BitFlow SDK
Example Programs

Table of Contents

Console Applications

BiSeqSimple

Allocates and acquires frames to ten buffers using the Buffer Interface (BufIn) API. Gives the option to save these frames to the working directory as TIFFs.

A BitFlow board must first be selected by the user, assuming more than one is present. Ten buffers are created and the chosen board is configured for a sequence capture. After running the sequence acquisition, the user is given the option to save the data in the working directory with names of the form "BiSeqSimple*.tif," where the asterisk will be a number indicating which buffer element has been stored to the file.

API functions used: `BiBrdClose, BiBrdOpen, BiBufferAllocCam, BiBufferFree, BiCaptureStatusGet, BiDiskBufWrite, BiErrorShow, BiInternalTimeoutSet, BiSeqAqSetup, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqParameters, BiSeqWaitDone, DoBrdOpenDialog.`

BiSeqSimple-2Brds

Functionally similar to, albeit slightly simplified from, BiSeqSimple. Captures from two Virtual Frame Grabbers (VFGs) simultaneously, instead of just one, and has a separate buffer for the second VFG.

Nearly identical to BiSeqSimple, but with a second set of buffers and acquisition functions for a second BitFlow board. The number of buffers are only half the the number used in BiSeqSimple, five buffer for each VFG. If saving the captured sequences to disk is desired, they will be saved in the current working directory with names of the form "BiSeqSimple-2Brds1*.tif" and "BiSeqSimple-2Brds2*.tif," where the asterisks will be replaced by numbers indicating which buffer element has been stored in the file.

API functions used: `BiBrdClose, BiBrdOpen, BiBufferAllocCam, BiBufferFree, BiCaptureStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqWaitDone, DoBrdOpenDialog.`

BiSeqSimple-NBrds

Opens up to sixteen Virtual Frame Grabbers (VFGs) simultaneously and captures a sequence of images from each, giving an option to save all of the buffers' contents to the current working directory upon completion of all acquisitions. This is similar in concept to BiSeqSimple and BiSeqSimple-2Brds, but structured as to be more readily scalable.

As opposed to most of the example programs, no facility is provided to select a specific BitFlow board. Instead, the user is prompted to select how many boards to open, and that number of boards are opened sequentially, starting from board zero. An equal number of buffers are created for each board, with the number being specified by the user. All of the boards are opened before any acquisition begins, and acquisition is performed continuously and asynchronously until all of the acquisitions are complete or an error occurs. The user is given the option to save the buffers' contents in the current working directory with names of the form "BiSeqSimple-BrdsN*.tif," where N is represents the corresponding board number and the asterisk indicates which buffer element has been saved to the file.

API functions used: `BiBrdClose, BiBrdOpen, BiBufferAllocAlignedCam, BiBufferFree, BiCaptureStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqWaitDone.`

BiSeqSimpleDisp

Captures a sequence of ten frames, much like BiSeqSimple, with the additional ability to review the images stored in the buffer visual in a display window. The buffers are written to disk without asking the user.

The user must first select which BitFlow board to use, but only if more than one is present. The relevant board is opened and configured for sequential acquisition and 10 buffers are allocated. Frames are captured until all buffers are full, at which point an acquisition review window is created and the user is given the option to loop through all of the acquired images at once (L), exit the program (X), or step through each of the images (any other key). After each image has been reviewed (or review mode has been exited), the buffer contents are flushed to the current working directory as TIFF files with names of the form "BiSeqSimpleDisp*.tif," where the asterisk will be a number indicating which buffer element has been stored in that file.

Pixel depths up to 48-bits are supported for capture and display.

API functions used: `BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocCam, BiBufferArrayGet, BiBufferFree, BiCaptureStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqParameters, BiSeqWaitDone, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfGetLut, DispSurfOffset, DoBrdOpenDialog.`

BiSeqSimpleDisp2x

Captures a sequence of frames from two boards simultaneously into buffers sharing the same memory, interleaving each corresponding frame from the two by enabling only even lines on one of the frame grabbers, and only odd lines on the other. This main purpose of this example is to show how to acquire from cameras using the 10-tap "2x" firmware, such as the Basler A406k.

The user must specify only the first of the two BitFlow boards to be used, with the second being always the next sequential board in the standard BitFlow board selection dialog. Acquisition begins immediately, with the `BiBufferAssign` function used to provide both buffers access to the same memory, which is allocated manually with `malloc`. The dimensions of the buffers correspond to the first, user selected board, and if the second board has larger dimensions, some data will be lost. After acquisition, the user is given the option to loop or step through the sequence of captured frames once in a separate display window.

If the `DO_SAVE` flag is defined for the C preprocessor, the sequence of data captured into the buffers will be saved to disk before the program exits.

API functions used: `BFRegRMW, BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAssign, BiBufferUnassign, BiCaptureStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqCleanUp, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqParameters, BiSeqWaitDone, DispSurfClose, DispSurfCreate, DispSurfFormatBlit, DispSurfGetBitmap, DispSurfOffset, DoBrdOpenDialog.`

BiSimplePlusPlus

Functionally equivalent to BiSeqSimpleDisp, but using the SequenceInterface class to perform acquisition, rather than the Buffer Interface (BufIn) API, producing much simpler, more concise code.

With the exception of slightly different console messages, user-visible functionality is essentially identical to that of BiSeqSimpleDisp. By using the SequenceInterface C++ class, the code size is notably reduced, with fewer setup functions required and no need for manual cleanup functions, along with simpler exception-based error handling.

API functions used: `DispSurfClose, DispSurfCreate, DispSurfFormatBlit, DispSurfGetBitmap, DispSurfOffset, DoBrdOpenDialog.`

CiChangeSize

Captures and displays frames continuously, periodically toggling the board and display window between the original dimensions and halved height and width. A straightforward example of the Camera Interface (Ci) API.

First prompts the user to select a BitFlow board, but only if more than one is present. A display window is opened and acquisition is configured. Frames are captured continuously, while every one-hundred frames, the display window and frame grabber are reconfigured to display at, alternatingly, half the original dimensions (one-quarter frame) or the original dimensions. Entering any key into the console window exits the program.

API functions used: `BFErrorShow, CiAqCleanUp, CiAqCommand, CiAqFrameSize, CiAqSetup, CiBrdClose, CiBrdInquire, CiBrdOpen, CiSysBrdFind, DispSurfChangeSize, DispSurfCreate, DispSurfGetBitmap, DispSurfGetLut, DispSurfIsOpen, DoBrdOpenDialog.`

CiCLSimple

A basic example demonstrating how to use the Camera Link serial interface API to control a Camera Link camera, in particular modifying the exposure rate as the camera runs. This example illustrates controlling a Dalsa camera, however the code is easily modified to work with any camera as long as the protocol is known.

A BitFlow Camera Link compatible board must first be selected by the user, if more than one is present. The Camera Link serial interface is initialized, as is acquisition for the indicated board, and a window is created to display any captured frames. The main loop begins running and continues until an error occurs or the user enters a key into the console window. Main loop tasks include acquiring and displaying a frame from the frame grabber, and calling a function, which attempts to cycle the camera through several different exposure rates using the CL API.

API functions used: `BFErrorShow, CiAqCleanUp, CiAqCommand, CiAqSetup, CiBrdClose, CiBrdInquire, CiBrdOpen, CiSysBrdFind, clGetErrorText, clGetNumPorts, clSerialClose, clSerialInit, clSerialRead, clSerialWrite, DispSurfBlit, DispSurfCreate, DispSurfGetBitmap, DispSurfIsOpen, DoBrdOpenDialog.`

CircClassExample

A relatively simple example of capturing data to a circular buffer from a single frame grabber board, with a multi-threaded architecture demonstrating both error handling and asynchronous user input, image acquisition and image display. Equivalent to the Circular example, using the CircularInterface class instead of the Buffer Interface (BufIn) API.

The user is first prompted to specify the BitFlow board to use (if more than one is present). At any time after this, the user may start/go (G), stop (S), pause (P), continue (C) or abort (A) data acquisition by entering the associated key into the console window, given that the state transition is valid (one may only continue after pausing, start after stopping or aborting). The program exits when the user enters the letter X into the console window or whenever an error is produced. Upon exiting, the user is informed of the total number of frames captured and missed, and prompted with messages for any errors produced. Image capture and display, error handling and user interaction are handled in separate threads.

Various pixel depths are supported, up to 48 bits.

API functions used: `DispSurfClose, DispSurfCreate, DispSurfFormatBlit, DispSurfIsOpen, DoBrdOpenDialog.`

CircHoldSimple

Uses the CircularInterface class to demonstrate holding of an image buffer to facilitate multi-threaded programming. The code is in C++ and is straightforward in its use of the BitFlow API, while presenting a somewhat more sophisticated multi-threaded system, implemented with semaphores and mutexes.

User interaction is fairly simple. If more than one BitFlow board is present, the user must first select which board to use from. The number of buffers to create must also be specified. Some, but not all, of the frames captured are copied into the circular buffer, and every frame in the buffer is displayed in a window. The program uses a console window, image display window and the standard BitFlow board selection dialog.

Underlying operation is somewhat more complicated than the user interaction suggests. Two threads are spawned from the main thread, one which copies captured frames into the circular buffer when they become available, but only under a pseudo-random condition, the other of which pops off and displays the most recently added image in the buffer. A mutex is used to stop the image copying frame from writing to the buffer while it is being popped and, vice versa, to stop the buffer being popped while an image is being captured. A semaphore is used to indicate when an image has been added to the buffer.

API functions used: `DispSurfClose, DispSurfCreate, DispSurfFormatBlit, DoBrdOpenDialog.`

Circular

A relatively simple example of capturing data to a circular buffer from a single frame grabber board, with a multi-threaded architecture demonstrating both error handling and asynchronous user input, image acquisition and image display.

The user is first prompted to specify the BitFlow board to use (if more than one is present) and the number of buffers to create (must be at least two). At any time after this, the user may start/go (G), stop (S), pause (P), continue (C) or abort (A) data acquisition by entering the associated key into the console window, given that the state transition is valid (one may only continue after pausing, start after stopping or aborting). The program exits when the user enters the letter X into the console window or whenever an error is produced. Upon exiting, the user is informed of the total number of frames captured and missed, and prompted with messages for any errors produced. Image capture and display, error handling and user interaction are handled in separate threads.

Various pixel depths are supported, up to 48 bits.

API functions used: `BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocCam, BiBufferFree, BiCaptureStatusGet, BiCircAqSetup, BiCircCleanUp, BiCirControl, BiCirErrorWait, BiCirStatusSet, BiCirWaitDoneFrame, BiControlStatusGet, BiErrorShow, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfOffset, DoBrdOpenDialog.`

Circular-2Brds

A relatively simple example derived from the Circular example, modified to demonstrate capturing from and controlling two BitFlow boards simultaneously using separate circular buffers. A multi-threaded architecture allows asynchronous user input, acquisition monitoring and error checking.

Functionally, very similar to the Circular example, with the addition of a separate buffer and acquisition monitoring and error checking threads for the second card. User controls are the same, excepting that lowercase letters control one of the boards, and uppercase the other. Multiple frame grabbers, or a single BitFlow board with multiple Virtual Frame Grabbers are necessary to run this example.

API functions used: `BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocCam, BiBufferFree, BiCaptureStatusGet, BiCircAqSetup, BiCircCleanUp, BiCirControl, BiCirErrorCheck, BiCirErrorWait, BiCirStatusSet, BiCirWaitDoneFrame, BiControlStatusGet, BiErrorShow, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfOffset, DoBrdOpenDialog.`

CircularChangeCamFile

A relatively simple example derived from the Circular example, modified to demonstrate on-the-fly changing of BitFlow camera files, allowing for great versatility of operation. A *multi-threaded architecture allows asynchronous user input, acquisition monitoring and error checking.*

Functionally very similar to the Circular example. The only user visible change is the addition another acquisition control providing the option to alternate between the original camera file and a synthetic camera file, which is done by entering Q in the console window. However, the underlying code is notably divergent, with the BoardCleanup and ChangeCamFiles functions substituting and expanding upon functionality included in the main function of Circular. Error checking, data acquisition and image display are still implemented using an asynchronous, multi-threaded architecture.

API functions used: `BFErrorShow, BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocCam, BiBufferFree, BiCaptureStatusGet, BiCircAqSetup, BiCircCleanUp, BiCircCleanUp, BiCirControl, BiCirErrorCheck, BiCirErrorWait, BiCirStatusSet, BiCirWaitDoneFrame, BiControlStatusGet, BiControlStatusGet, BiErrorShow, CiBrdCamGetFileName, CiBrdCamSel, CiBrdCamSetCur, CiCamClose, CiCamOpen, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfOffset,DoBrdOpenDialog.`

CircularSimple

One of the simplest examples, creating a circular buffer, which is filled indefinitely with image data captured from a single frame grabber board. This is a great example to start with because of its simplicity.

Before operation begins, the user must specify the BitFlow board to use (if more than one is present) and the number of buffers to create (must be at least two). After each frame is captured to the buffer, its first byte of data is displayed as a hex number, along with the decimal number of frames missed since the previous capture. Once data capture has commenced, the program will not exit until the user presses a keyboard key in the console window.

API functions used: `BiBrdClose, BiBrdOpen, BiBufferAllocCam, BiBufferFree, BiBufferQueueSize, BiCircAqSetup, BiCircCleanUp, BiCirControl, BiCirStatusSet, BiCirWaitDoneFrame, BiErrorShow,DoBrdOpenDialog.`

CiSimple

Opens, configures and displays data from a single BitFlow camera board using the lower-level Camera Interface (Ci) API. This is a fairly simple example, but significantly more involved than a similar Buffer Interface (BufIn) API based application, such as the functionally similar CircularSimple.

Three command line arguments are available, with `-Switch` and `-Con` (specifying the board switch and board connector, respectively) allowing a BitFlow Virtual Frame Grabber (VFG) to be selected, and `-Cam` allowing a specific camera file to be used. If a VFG is not specified in the arguments, the standard BitFlow board selection dialog is presented, and the user must choose one of the boards listed, assuming more than one is present. If compiled with `USE_DISPLAY` defined, a separate thread is spawned to create and update a display window, which the main thread will then display captured frames to. Otherwise, the first two bytes of data captured are printed to the console, along with the number of buffer overflows that have occurred. Bit depths of 32 bits or less are supported.

API functions used: `BFErrorShow, BFQTabModeRequest, CiAqCleanUp, CiAqCommand, CiAqOverflowCheck, CiAqSetup, CiBrdClose, CiBrdInquire, CiBrdOpen, CiBrdOpenCam, CiConInt, CiConSwTrig, CiConVTrigModeGet, CiConVTrigModeSet, CiSysBoardFindSWConnector, CiSysBrdFind, DispSurfClose, DispSurfCreate, DispSurfFormatBlit, DispSurfGetBitmap, DispSurfIsOpen, DoBrdOpenDialog.`

CiSimpleNTG

A fairly simple example of the lower-level Camera Interface (Ci) API, which opens a single BitFlow board for capture and allows the user to adjust the New Timing Generator (NTG) on the fly.

The user must first select which BitFlow board to use (if more than one is present) and then provide values for several of the NTG parameters: The exposure time, line/frame period, trigger mode, exposure signal mode and output signal. Exact arbitrary values for some of these parameters cannot be used, so the rounded values actually set are printed in confirmation. The encoder divider scale factor, DC mode and phase mode are similarly be set and confirmed. A separate thread and associated window are spawned to display frames acquired from the frame grabber as they are captured. During this time, the user may enter into the console window E, to modify the encoder divider scale factor; N, to modify the exposure time and line/frame period; or X, to exit the program.

API functions used: `BFErrorShow, CiAqCleanUp, CiAqCommand, CiAqSetup, CiBrdClose, CiBrdInquire, CiBrdOpen, CiConExposureControlGet, CiConExposureControlSet, CiEncoderDividerGet, CiEncoderDividerSet, CiSignalCancel, CiSignalCreate, CiSignalFree, CiSignalWait, CiSysBrdFind, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfIsOpen, DoBrdOpenDialog.`

DiskIOSimple

A simple example demonstrating several useful operations in their most basic form: A single image frame is captured, displayed, and saved to the disk.

The user is first prompted to select which BitFlow board to use (if more than one is present). A single frame is immediately captured from the corresponding camera and then displayed in a window. Entering a key in the console window will produce a standard save-file dialog, in which the user may save the captured image frame to any locally available storage device using either the Bitmap, TIFF or Raw file format. Selecting "Cancel" in the save-file dialog will exit the program immediately, whereas "Enter" will save the file immediately, but require another key-press in the console window before exiting the program.

Only 8-bit image capture is supported.

API functions used: `BFErrorShow, BFIOSaveDlg, BFIOWriteSingle, CiAqCleanUp, CiAqCleanUp, CiAqCommand, CiAqSetup, CiBrdClose, CiBrdInquire, CiBrdOpen,` `CiSysBrdFind, DispSurfBlit, DispSurfCreate, DispSurfGetBitmap, DispSurfIsOpen, DoBrdOpenDialog.`

DiskRead

Reads from the active directory of the disk a sequence of ten TIFF images with file names of the form "Sequence000000.tif." Images are displayed consecutively in a separate window.

The user is first prompted to select which BitFlow board to use (if more than one is present), although this program does not actually access the board, merely requiring a board handle to use the Buffer Interface (BufIn) API functions. `BiDiskParamRead` is used to determine various parameters of the images to be read, and those parameters are used to generate an appropriate buffer and display window. Once the images are loaded into memory, the user has the option to cycle through displaying each once in the display window by entering any key into the console window other than L, which will display images continuously until all have been displayed, and X, which will exit the program.

Images with pixels depths of 48 bits and lower are supported. If 16-bit or 48-bit images are to be loaded, the user will be prompted to enter an exact bit-depth for use while displaying.

An appropriate image sequence is included with the example files in the same directory as the DiskRead example. If desired, the Sequence example can be used to generate an appropriate image sequence and save it to disk, although the default file names it uses are not the same as those used here.

API functions used: `BiBrdClose, BiBrdOpen, BiBufferAlloc, BiBufferArrayGet, BiBufferFree, BiDiskBufRead, BiDiskParamRead, BiErrorShow, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap,DoBrdOpenDialog.`

IntTime

Demonstrates usage of a BitFlow driver interrupt queue, and retrieval of an interrupt's timestamp.

This exmple primarily demonstrates usage of the BitFlow driver's interrupt queue, and in particular the `CiSignalWaitEx` method. The user will pass as arguments a list of all interrupt signals to follow, and will then be prompted for a board to open, which is opened immediately. A worker thread is spawned for each specified signal, which then wait continuously for each signal to be raised. On each signal raise, the signal name and timestamp are recorded in a mutex locked circular buffer. The main thread waits for the user to press any keyboard key, at which time the worker threads are quit, the contents of the circular buffer are printed to stdout, and the main thread finishes.

A variety of command line arguments are availble, providing various options: a specific board to open; the circular buffer length; an option to output the buffer using CSV formatting; an optionally specified output file, used instead of stdout; and a flag to print all available signals without recording any signals, then immediately quit.

API functions used: `BFFine, BFFineRate, CiBrdClose, CiBrdOpen, CiSignalFree,` `CiSignalNameGet, CiSignalWaitEx, CiSysBrdFind, DoBrdOpenDialog`

MemAssign

A somewhat involved example demonstrating how to create a buffer using standard C memory allocation, rather than the BitFlow API.

Functionally equivalent to the Sequence example, but with buffers created using `malloc` instead of `BiBufferAllocCam`. Additionally, the multi-threaded error checking used in Sequence is absent here.

API functions used: `BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAssign, BiBufferUnassign, BiCaptureStatusGet, BiControlStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqParameters, BiSeqStatusGet, BiSeqWaitDone, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfGetLut, DispSurfOffset, DoBrdOpenDialog.`

Sequence

A somewhat involved example demonstrating multi-threaded error handling and user controlled sequential data acquisition, basic display of the acquired data, and buffer array file saving.

Before operation can begin, the user must select which BitFlow board to acquire from (if more than one is present) and the number of image frames to acquire. Acquisition of these frames is then controlled by entering into the console window G, to go/start; S, to stop; A, to abort; P, to pause; C, to continue; X, to exit the acquisition controls (acquisition will continue in background if running); or Z, to exit the program altogether. Invalid acquisition state transitions (continuing while stopped, starting while paused) can be entered, but will generate errors. Once acquisition is either completed or exited, a window will open to display the acquired frames, which the user can step through one at a time by entering any key but L or X into the console window (L loops through all frames in the buffer, X exits the display routine); the display routine exits after the final buffer has been displayed. Finally, the user is given the option save the acquired images, which are stored with predetermined file-name pattern in the working folder.

Acquisition monitoring and error checking are performed in separate threads, allowing for asynchronous acquisition and user control. Various pixel depths less than or equal to 48 bits are supported.

API functions used: `BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocCam, BiBufferArrayGet, BiBufferFree, BiCaptureStatusGet, BiControlStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqErrorWait, BiSeqParameters, BiSeqStatusGet, BiSeqWaitDone, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfGetLut, DispSurfOffset, DoBrdOpenDialog.`

SimpleCom

Rudimentary implementation of a Camera Link serial com terminal client using mutli-

threading to handle user command input and data reception from the camera's serial device asynchronously.

Upon execution, the user is prompted to select which BitFlow Camera Link compatible board to use (if more than one is present). A separate thread is spawned to listen for and report any data received from the Camera Link serial device, while the main process waits for the user to input a less-than 256 character string. Upon hitting enter, either the user entered string is sent to the camera over the serial com, or, if the user has entered the term "Exit," program execution is brought to a halt.

For a more feature complete Camera Link terminal client with a custom GUI, see the BFCom example.

API functions used: `clBFGetSerialRef`, `clBFSerialCancelRead`, `clBFSerialRead`, `clSerialClose`, `clSerialInit`, `clSerialWrite`, `DoBrdOpenDialog`.

GUI Applications

BayView

An application for viewing the output of a Bayer filter imager in color. Though essentially just a live feed viewer, several features relevant to Bayer filter and color handling are implemented.

Upon startup, the user must select which BitFlow board to open, if more than one is present. A main window and display window will then open, along with a display window of whatever input the previously indicated board has. Several Color options are available, including color correction with manual, auto and white balance. Various Bayer matrices are selectable, it is important to choose the matrix that exactly matches the configuration of the camera being used. If the wrong matrix is selected the colors will be completely wrong (e.g. a red object will appear green) and/or there will be grid pattern overlaying the image. Nearest-neighbor, bilinear and smooth hue interpolation methods are all available, each with its own trade-off between image quality and processing requirements.

For multi-core computers, there is an option to create a semi-arbitrary number of threads to perform the Bayer processing, with variable thread timeout. It is best to choose the number of threads to match the number of cores in your PC. All rendering configurations are changeable on the fly. While acquisition is stopped, frames can be captured and saved to file. Images can also be opened from file for viewing.

API functions used: `BFIOErrorShow, BFIOReadParameters, BFIOReadSingle, BFIOWriteSingle, BFIsR64, BFRegPeek, BFRegRMW, BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocAlignedCam, BiBufferArrayGet, BiBufferFree, BiCircAqSetup, BiCircCleanUp, BiCirControl, BiCirStatusSet, BiCirWaitDoneFrame, BiErrorShow, CiConSwTrig, CiConSwTrigStat, CiConVTrigModeGet, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfDisableClose, DispSurfGetBitmap, DispSurfIsOpen, DispSurfOffset, DispSurfTitle, DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

BFCom

Implements a Camera Link terminal client similar to HyperTerminal with several useful and convenient features.

Whereas the SimpleCom example is essentially just a wrapper for DOS Prompt, BFCom implements a complete Camera Link terminal in a custom GUI front-end with various configurations and features. These include adjustable Com Properties with configurable serial port settings; ASCII, decimal and hex Com Modes; a variable Transmit Delay; a command history accessible via the up and down arrows; and the ability to send text files. This is itself a more useful application, but SimpleCom is likely a better place to start for one looking to understand serial communication over Camera Link.

API functions used: `BFRegPoke, CiBrdClose, CiBrdOpen, CiSysBrdFind, clBFGetBaudRate, clBFGetSerialRef, clBFSerialRead, clBFSerialSettings, clGetErrorText, clGetPortInfo, clSerialClose, clSerialInit, clSerialWrite,` `DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

BiFlow

An application using the Buffer Interface (BufIn) API to capture and display a sequence of images. Features include previewing and image sequence capture, playback and saving.

The user is first prompted to specify the quantity of system memory to be allocated for acquisition buffers, with the possibility of designating a large majority of free system memory (though this is not necessarily recommended). Next, a board must be selected from the standard BitFlow selection dialog, assuming more than one is present. The main application window and a display window will then open.

Within the application window, several configurable options are available through the main menu and toolbar concerning behavior related to camera type, overflow handling and playback speed, amongst others. Without storing any frames in memory, a preview from the current camera can be enabled and disabled via the *Start Preview* and *Stop Preview* items of the *Preview* menu. Acquisition can be enabled by selecting *Acquisition > Setup* and then started by selecting *Processes > Start Acquisition* and completing the *Capture settings* dialog. Acquisition can be Stopped, Paused and Resumed while running. Once acquisition is completed, playback of the captured sequence can be performed in the display window with the various standard controls in the toolbar and *Playback* menu. Acquisition mode can be exited be selecting *Acquisition > Clean up*.

API functions used: `BFInterruptDisableStamps, BFInterruptEnableStamps, BFInterruptGetStamps, BiCaptureStatusGet, BiDiskBufWrite, BiErrorShow, BiSeqAqSetup, BiSeqBufferStatus, BiSeqBufferStatusClear, BiSeqCleanUp, BiSeqControl, BiSeqErrorCheck, BiSeqParameters, BiSeqWaitDone, BiSeqWaitDoneFrame, BiTrigModeGet, CiAqCleanUp, CiAqCommand, CiAqSetup, CiBrdInquire, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfDisableClose, DispSurfFormatBlit, DispSurfGetBitmap, DispSurfIsOpen, DispSurfOffset, WaitDialogClose, WaitDialogOpen.`

BiFlowUpdate

Very similar functionally to the BiFlow example, to which it is indeed something of an update. To the user, the primary difference is a cleaner, more consolidated configuration system. The new implementation forgoes the Buffer Interface (BufIn) API in favor of the simpler to use SequenceInterface C++ class.

Unlike in the BiFlow example, where the user must first specify the quantity of system memory to reserved for buffers, BiFlowUpdate requires first that the BitFlow card to be used be selected (if more than one is present), then that the number of buffers to have memory reserved for be specified (the maximum number of buffers possible is listed, although this estimate is theoretical and may be unrealistic). Operation is nearly identical to that of the BiFlow example, excepting for configuration options which are now grouped together in the Setup Options window, accessed by selecting *Options > Setup Options...* Acquisition, playback and saving of image sequences is unchanged.

API functions used: `BFInterruptGetCounters, BFInterruptGetStamps, BFQTabModeRequest, CiAqCleanUp, CiAqCommand, CiAqSetup, CiConExTrigConnect, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfDisableClose, DispSurfFormatBlit, DispSurfGetBitmap, DispSurfIsOpen, DispSurfOffset, DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

BiProcess

An application using the Buffer Interface (BufIn) API to capture images to a circular buffer and calculate and display a histogram of the most recent frame, all on the fly. Features include a variable sampling rate, changeable overwrite method and printing of the histogram. The primary purpose of this application is to provide a simple frame work for the user to plug in their own processing algorithm.

Upon first opening, the user must specify a BitFlow board to acquire from, assuming more than one is present, and the number of buffers to use during circular acquisition. In the main window, the user can change the overwrite method and sample rate in the program menu, and start/stop and print the histogram generation through either the menu or the toolbar. While acquisition is running, the histogram is displayed in the main program window, along with various statistics of acquisition.

The histogram itself is generated in a separate thread, which runs constantly, regardless of acquisition status, with `BiCirWaitDoneFrame` used to effectively pause the thread when acquisition is stopped, continuing automatically when acquisition begins. The algorithm to generate the histogram can handle pixel depths of 8 or 16 bits. Drawing is performed during standard system repaints, minimizing unnecessary drawing.

API functions used: `BFIsR64, BFQTabModeRequest, BFRegPoke, BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocAlignedCam, BiBufferArrayGet, BiBufferFree, BiCircAqSetup, BiCircCleanUp, BiCirControl, BiCirErrorCheck, BiCirStatusSet, BiCirWaitDoneFrame,` BiErrorShow, `DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

Circ

An MDI GUI application demonstrating acquisition to a circular buffer using the Buffer Interface (BufIn) API, with each element of the buffer displayed in its own corresponding window upon each frame capture into it.

At program start up, the user is prompted to select which available BitFlow board to use, assuming more than one is present, and how many buffers are to be allocated. The main program window presents a variety of options, some accessible through the toolbar, all accessible through the program menu. *Setup Options* include *Abort on missed frames*, *Disable error signals*, *Display on the fly* (the only enabled by default), *Save on the fly* and *Limit the number of frames saved to disk…*, which opens an appropriate dialog. In order to start acquisition, one must select *Acquisition > Setup* from the main menu, which will then enable the various acquisition controls, which are *Start*, *Stop*, *Pause*, *Resume* and *Abort*, all available in the *Capture* menu. While paused or stopped, the buffers can also be cleared. Selecting *Acquisition > Clean up* disables acquisition and allows all of the *Setup Options* to be modified again. *Overflows* may also be set to be either ignored, or to trigger a reset. Frames captured to the buffer may be saved to file either individually or in a batch, *Save all*, manner.

API functions used: `BFIOErrorShow, BFIOSaveDlg, BFIOWriteSingle, BFQTabModeRequest, BiBrdClose, BiBrdInquire, BiBrdOpen, BiBufferAllocCam, BiBufferArrayGet, BiBufferFree, BiCircAqSetup, BiCircCleanUp, BiCirControl, BiCirErrorCheck, BiCirErrorWait, BiCirStatusSet, BiCirWaitDoneFrame, BiDiskBufWrite, BiDiskBufWrite, BiErrorShow, CiBrdType, DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

CircUpdate

Superficially very similar to the Circ example, but implemented using primarily the BufferInterface class library, rather than the Buffer Interface (BufIn) API used by Circ. Additionally, the configuration options are much expanded and individual buffers can be held, while the others continue to update.

Operated nearly identically to the Circ example, with extensive configuration options available while not in acquisition mode by selecting *Options > Setup Options…* from the main menu. Left clicking on the canvas of a buffer window while in acquisition mode will toggle that buffer between a held and non-held state. While in a hold state, the buffer contents will be preserved in memory and on the screen, even as the other buffers and windows are updated with newly captured frames. A hold can be applied while actively acquiring, although not while the given buffer is being updated, which can make applying a hold tricky while running; pause or stop the acquisition for easier hold application.

By using the BufferInterface C++ class, rather than the C BufIn API, far fewer functions are required to achieve essentially identical operation. If C++ can or is to be used, the BufferInterface class is well worth considering as an alternative to the base C API.

API functions used: `BFIOSaveDlg, BFIOWriteSingle, BFQTabModeRequest, CiBrdType, DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

CiView

One of the simplest of the GUI programs, acquiring from merely one board to a display window. Acquisition controls are also present.

Upon opening, the user is first prompted to select a BitFlow board to acquire from, if more than one is present. The main window and a display window will open, with captured frames displayed as they become available. The board can be closed and reopened without exiting the application, and while it is open the user has the options to Freeze acquisition, Grab frames continuously or Snap only a single frame. Capturing will create an MDI sub-window, within the main window, of the most recently acquired frame, which can then be saved to file. If the board is closed, selecting *Preview > Set Thread Options…* opens the Thread Options window, in which acquisition reset threads can be enabled and disabled to handle overflow and loss of sync errors.

In addition to the features listed, several other incomplete features are present. Selecting

File > Print Setup… allows configuring printing options, although actual printing is not currently possible. Acquisition options for Triggering and Free Run Mode are present in the interface, but disabled. Additionally, by running CiView with the argument "`-2`", a dual board mode is enabled, in which the user must select two boards at startup, and the data captured from each is written to either the odd or even lines of the same buffer, creating an interleaved effect (operation does not change otherwise). The main purpose of this command line parameter is to show how to acquire from cameras using the 10-tap "2x" firmware, such as the Basler A406k.

API functions used: `BFErrorShow, BFIOErrorShow, BFIOWriteSingle, BFIsR64, BFQTabModeRequest, BFRegRMW, BFTick, CiAqCleanUp2Brds, CiAqCommand, CiAqCommand, CiAqSetup, CiAqSetup2Brds, CiBrdCamSel, CiBrdClose, CiBrdInquire, CiBrdOpen, CiConSwTrig, CiConSwTrigStat, CiConVTrigModeGet, CiConVTrigModeSet, CiLutWrite, CiSignalCancel, CiSignalCreate, CiSignalFree, CiSignalNextWait, DispSurfBlit, DispSurfClose, DispSurfCreate, DispSurfGetBitmap, DispSurfGetLut, DispSurfOffset, DispSurfTitle, DoBrdOpenDialog, WaitDialogClose, WaitDialogOpen.`

Index of BitFlow API Functions