingle(char* *FileName,* PBFU32 *pBuffer,* BFU32 *XSize,* BFU32 *YSize,* BFU32 *BitDepth,* BFU32 *Options*)

Description Writes one buffer to a file on the disk in the BMP, TIFF, or raw file format.

Parameters

FileName

The file name to be saved to disk. The file name includes the file extension. Valid file extensions are .bmp,.raw,.tif and.tiff. The file name is case insensitive.

pBuffer

Pointer to the image data.

XSize

Width of the image in pixels.

YSize

Height of the image in lines.

BitDepth

Depth of the pixels in bits.

Options

The options for saving a buffer to disk are:

SwapRGB - Swap the RGB format to BGR.
PACK32TO24BIT - Packs 32 bit (RGBX) data into 24 bit data (RGB).
UNPACKPIXELS - The pixels in the image buffers are presumed to be packed 10 or 12 bit pixels. Using this option the pixels will be unpacked into 16-bit words. This option is only supported for TIFF files.
BOTTOM_UP - Saves the data to disk upside down from what is being displayed.
OVERWRITE - If the file already exists, it will be overwritten. Default behavior is to return an error and not overwrite an existing file.

Returns

BF_OK	Function was successful.
BF_ER_FILE_NAME	No file name given.
BF_ER_BUF_POINTER	Invalid buffer pointer.

BF_ER_NUM_BUFFERS	The number of buffers must be greater than zero.
BF_ER_XSIZE	Invalid XSize. The XSize must be greater than zero.
BF_ER_YSIZE	Invalid YSize. The YSize must be greater than zero.
BF_ER_BITDEPTH_ UNKNOWN	Unknown bit depth. The bit depth must be 8, 10, 12, 14, 16, 24 or 32.
BF_ER_BITDEPTH_ SWAPRGB	The bit depth must be greater than or equal to 24 to use the SWAPRGB option.
BF_ER_PACK24_BITDETPH	Must start with 32 bit data to use the PACK32-TO24BIT option.
BF_ER_BMP_BIT_DEPTH	Invalid bit depth for BMP. BMP supports 8, 24 and 32 bit pixel depths.
BF_ER_BMP_OPEN_FILE	Failed opening BMP file.
BF_ER_BMP_FILE_HEADER	Failed writing BMP header to file.
BF_ER_BMP_DATA_WRITE	Failed writing image data to BMP file.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_TIF_BIT_DEPTH	Invalid tif bit depth. The bit depth must be 8, 10, 12, 14, 16, 24 or 32.
BF_ER_TIF_OPEN_FILE	Failed opening tif file.
BF_ER_TIF_FILE_HEADER	Failed writing tif header to file.
BF_ER_TIF_DATA_WRITE	Failed writing image data to tif file.
BF_ER_RAW_OPEN_FILE	Failed opening raw file.
BF_ER_RAW_DATA_WRITE	Failed writing image data to raw file.
BF_ER_AVI_BIT_DEPTH	Invalid bit depth for AVI. AVI supports 8, 24 and 32 bit pixel depths.
BF_ER_AVI_OPEN_FILE	Failed opening AVI file.
BF_ER_AVI_DATA_WRITE	Failed writing image data to the AVI stream.
BF_ER_CREATE_STREAM	Error creating AVI stream.
BF_ER_SAVE_OPTIONS	Error with dialog box save options for AVI.
BF_ER_COMPRESS_STREAM	Error with compressing the stream.
BF_ER_AVI_HEADER	Error putting AVI header in the stream.
BF_ER_FILE_FORMAT	Invalid file format. Please use BMP, TIFF, or raw.
BF_ER_RAW_OPEN_TEXT-FILE	Failed to open the text file to write raw image file information.
BF_ER_RAW_TEXT_WRITE	Failed to write the data to the raw text file.

Comments

This function will write image data to disk in either the BMP, tif or raw file formats.

If the raw file format is used, a text file with the same name as the file saved will also be generated that will contain the xsize, ysize and bit depth for the saved image. This information will become useful when trying to open a raw file type.

If no path is used in the file name parameter, the file will be saved in the same directory as the application is being run from.

53.3 BFIOWriteMultiple

Prototype BFU32 BFIOWriteMultiple(char* *FileName,* PBFU32 **pBufArray,* BFU32 *StartNum,* BFU32 *XSize,* BFU32 *YSize,* BFU32 *BitDepth,* BFU32 *NumBuffers,* BFU32 *Options*)

Description Writes multiple buffers to multiple files on the disk in BMP, TIFF, AVI or raw file formats.

Parameters

FileName

The file name to be saved to disk. The file name includes the file extension. Valid file extensions are .bmp, .raw, .tif, .tiff and .avi. The file name is case insensitive.

**pBufArray*

Pointer to the array of pointers that point to the image data.

StartNum

Specifies the first number to start the file names with. If StartNum = 5, the first file name will be "'XXXX00000005.BMP", but contain image data from buffer 0.

XSize

Width of the image in pixels.

YSize

Height of the image in lines.

BitDepth

Depth of the pixels in bits.

NumBuffers

The number of buffers to be written.

Options

The options for saving a buffer to disk are:

SwapRGB - Swap the RGB format to BGR.
PACK32TO24BIT - Packs 32 bit (RGBX) data into 24 bit data (RGB).
UNPACKPIXELS - The pixels in the image buffers are presumed to be packed 10 or 12 bit pixels. Using this option the pixels will be unpacked into 16-bit words. This option is only supported for TIFF files.
BOTTOM_UP - Saves the data to disk upside down from what is being displayed.
OVERWRITE - If the file already exists, it will be overwritten. Default behav-

ior is to return an error and not overwrite an existing file.

Returns

BF_OK	Function was successful.
BF_ER_FILE_NAME	No file name given.
BF_ER_BUF_POINTER	Invalid buffer pointer.
BF_ER_NUM_BUFFERS	The number of buffers must be greater than zero.
BF_ER_XSIZE	Invalid XSize. The XSize must be greater than zero.
BF_ER_YSIZE	Invalid YSize. The YSize must be greater than zero.
BF_ER_BITDEPTH_ UNKNOWN	Unknown bit depth. The bit depth must be 8, 10, 12, 14, 16, 24 or 32.
BF_ER_BITDEPTH_ SWAPRGB	The bit depth must be greater than or equal to 24 to use the SWAPRGB option.
BF_ER_PACK24_BITDETPH	Must start with 32 bit data to use the PACK32-TO24BIT option.
BF_ER_BMP_BIT_DEPTH	Invalid bit depth for BMP. BMP supports 8, 24 and 32 bit pixel depths.
BF_ER_BMP_OPEN_FILE	Failed opening BMP file.
BF_ER_BMP_FILE_ HEADER	Failed writing BMP header to file.
BF_ER_BMP_DATA_WRITE	Failed writing image data to BMP file.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_TIF_BIT_DEPTH	Invalid tif bit depth. The bit depth must be 8, 10, 12, 14, 16, 24 or 32.
BF_ER_TIF_OPEN_FILE	Failed opening tif file.
BF_ER_TIF_FILE_HEADER	Failed writing tif header to file.
BF_ER_TIF_DATA_WRITE	Failed writing image data to tif file.
BF_ER_RAW_OPEN_FILE	Failed opening raw file.
BF_ER_RAW_DATA_WRITE	Failed writing image data to raw file.
BF_ER_AVI_BIT_DEPTH	Invalid bit depth for AVI. AVI supports 8, 24 and 32 bit pixel depths.
BF_ER_AVI_OPEN_FILE	Failed opening AVI file.
BF_ER_AVI_DATA_WRITE	Failed writing image data to the AVI stream.
BF_ER_CREATE_STREAM	Error creating AVI stream.
BF_ER_SAVE_OPTIONS	Error with dialog box save options for AVI.

BF_ER_COMPRESS_ STREAM	Error with compressing the stream.
BF_ER_AVI_HEADER	Error putting AVI header in the stream.
BF_ER_FILE_FORMAT	Invalid file format. Please use bmp, tif, raw or avi.
BF_ER_RAW_OPEN_TEXT-FILE	Failed to open the text file to write raw image file information.
BF_ER_RAW_TEXT_WRITE	Failed to write the data to the raw text file.

Comments

This function will write a sequence of images to disk in either the bmp, tif, avi or raw file formats. The files will be sequentially named.

If the raw file format is used, a text file with the same name as the file saved will also be generated that will contain the xsize, ysize and bit depth for the saved image. This information will become useful when trying to open a raw file type. Only one file will be written per sequence (i.e. on file is written each time this function is called).

If no path is used in the file name parameter, the file will be saved in the same directory as the application is being run from.

53.4 BFIOReadSingle

Prototype BFU32 BFIOReadSingle(char* *FileName,* PBFU32 *pBuffer,* BFU32 *XSize,* BFU32 *YSize,* BFU32 *BitDepth*)

Description Reads in a single frame from a file on the disk into memory allocated by the user.

Parameters

FileName

The file to be read into memory. The file name includes the file path and extension. Valid file extensions are .bmp, .raw, .tif and .tiff. The file name is case insensitive.

pBuffer

The buffer the file will be read into. This buffer is allocated by the user.

XSize

Width of the image in pixels.

YSize

Height of the image in lines.

BitDepth

Depth of the pixels in bits.

Returns

BF_OK	Function was successful.
BF_ER_BMP_BIT_DEPTH	Invalid bit depth for BMP. BMP supports 8 and 24 bit pixel depths.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_BMP_OPEN_FILE	Failed opening BMP file
BF_ER_BMP_DATA_READ	Failed reading image data from BMP file.
BF_ER_TIF_OPEN_FILE	Failed opening tif file.
BF_ER_TIF_DATA_READ	Failed reading image data from tif file.
BF_ER_AVI_READ_SINGLE	Can't read an AVI file with this function. Use BFIOReadMultiple to read an AVI file.
BF_ER_FILE_FORMAT	Invalid file format. Please use BMP, TIFF or AVI

Comments This function reads a single file from disk into user allocated memory. This function can read bmp, raw or tif file formats.

This function was written to read in files that were saved to disk from BitFlow's write single and multiple functions. BitFlow can not guarantee successful reading of a file from anything other than BitFlow's write functions.

53.5 BFIOReadMultiple

Prototype BFU32 BFIOReadMultiple(char* *FileName,* PBFU32 **pBufArray,* BFU32 *XSize,* BFU32 *YSize,* BFU32 *BitDepth,* BFU32 *NumBuffers,* BFU32 *AVIStartFrame*)

Description Reads in a sequence of files into user allocated memory.

Parameters

FileName

The first file to be read into memory. The file name should include the path and file extension. Valid file extensions are .bmp, .raw, .tif, .tiff and .avi. The file name is case insensitive.

**pBufArray*

Pointer to an array of pointers to user allocated buffers, where the files will be read into.

XSize

Width of the image in pixels.

YSize

Height of the image in lines.

BitDepth

Depth of the pixels in bits.

NumBuffers

The number of buffers to be read into.

AVIStartFrame

The first frame out of an AVI file to be read into memory. This parameter will be ignore for all file formats except the AVI format.

Returns

BF_OK	Function was successful.
BF_ER_BMP_BIT_DEPTH	Invalid bit depth for BMP. BMP supports 8 and 24 bit pixel depths.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_BMP_OPEN_FILE	Failed opening BMP file
BF_ER_BMP_DATA_READ	Failed reading image data from BMP file.

BF_ER_TIF_OPEN_FILE	Failed opening tif file.
BF_ER_TIF_DATA_READ	Failed reading image data from tif file.
BF_ER_AVI_READ_SINGLE	Can't read an AVI file with this function. Use BFIOReadMultiple to read an AVI file.
BF_ER_FILE_FORMAT	Invalid file format. Please use BMP, TIFF or AVI

Comments

This function reads in a sequence of files into memory. This function supports bmp, tif, raw and avi file formats. After the first file specified by the FileName parameter, the number of the file will be incremented and then read into memory. The number of reads will continue based on the NumBuffers parameter.

This function was written to read in files that were saved to disk from BitFlow's write single and multiple functions. BitFlow can not guarantee successful reading of a file from anything other than BitFlow's write functions.

Note: As of SDK version 5.00 a 8 digit number is appended to the file name when writing multiple files. In SDK versions before 5.00 a 6 digit number was appended. For example, FileName000000.bmp has now become FileName00000000.bmp. Files that were saved with the 6 digit number will fail to be read with the SDK 5.00 multiple read. In order to have those file read in properly, the file name will need to be updated to have the 8 digit number.

53.6 BFIOReadParameters

Prototype BFU32 BFIOReadParameters(char* *FileName,* PBFU32 *XSize,* PBFU32 *YSize,* PBFU32 *BitDepth, PBFU32 NumFrames*)

Description Reads back the parameters of a bmp, avi or tif file.

Parameters

FileName

The file to read the parameters from. The path and file extension are included. Valid file extensions are .bmp, .tif, .tiff and .avi. The file name is case insensitive.

XSize

The returned width of the image in pixels

YSize

The returned height of the image in lines.

BitDepth

The returned depth of the pixels in bits.

NumFrames

Returns the number of frames in a multi-framed format (AVI). For all other formats, this value will be one.

Returns

BF_OK	Function was successful.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_RAW_READ_ PARAMS	Can't read parameters from raw file format.
BF_ER_OPEN_FILE	Error opening file to read header information.
BF_ER_DATA_READ	Error reading in header information.
BF_ER_FILE_FORMAT	A file format is being used other that BMP, tif or AVI.

Comments This function reads and returns the xsize, ysize, bit depth and number of frames for multi-framed formats, from the file's header. Since the raw file format has no header information, this function will only work for BMP, TIFF and AVI file formats.

The NumFrames parameter will always return the value of one for all file formats except for AVI. For the AVI format, the number of frames that make up the AVI will be returned.

53.7 BFIOSaveDlg

Prototype BFU32 BFIOSaveDlg(BFBOOL *SingleFrame,* char* *FileName*, BFSIZET *FileNameSize*)

Description Provides a dialog box to specify a file name to save to disk.

Parameters

SingleFrame

Determines which file formats can be saved:

TRUE - Gives options for BMP, tif and raw.
FALSE - Gives options for BMP, tif, raw and AVI.

FileName

Returns the file name for the file.

FileNameSize

The size in bytes of the *FileName* parameter buffer.

Returns

BF_OK	Function was successful.
BF_CANCEL	User cancelled the dialog box.

Comments This function provides a dialog box to prompt the user for a file name, location and file format to save image data to disk. The name of the file is returned from this file and can then be passed on the one of the write functions.

53.8 BFIOOpenDlg

Prototype BFU32 BFIOOpenDlg(char* *FileName*, BFSIZET *FileNameSize*)

Description A dialog to retrieve a file.

Parameters

FileName

Returns the file name to be opened. The file name includes the file extension and path.

FileNameSize

The size in bytes of the *FileName* parameter buffer.

Returns

BF_OK	Function was successful.
BF_CANCEL	User cancelled the dialog box.

Comments This function brings up the open dialog so the user can pick a file to open. The file name can then be passed on to one of the read functions.

53.9 BFIOErrorShow

Prototype BFU32 BFIOErrorShow(BFU32 *ErrorCode*)

Description Displays a dialog box with information about the error.

Parameters *ErrorCode*

The error code to get the description for.

Returns

BF_OK	In all cases.

Comments This function will give a description of the error that has been returned by one on the BFIO functions. This function will only work correctly with the BFIO disk functions in this chapter.

53.10 BFIOErrorGetMes

Prototype BFU32BFIOErrorGetMes(BFU32 *ErrorCode*, PBFCHAR *Message*, PBFU32 *pMessageBufSize*)

Description Returns the error text for a given BFIO error code.

Parameters

ErrorCode

The error code to get the description for.

Message

Pointer to the output string buffer. May be provided as a null pointer, to retrieve just the *pMessageBufSize*. Message is filled as completely as possible, even if shorter than the total message length

pMessageBufSize

Pointer to the message buffer size. On input, this should be the size of the Message buffer (if Message is non-null). On output, this is set to the total size of the error message, including the null terminator.

Returns

BF_OK	Success.
BF_ER_UNKNOWN_ERROR_CODE	The value passed in the parameter *ErrorCode* is not an error code returned from any BFIOxxx function.
BF_ER_BAD_BUFFER_LENGTH	Error with buffer sizes or pointers.

Comments Retrieve the error text associated with a BFIO error code. Note this function should only be used with errors returned from BFIOxxx functions.

53.11 BFIOWriteSingleEx

Prototype BFU32 BFIOWriteSingleEx(char* *FileName,* PBFU32 *pBuffer,* BFU32 *XSize,* BFU32 *YSize,* BFU32 *BitDepth,* BFU32 *Options,* PBFIOParams *Params*)

Description Writes one buffer to a file on the disk in the DNG, BMP, TIFF, or raw file format with metadata support.

Parameters

FileName

The file name to be saved to disk. The file name includes the file extension. Valid file extensions are .bmp,.raw,.tif and.tiff. The file name is case insensitive.

pBuffer

Pointer to the image data.

XSize

Width of the image in pixels.

YSize

Height of the image in lines.

BitDepth

Depth of the pixels in bits.

Options

The options for saving a buffer to disk are:

SwapRGB - Swap the RGB format to BGR.
PACK32TO24BIT - Packs 32 bit (RGBX) data into 24 bit data (RGB).
BOTTOM_UP - Saves the data to disk upside down from what is being displayed.
OVERWRITE - If the file already exists, it will be overwritten. Default behavior is to return an error and not overwrite an existing file.

Params

A pointer to a BFIOParams structure. This provide support for inserting metadata in the file. Metadata is only supported for TIFF and DNG files. It is optional for TIFF files and required for DNG files. This pointer must be pre-allocated using BFIOMakeExParams.

Returns

BF_OK	Function was successful.

BF_ER_FILE_NAME	No file name given.
BF_ER_BUF_POINTER	Invalid buffer pointer.
BF_ER_NUM_BUFFERS	The number of buffers must be greater than zero.
BF_ER_XSIZE	Invalid XSize. The XSize must be greater than zero.
BF_ER_YSIZE	Invalid YSize. The YSize must be greater than zero.
BF_ER_BITDEPTH_ UNKNOWN	Unknown bit depth. The bit depth must be 8, 10, 12, 14, 16, 24 or 32.
BF_ER_BITDEPTH_ SWAPRGB	The bit depth must be greater than or equal to 24 to use the SWAPRGB option.
BF_ER_PACK24_BITDETPH	Must start with 32 bit data to use the PACK32-TO24BIT option.
BF_ER_BMP_BIT_DEPTH	Invalid bit depth for BMP. BMP supports 8, 24 and 32 bit pixel depths.
BF_ER_BMP_OPEN_FILE	Failed opening BMP file.
BF_ER_BMP_FILE_HEADER	Failed writing BMP header to file.
BF_ER_BMP_DATA_WRITE	Failed writing image data to BMP file.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_TIF_BIT_DEPTH	Invalid tif bit depth. The bit depth must be 8, 10, 12, 14, 16, 24 or 32.
BF_ER_TIF_OPEN_FILE	Failed opening tif file.
BF_ER_TIF_FILE_HEADER	Failed writing tif header to file.
BF_ER_TIF_DATA_WRITE	Failed writing image data to tif file.
BF_ER_RAW_OPEN_FILE	Failed opening raw file.
BF_ER_RAW_DATA_WRITE	Failed writing image data to raw file.
BF_ER_AVI_BIT_DEPTH	Invalid bit depth for AVI. AVI supports 8, 24 and 32 bit pixel depths.
BF_ER_AVI_OPEN_FILE	Failed opening AVI file.
BF_ER_AVI_DATA_WRITE	Failed writing image data to the AVI stream.
BF_ER_CREATE_STREAM	Error creating AVI stream.
BF_ER_SAVE_OPTIONS	Error with dialog box save options for AVI.
BF_ER_COMPRESS_STREAM	Error with compressing the stream.
BF_ER_AVI_HEADER	Error putting AVI header in the stream.
BF_ER_FILE_FORMAT	Invalid file format. Please use BMP, TIFF, or raw.

BF_ER_RAW_OPEN_TEXT-FILE	Failed to open the text file to write raw image file information.
BF_ER_RAW_TEXT_WRITE	Failed to write the data to the raw text file.

Comments

This function will write image data to disk in either the BMP, TIFF, raw or DNG file formats. This function has expanded capabilities over the BFIOWriteSingle in that it adds support for the DNG file format as well as supporting metadata for both TIFF and DNG formats.

If the raw file format is used, a text file with the same name as the file saved will also be generated that will contain the xsize, ysize and bit depth for the saved image. This information will become useful when trying to open a raw file type.

If no path is used in the file name parameter, the file will be saved in the same directory as the application is being run from.

53.12 BFIOReadSingleEx

Prototype BFU32 BFIOReadSingleEx(char* *FileName,* PBFU32 *pBuffer,* BFU32 *XSize,* BFU32 *YSize,* BFU32 *BitDepth,* PBFIOParams *Params*)

Description Reads in a single frame from a file on the disk into memory allocated by the user. Supports BMP, TIFF, Raw and DNG file formats.

Parameters

FileName

The file to be read into memory. The file name includes the file path and extension. Valid file extensions are .bmp, .raw, .tif and .tiff. The file name is case insensitive.

pBuffer

The buffer the file will be read into. This buffer is allocated by the user.

XSize

Width of the image in pixels.

YSize

Height of the image in lines.

BitDepth

Depth of the pixels in bits.

Params

A pointer to an allocated BFIOParams structure. This pointer must be pre-allocated using BFIOMakeExParams. This provide support for reading metadata from the file. Metadata is only supported for TIFF and DNG files.

Returns

BF_OK	Function was successful.
BF_ER_BMP_BIT_DEPTH	Invalid bit depth for BMP. BMP supports 8 and 24 bit pixel depths.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_BMP_OPEN_FILE	Failed opening BMP file
BF_ER_BMP_DATA_READ	Failed reading image data from BMP file.
BF_ER_TIF_OPEN_FILE	Failed opening tif file.
BF_ER_TIF_DATA_READ	Failed reading image data from tif file.

BF_ER_AVI_READ_SINGLE	Can't read an AVI file with this function. Use BFIOReadMultiple to read an AVI file.
BF_ER_FILE_FORMAT	Invalid file format. Please use BMP, TIFF or AVI

Comments

This function reads a single file from disk into user allocated memory. This function can read BMP, TIFF, raw or DNG file formats. This function has expanded capabilities over the BFIOReadSingle in that it adds support for the DNG file format as well as supporting metadata for both TIFF and DNG formats.

This function was written to read in files that were saved to disk from BitFlow's write single and multiple functions. BitFlow can not guarantee successful reading of a file from anything other than BitFlow's write functions.

53.13 BFIOReadParametersEx

Prototype BFU32 BFIOReadParametersEx(char* *FileName,* PBFU32 *XSize,* PBFU32 *YSize,* PBFU32 *BitDepth, PBFU32 NumFrames,* PBFIOParams *Params*)

Description Reads back the parameters of a BMP, AVI, TIFF or DNG file. The Ex version of this function also reads TIFF and DNG metadata.

Parameters

FileName

The file to read the parameters from. The path and file extension are included. Valid file extensions are .bmp, .tif, .tiff and .avi. The file name is case insensitive.

XSize

The returned width of the image in pixels

YSize

The returned height of the image in lines.

BitDepth

The returned depth of the pixels in bits.

NumFrames

Returns the number of frames in a multi-framed format (AVI). For all other formats, this value will be one.

Params

A pointer to an allocated BFIOParams structure. This pointer must be pre-allocated using BFIOMakeExParams. This provide support for reading metadata from the file. Metadata is only supported for TIFF and DNG files.

Returns

BF_OK	Function was successful.
BF_ER_LOW_MEM	Failed allocating memory. Free resources and try again.
BF_ER_RAW_READ_ PARAMS	Can't read parameters from raw file format.
BF_ER_OPEN_FILE	Error opening file to read header information.
BF_ER_DATA_READ	Error reading in header information.
BF_ER_FILE_FORMAT	A file format is being used other that BMP, tif or AVI.

Comments

This function reads and returns the xsize, ysize, bit depth and number of frames for multi-framed formats, from the file's header. Since the raw file format has no header information, this function will only work for BMP, TIFF, AVI and DNG file formats. This function has expanded capabilities over the BFIOReadParameters in that it adds support for the DNG file format as well as supporting metadata for both TIFF and DNG formats.

The NumFrames parameter will always return the value of one for all file formats except for AVI. For the AVI format, the number of frames that make up the AVI will be returned.

53.14 BFIOMakeExParams

Prototype PBFIOParams BFIOMakeExParams()

Description Allocates a BFIOParams .

Returns A pointer to a fully allocated BFIOParams structure.

Comments The parameters structure is returned in a cleared/initial state and read to be used by BFIOWriteSingleEx, BFIOReadSingleEx and BFIOReadParametersEx.

If calling BFIOWriteSingleEx repeatedly, the BFIOParams structure must be re-initialized before each call. The function BFIOClearExParams is provided for this purposed BFIOClearExParams simply returns the structure to its initial that state. BFIOClearExParams allows reuse of a handle without having to manually reset every parameter. This is efficient than call BFIOMakeExParams and BFIOFreeExParams repeatedly.

When the structure is no longer needed, free the structure using the function BFIOFreeExParams.

53.15 BFIOFreeExParams

Prototype BFU32 BFIOFreeExParams(PBFIOParams *Params*)

Description Frees the memory associated with a BFIOParams .

Parameters *Params*

A pointer to an allocated BFIOParams structure. This pointer must be pre-allocated using BFIOMakeExParams.

Returns

BF_OK	Function was successful.
BF_ER_INVALID_PARAMS_STRUCT	*Params* is a null pointer.

Comments This function frees all the memory associated with a BFIOParams structure.

53.16 BFIOClearExParams

Prototype BFU32 BFIOClearExParams(PBFIOParams *Params*)

Description Set a BFIOParams structure to default values.

Parameters *Params*

A pointer to an allocated BFIOParams structure. This pointer must be pre-allocated using BFIOMakeExParams.

Returns

BF_OK	Function was successful.
BF_ER_INVALID_PARAMS_STRUCT	*Params* is a null pointer.

Comments This function fills out a BFIOParams structure with default values. After calling this function the structure can be further customized with values pertinent from your application.

If calling BFIOWriteSingleEx repeatedly, the BFIOParams structure must be re-initialized before each call. The function BFIOClearExParams is provided for this purposed BFIOClearExParams simply returns the structure to its initial that state. BFIOClearExParams allows reuse of a handle without having to manually reset every parameter. This is efficient than call BFIOMakeExParams and BFIOFreeExParams repeatedly.

Use BFIOMakeExParams to allocate and initialize a BFIOParams structure.

When the structure is no longer needed, free the structure using the function BFIOFreeExParams.

BitFlow Types

Chapter 54

54.1 List of Defined Types

The Table 54-1 identifies the BitFlow defined types.

Table 54-1 BitFlow Types

Standard C	Microsoft Type	BitFlow Type	BitFlow Pointer
void	void	BFVOID	*PBFVOID
char	char	BFCHAR	*PBFCHAR
char	_int8	BFS8	*PBFS8
unsigned char	unsigned_int8	BFU8	*PBFU8
short	_int16	BFS16	*PBFS16
unsigned short	unsigned_int16	BFU16	*PBFU16
long	int_32	BFS32	*PBFS32
unsigned long	unsigned_int32	BFU32	*PBFU32
int	int	BFBOOL	*PBFBOOL
	_int64	BFU64	*PBFU64
	unsigned _int64	BFS64	*PBFS64
size_t	size_t	BFSIZET	*PBFSIZET
See note below		BFSPTR	*PBFSPTR
See note below		BFUPTR	*PBFUPTR

Note: BFSPTR and BFUPTR are the size of a signed/unsigned (respectively) pointer in the given operating system. This means for 32-bit operating systems, these will be 4 byte pointers and for 64-bit operating systems these will be 8 byte pointers.

Index

B

C

D

E

R

W