

MATLAB Image Acquisition Toolbox: BitFlow Adaptor Custom Properties

Table of Contents

Introduction.....	3
Device Specific Property Usage.....	3
The Source Object.....	3
Accessor and Modifier Methodology.....	3
Advanced Adaptor Initialization.....	4
BitFlow Adaptor Device Specific Properties.....	5
BFReg.....	5
BFRegVal.....	5
BuffersAllocated.....	5
BuffersToUse.....	5
CLSerialBaudRate.....	6
CLSerialBytesDropped.....	6
CLSerialBytesQueued.....	6
CLSerialLineEndAscii.....	6
CLSerialLineEndHex.....	7
CLSerialQueueLen.....	7
CLSerialReadAscii.....	7
CLSerialReadHex.....	8
CLSerialReadLen.....	8
CLSerialTimeout.....	9
CLSerialWriteAscii.....	9
CLSerialWriteHex.....	9
CXPRRegAddr.....	10
CXPRRegReadAscii.....	10
CXPRRegReadHex.....	10
CXPRRegReadLen.....	10
CXPRRegVal.....	11
CXPRRegWriteAscii.....	11
CXPRRegWriteHex.....	11
Exposure.....	11
GPIIn0 to GPIIn4.....	12
GPOut0 to GPOut6.....	12
LineFrame.....	12
TriggerMode.....	13
UseHardwareROI.....	13
Appendix A: Notes on Data-types.....	14
I. IMAQ Integers and Unsigned Integer Properties.....	14
II. Hex-array Strings.....	14
Appendix B: M-Functions.....	15
I. bf_dec2hex.m.....	15
II. bf_hex2dec.m.....	15
III. bf_bytary2hexstr.m.....	15
IV. bf_hexstr2bytary.m.....	16

Introduction

Ideal for use in research, education and any other environment desiring rapid application development, the MathWorks, Inc. MATLAB Image Acquisition Toolbox (IMAQ), paired with the Image Processing Toolbox, provides a consistent programmatic framework with which advanced image capture and processing can be achieved. In certain use cases, however, the standard feature set of the Image Acquisition Toolbox can be found inadequate. To accommodate users with more demanding needs, individual Adaptors may provide accessors and modifiers to device specific properties, exposing more of the underlying hardware's full capabilities. It is the express purpose of this document to detail and explain the device specific properties included in the BitFlow, Inc. Adaptor for the MATLAB Image Acquisition Toolbox.

Detailed information on the use of MATLAB and its various toolboxes can be found on the [MathWorks](#) website, including demonstration programs and complete API documentation. Documentation of the MATLAB IMAQ specifically is available on its [product webpage](#).

Device Specific Property Usage

The Source Object

Device specific properties are interfaced using the video Source object, a property of the video object, which can be acquired using one of two methods:

```
vidobj = videoinput('bitflow')           % Create the video object
srcobj = get(vidobj, 'Source')           % Acquire the Source object
```

or

```
vidobj = videoinput('bitflow')           % Create the video object
srcobj = vidobj.Source                   % Acquire the Source object
```

Accessor and Modifier Methodology

Three methods are provided by the Image Acquisition Toolbox to access or modify object properties: Verbosely using the `get` and `set` functions, concisely via the dot operator, and visually via the `inspect` function and corresponding GUI dialog. Properties are accessed and modified using the following command operations:

```
get(srcobj)                             % List all available properties

returned_value = get(srcobj, 'Property') % Verbose accessor

set(srcobj)                             % List each property and its valid values

set(srcobj, 'Property')                  % List valid values for the property

set(srcobj, 'Property', new_value)       % Verbose modifier

returned_value = srcobj.Property          % Concise accessor

srcobj.Property = new_value              % Concise modifier
```

Alternatively, a GUI dialog is produced by running `inspect(srcobj)`, which provides an overview of all available properties and their values as of the previous action (dialog open, property modification). This allows non read-only properties to be modified using the mouse and keyboard, for those who prefer such interaction.

Advanced Adaptor Initialization

As described in the [MATLAB IMAQ](#) documentation, the `videoinput` command can provide configuration information to an IMAQ adaptor via a format string, especially to supply a non-standard camera configuration file. Additionally, the standard IMAQ adaptor features (including triggering and region-of-interest configuration) may be initialized with non-default values by providing property-value pairs to the `videoinput` command. The format string and property pairs are provided using this syntax:

```
videoinput('bitflow', <board-index>, <format-string>, <property-name-1>, ...  
          <property-value-1>, ... , <property-name-N>, <property-value-N>)
```

The BitFlow adaptor supports setting a camera file via the format string as one would expect from the IMAQ documentation:

```
% Initialize a Karbon CXP with a synthetic image camera file.  
vidobj = videoinput('bitflow', 1, 'Generic-Synthetic-1024x1024-E1.kcxp')
```

Expanding on the format string concept, the BitFlow adaptor (from version 1.2) supports non-default initialization of the device-specific properties (the *raison d'être* of this document, and separate from the standard adaptor features `videoinput` accepts) using a list of property-value pairs specified within the format string. Each pair is begun with a semicolon, followed by the property name, the equal sign, then the property value. So to create a video object with five *BuffersToUse*, the following code may be used:

```
vidobj = videoinput('bitflow', 1, ';BuffersToUse=5')
```

This example is particularly useful, because the BitFlow adaptor attempts to allocate acquisition buffers as soon as the video object is created. By setting this value via the format string, rather than via the video source object after initialization, the extra step of deallocating then reallocating the buffers can be circumvented.

Any data coming before the first property pair in the format string is taken to be a camera file name, and no parsing or cleanup is performed upon any part of the string, beyond the simple tokenization necessary to extract each element. If no camera file is specified (as in the example above), the default camera file attached using the BitFlow SysReg utility is used, just as if no format string were specified at all. Any number of property pairs may be provided, and even repeated, which will be executed in the order provided, left-to-right. The only exception to the order rule is *BuffersToUse*, the first pair of which by necessity is set before basic board initialization.

```
% Configure the board for an Adimec Quartz camera, allocating 25 acquisition  
% buffers. Set the DeviceUserID CoaXPress Bootstrap register on the Quartz  
% to 'Hello World!', and enable the Adimec test pattern.  
frmtstr = [ 'Adimec-Quartz-CXP-2Kx1K-E1-2XDMA.kcxp' ...  
           ';BuffersToUse=25' ...  
           ';CXPRegAddr=8384' ...  
           ';CXPRegWriteHex=48656C6C6F20576F726C642100000000' ...  
           ';CXPRegAddr=33116' ...  
           ';CXPRegVal=65536' ];  
vidobj = videoinput('bitflow', 1, frmtstr)
```

BitFlow Adaptor Device Specific Properties

The following are the device specific properties provided by the BitFlow, Inc. MATLAB Image Acquisition Toolbox adaptor. See the *Device Specific Property Usage* section for an overview on how device specific properties are used.

BFReg

An enumerated accessor and modifier specifying which BitFlow board register should be read or written by the *BFRegVal* property. Each complete register and bit-field sub-register available on the frame grabber will be listed in the enumeration. Details on the registers available for your specific board are listed in the Hardware Reference Manual available at the [Downloads](#) page of the BitFlow website.

Type: Enumeration
Default value: 'con0'
Valid values: Board dependent
Read only: Never

BFRegVal

An integer accessor to and modifier of the BitFlow board register corresponding to the selected *BFReg* property. Registers are up to 32-bits long and can be read or written at any time, although not all registers are static, some registers are read-only and some registers are write only. Reference the appropriate BitFlow Hardware Reference Manual – available at the [Downloads](#) page of the BitFlow website – for detailed information regarding your board's registers.

See also: *Appendix A: Notes on Data-types > IMAQ Integers and Unsigned Integer Properties*

Type: Integer
Default value: Register dependent or N/A
Valid range: -2147483648 to 2147483647 (complete 32-bit signed integer range)
Read only: Register dependent

BuffersAllocated

This read-only integer indicates how many buffers the adaptor has actually allocated in RAM for the accessed instance of the current Virtual Frame Grabber (VFG). The number of buffers desired for allocation is specified by the *BuffersToUse* property. The value returned will be zero if another instance of the same VFG is actively acquiring, or if the buffers could not be allocated (usually resulting from memory size limitations). Otherwise, the value of Buffers Allocated should equal that of *BuffersToUse*. During run-time, reading *BuffersAllocated* will attempt to dynamically allocate the desired number of buffers before returning, and can be used to verify that the VFG is ready for acquisition.

Type: Integer
Default value: N/A
Valid range: 0 U 2 to 50,000
Read only: Always

BuffersToUse

In complement of the *BuffersAllocated* property, this integer value can be read or written at any time, though

allocation may not be performed immediately, as the buffer object is shared between each instance of a given Virtual Frame Grabber (VFG), necessitating that allocations only be performed when no other instance of the VFG is acquiring. Additionally, if the specified *BuffersToUse* would exceed available RAM if allocated, no allocation will occur. The *BuffersAllocated* property should generally be checked before attempting to begin acquisition to verify that the desired *BuffersToUse* are actually available and ready to use.

Type: Integer
Default value: 10
Valid range: 2 to 50000
Read only: Only while this instance is acquiring

CLSerialBaudRate

The baud rate of the Camera Link serial port. This must be set to match the current baud rate of the attached camera for communication to succeed at all. In practice, no current BitFlow frame grabbers support 921600 baud operations, but the option is available nevertheless.

Type: Enumeration
Default value: '9600'
Valid values: '9600', '19200', '38400', '57600', '115200', '230400', '460800', '921600'
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialBytesDropped

The number of bytes received by the Camera Link serial port that were previously queued (see *CLSerialBytesQueued*), but overwritten by incoming data before they could be read (by *CLSerialReadAscii* or *CLSerialReadHex*). This value is reset to zero, if *CLSerialQueueLen* is changed.

Type: Integer
Default value: 0
Valid range: 0 to 2147483647
Read only: Always
Availability: Camera Link frame-grabbers

CLSerialBytesQueued

The number of bytes received by the Camera Link serial port that are still in the read queue. Use *CLSerialReadAscii* or *CLSerialReadHex* to read this data before it is overwritten. Any data that is overwritten before being read is counted by *CLSerialBytesDropped*. For each byte of data read, *CLSerialBytesQueued* will be decremented. If the queue length is changed by setting *CLSerialQueueLen*, all previously queued data is cleared, and consequently, *CLSerialBytesQueued* will go to zero until more data is received.

Type: Integer
Default value: 0
Valid range: 0 to 2147483647
Read only: Always
Availability: Camera Link frame-grabbers

CLSerialLineEndAscii

One of the Camera Link serial read conditions, along with *CLSerialReadLen*. *CLSerialLineEndAscii* represents

the same internal variable as [*CLSerialLineEndHex*](#); that is, each represents the same data, but in a different format. This condition is disabled if set to an empty string. See [*CLSerialReadAscii*](#) or [*CLSerialReadHex*](#) for more information regarding the serial read conditions.

Use MATLAB's `sprintf` function to generate non-printable string characters, such as carriage return and line feed, which happen to be the default value (eg., `CLSerialLineEndAscii = sprintf('\r\n');`). Because MATLAB IMAQ handles strings as C-style ASCII arrays, it is not possible to set line-end sequence with a null-byte using [*CLSerialLineEndAscii*](#), although this can be done using [*CLSerialLineEndHex*](#).

Type: String
Default value: Carriage return-line feed (ie, C-string “\r\n”)
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialLineEndHex

One of the Camera Link serial read conditions, along with [*CLSerialReadLen*](#). [*CLSerialLineEndHex*](#) represents the same internal variable as [*CLSerialLineEndAscii*](#); that is, each represents the same data, but in a different format. This condition is disabled if set to an empty string. See [*CLSerialReadAscii*](#) or [*CLSerialReadHex*](#) for more information regarding the serial read conditions.

See also: [*Appendix A: Notes on Data-types > Hex-array Strings*](#)

Type: String
Default value: 0D0A (hex-array representation of carriage return and line feed)
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialQueueLen

The byte-length of the Camera Link serial data receiving queue. The queue is a circular buffer, which queues data whenever it is received, overwriting the oldest queued data if necessary (see [*CLSerialBytesDropped*](#)). The user removes data from the queue using either [*CLSerialReadAscii*](#) or [*CLSerialReadHex*](#). Setting a longer queue length reduces the risk that data will be overwritten before it can be read, but requires more system memory. Any time the queue length is changed, all currently queued data is cleared (consequently, [*CLSerialBytesQueued*](#) goes to zero), and [*CLSerialBytesDropped*](#) is reset to zero.

Type: Integer
Default value: 4096
Valid range: 256 to 65536
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialReadAscii

Read and dequeue data from the Camera Link serial receiving queue as an ASCII string. The length of data read depends upon the read condition met:

1. Null-terminator – If a null-terminator is found that would yield a string-length (including the null-terminator) less-than-or-equal-to [*CLSerialReadLen*](#), dequeue and return all data up-to-and-including the null-terminator.
2. Line-end sequence – If the sequence is found ([*CLSerialLineEndAscii*](#)/[*CLSerialLineEndHex*](#)), and the

length up-to-and-including the sequence is less-than-or-equal-to CLSerialReadLen, dequeue and return all data up-to-and-including the sequence. If the line-end sequence is empty, this condition is disabled, and will never be met.

3. Read-length – If the amount of available data matches or exceeds CLSerialReadLen, dequeue and return CLSerialReadLen bytes of the data. If CLSerialReadLen is zero and there is any data queued, dequeue and return whatever length of data is currently available.
4. Timeout – If CLSerialTimeout is exceeded, dequeue nothing and return an empty string.

These are evaluated in the listed order whenever more data is received by the queue, until a condition is met.

Because MATLAB IMAQ handles string properties using C-style ASCII arrays, it is not possible to return ASCII data beyond a null-terminator, and therefore an empty string may be returned if the only queued data is a null-terminator. To read raw binary data, and to circumvent the limitations of C-style ASCII strings, use CLSerialReadHex instead.

To write ASCII data to the port, see CLSerialWriteAscii.

Type: String
Read only: Always
Availability: Camera Link frame-grabbers

CLSerialReadHex

Read and dequeue data from the Camera Link serial receiving queue as a hex-array string. The length of data read depends upon the read condition met:

1. Line-end sequence – If the sequence is found (CLSerialLineEndAscii/CLSerialLineEndHex), and the length up-to-and-including the sequence is less-than-or-equal-to CLSerialReadLen, dequeue and return all data up-to-and-including the sequence. If the line-end sequence is empty, this condition is disabled, and will never be met.
2. Read-length – If the amount of available data matches or exceeds CLSerialReadLen, dequeue and return CLSerialReadLen bytes of the data. If CLSerialReadLen is zero and there is any data queued, dequeue and return whatever length of data is currently available.
3. Timeout – If CLSerialTimeout is exceeded, dequeue nothing and return an empty string.

These are evaluated in the listed order whenever more data is received by the queue, until a condition is met.

To read ASCII data from the serial queue, use CLSerialReadAscii.

To write hex-array data to the port, see CLSerialWriteHex.

See also: Appendix A: Notes on Data-types > Hex-array Strings

Type: String
Read only: Always
Availability: Camera Link frame-grabbers

CLSerialReadLen

The desired and maximum length of data to read from the Camera Link serial receive queue in a single read operation. This is one of the read conditions (see also CLSerialLineEndAscii/CLSerialLineEndHex), and limits absolutely the amount of data that can be returned in a single read. If the line-end condition is disabled (set to an empty string) and CLSerialReadHex is used to read, this is the exact length of data that will be returned,

barring a timeout.

If *CLSerialReadLen* is set to zero, whatever length of data is available at the time of a read will be returned, unless another condition is met first.

Type: Integer
Default value: 2048
Valid range: 0 to 65535
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialTimeout

The timeout period in milliseconds for calls to *CLSerialReadAscii* and *CLSerialReadHex*. If the timeout is reached, those functions will return empty strings. If the *CLSerialTimeout* is set to its maximum value, the read operations will never time-out.

Type: Integer
Default value: 100
Valid range: 0 to 65536 (milliseconds)
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialWriteAscii

Write an arbitrary-length ASCII string to the Camera Link serial port, excluding the null-terminator. Simply assign a string to this property and its contents will be written to the port as ASCII values, up-to-but-excluding the null-terminator. Use *CLSerialWriteHex* to write raw binary data to the port, which can write zero valued bytes (ie., null-terminators). If read, this property will always return an empty string.

Keep in mind that cameras controllable with ASCII commands often expect each command to be followed by a carriage return character. This can be easily achieved using MATLAB's `sprintf` function (eg., `CLSerialWriteAscii = sprintf('%s\r', my_command_string);`).

To read ASCII data from the port, see *CLSerialReadAscii*.

Type: String
Read only: Never
Availability: Camera Link frame-grabbers

CLSerialWriteHex

Write an arbitrary-length hex-array string to the Camera Link serial port. Simply assign a hex-array string to this property and the binary sequence represented by the hex-array will be written. Use *CLSerialWriteAscii* to write the ASCII values of a string sequence to the port. If read, this property will always return an empty string.

To read hex-array data from the port, see *CLSerialReadHex*.

See also: *Appendix A: Notes on Data-types > Hex-array Strings*

Type: String
Read only: Never
Availability: Camera Link frame-grabbers

CXPRegAddr

An integer value used to select the current CoaXPress camera register. The available registers and their function will vary substantially from camera-to-camera. Consult the camera vendor for details about available registers, or consult the CoaXPress documentation (available from CoaXPress.com) for details regarding the standardized Bootstrap registers.

See also: *[Appendix A: Notes on Data-types](#) > [IMAQ Integers and Unsigned Integer Properties](#)*

Type: Integer
Default value: 0
Valid range: -2147483648 to 2147483647 (complete 32-bit signed integer range)
Read only: Never
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

CXPRegReadAscii

Use this property to read *CXPRegReadLen* bytes of data from the CoaXPress camera register space, starting from address *CXPRegAddr*. Because ASCII strings are null-terminated, if the data read contains a zero valued (null-terminator) byte, the string will be cut short to that length proceeding that byte. Use *CXPRegReadHex* to read binary data, which will not be cut short by null-valued bytes.

To write ASCII data to a register sequence, see *CXPRegWriteAscii*.

Type: String
Read only: Always
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

CXPRegReadHex

Use this property to read *CXPRegReadLen* bytes of data from the CoaXPress camera register space, starting from address *CXPRegAddr*. Unlike *CXPRegReadAscii*, this will always return the full byte length of data, formatted as a hex-array string.

To write hex-array data to a register sequence, see *CXPRegWriteHex*.

See also: *[Appendix A: Notes on Data-types](#) > [Hex-array Strings](#)*

Type: String
Read only: Always
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

CXPRegReadLen

The number of bytes to be read from the CoaXPress camera register space (starting from *CXPRegAddr*) upon the next call to *CXPRegReadAscii* or *CXPRegReadHex*.

Type: Integer
Default value: 4
Valid range: 1 to 2147483647
Read only: Never
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

CXPRRegVal

Get or set the current 32-bit unsigned integer value at camera register [*CXPRRegAddr*](#).

See also: [*Appendix A: Notes on Data-types > IMAQ Integers and Unsigned Integer Properties*](#)

Type: Integer
Default value: Register dependent or N/A
Valid range: -2147483648 to 2147483647 (complete 32-bit signed integer range)
Read only: Never
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

CXPRRegWriteAscii

Write an arbitrary-length ASCII sequence to the CoaXPress camera register space, starting at address [*CXPRRegAddr*](#). Simply assign an ASCII string to this property, and it will be written byte-for-byte, up-to-and-including the null-terminator character. Note that some CoaXPress cameras have limitations upon where data sequences may be written, and may also have restrictions on the data length that can be written; consult the camera manual for more information.

Use [*CXPRRegReadAscii*](#) to read data as ASCII strings. Use [*CXPRRegWriteHex*](#) to write pure binary data.

If read, this node will always return an empty string.

Type: String
Read only: Never
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

CXPRRegWriteHex

Write an arbitrary-length hex-array sequence to the CoaXPress camera register space, starting at address [*CXPRRegAddr*](#). Simply assign a hex-array string to this property, and the equivalent binary data will be written in its entirety. Note that some CoaXPress cameras have limitations upon where data sequences may be written, and may also have restrictions on the data length that can be written; consult the camera manual for more information.

Use [*CXPRRegReadHex*](#) to read data as hex-array strings. Use [*CXPRRegWriteAscii*](#) to write ASCII string data.

If read, this node will always return an empty string.

See also: [*Appendix A: Notes on Data-types > Hex-array Strings*](#)

Type: String
Read only: Never
Availability: CoaXPress frame-grabbers, BitFlow SDK ≥ 5.70

Exposure

A double precision floating point value specifying the desired camera exposure period in milliseconds. This property has a limited range and may not work with all cameras; a camera may require special configuration before it will work with the frame grabber specified exposure period. Further, the value of the *Exposure* and *LineFrame* properties are interdependent, as the exposure period should never exceed the line frame period. For additional details, consult to the relevant BitFlow [Hardware Reference Manual](#) for your board and the manufacturer's reference materials for whatever camera is to be used.

Type: Double
Default value: Camera file dependent
Valid range: 0 to 2275.555284
Read only: Never

GPIIn0 to GPIIn4

Integer accessors to the five general purpose inputs. These include both single ended TTL and Differential (LVDS) inputs, with configurations varying from board to board. The only valid values are zero and one, with the former representing an electrical low and the latter an electrical high. These properties can be used in conjunction with the general purpose outputs, *GPOut0 to GPOut6*, to interface with external devices.

The general purpose inputs are also accessible via the board registers using the *BReg* and *BRegVal* properties. Not every BitFlow frame grabber has all five of these inputs. Consult the relevant BitFlow [Hardware Reference Manual](#) for additional details regarding your board model.

Type: Integer
Default value: Hardware input state determined
Valid range: 0 or 1
Read only: Always

GPOut0 to GPOut6

Integer accessors to and modifiers of the seven general purpose outputs. These include single ended TTL, Differential (LVDS), and Open Collector outputs, with configurations varying from board to board. The only valid values are zero and one, with the former representing an electrical low and the latter an electrical high. These properties can be used in conjunction with the general purpose inputs, *GPIIn0 to GPIIn4*, to interface with external devices.

The general purpose outputs are also accessible via the board registers using the *BReg* and *BRegVal* properties, with advanced configurations possible by setting the *GPOUT0_CON* to *GPOUT6_CON* registers to use the various available signal generating sources. Not every BitFlow frame grabber has all seven of these outputs. Consult the relevant BitFlow [Hardware Reference Manual](#) for additional details regarding your board model.

Type: Integer
Default value: 0
Valid range: 0 or 1
Read only: Never

LineFrame

A double precision floating point value specifying the desired camera line/frame period in milliseconds. The line period is equal to 1/line rate, similarly the frame period is equal to 1/frame rate. This property has a limited range and may not work with all cameras; a camera may require special configuration before it will work with the frame grabber specified line/frame period. Further, the value of the *LineFrame* and *Exposure* properties are interdependent, as the line frame period should never drop below the exposure period. For additional details, consult to the relevant BitFlow [Hardware Reference Manual](#) for your board and the manufacturer's reference materials for whatever camera is to be used.

Type: Double
Default value: Camera file dependent
Valid range: 0.000136 to 2275.555420

Read only: Never

TriggerMode

An enumeration, providing several triggering modes, which control at a low level the mode of acquisition. The default mode is 'Auto', which selects the most likely appropriate mode for the current trigger source, as configured in the IMAQ video device. The other options are described fully in the *TrigMode* portion of the `CiConVTrigModeSet` function documentation, available in the BitFlow [SDK Reference Manual](#). Certain modes are not available on every board, and will generate an error at acquisition start, if selected.

Type: Enumeration

Default value: 'Auto'

Valid values: 'Auto', 'Free Run', 'One Shot', 'Acquisition Command', 'Acquisition Command Start Stop', 'Continuous Data', 'One Shot Self Triggered', 'One Shot Start A Stop B', 'One Shot Start A Stop A', 'Snap Qualified', 'Continuous Data Qualified', 'One Shot Start A Stop A Level', 'NTG One Shot', 'Triggered Grab'

Read only: Only while acquiring

UseHardwareROI

An enumeration, with only the Boolean 'true' and 'false' options available, which indicates whether the BitFlow board's hardware Region Of Interest (ROI) support should be used (when 'true'), or if the ROI should be extracted via software alone. Hardware ROI has greater throughput and will usually require less memory for the same number of *BuffersAllocated*, but some cameras and configurations do not work well with hardware ROI, so software is available as a more reliable alternative; hardware ROI is not allowed when using Alta boards, as it is particularly problematic with analog cameras. Try setting `UseHardwareROI` to 'false' if acquisition becomes problematic when using ROI. Proper setup of a camera's configurations may remedy problems related to hardware ROI.

The acquisition system used when `UseHardwareROI` is 'true' is actually a mixed-mode hardware/software hybrid, as BitFlow boards do not allow for pixel precise ROI values. When an exact ROI is not possible, the closest-fit larger ROI area is set in hardware and the exact pixel area is extracted in software, mostly preserving the benefits of hardware ROI, while providing all the convenience of software ROI. If ROI acquisition without any software cropping is desired, consult the `CiAqROISet` function documentation in the BitFlow [SDK Reference Manual](#) to determine valid area values for your board, and simply ensure to remain within those constraints.

Type: Enumeration

Default value: 'true' ('false' for all Alta model boards)

Valid values: 'true', 'false'

Read only: Only while acquiring (always for all Alta model boards)

Appendix A: Notes on Data-types

I. IMAQ Integers and Unsigned Integer Properties

MATLAB's Image Acquisition Toolbox uses 32-bit signed integers to represent integer values, but several of the custom features exposed by the BitFlow adaptor (*BFRegVal*, *CXPRegAddr*, etc.) represent 32-bit unsigned integer values and indices. For values less than 0x80000000 (decimal value 2,147,483,648) the native MATLAB `dec2hex` and `hex2dec` functions are perfectly adequate for conversion but for greater values, they will provide inaccurate values results, or fail altogether. To alleviate this problem, the *bf_dec2hex.m* and *bf_hex2dec.m* MATLAB script functions are provided in *Appendix B: M-Functions*, which facilitate conversion between signed and unsigned integers using two's complement and sign inversion, otherwise replicating the `dec2hex` and `hex2dec` functionality.

II. Hex-array Strings

MATLAB's Image Acquisition Toolbox allows for custom properties of a variety of different types, but unfortunately, none of these correspond well to the concept of an arbitrary-length byte-array, the native format for Camera Link Serial and CoaXPress register data. This data is exposed in a near raw form using ASCII string arrays (see *CXPRegReadAscii*, *CXPRegWriteAscii*, *CLSerialReadAscii*, and *CLSerialWriteAscii*), but ASCII strings are inadequate for the case of true binary data; zero bytes would be interpreted as ASCII null-terminators. To accommodate the need for true binary data access, the BitFlow adaptor uses a hex-array string format to read and write raw data (see *CXPRegReadHex*, *CXPRegWriteHex*, *CLSerialReadHex*, and *CLSerialWriteHex*). This string format encodes individual bytes of binary data as hexadecimal pairs, so that the string "Hello!" would encode as "48656C6C6F2100", where each digit pair is the hex representation of an ASCII character, with the final "00" representing the string's null-terminator.

For the user's convenience, two MATLAB script functions are included in *Appendix B: M-Functions*, *bf_bytary2hexstr.m* and *bf_hexstr2bytary.m*, which will perform the conversion from hex-string to MATLAB byte-array, or vice versa, in a single step.

Appendix B: M-Functions

I. *bf_dec2hex.m*

```
function [ hex_val ] = bf_dec2hex( dec_val )
%BF_DEC2HEX Convert a signed 32-bit int to a hex number
% Regular dec2hex converts unsigned integers to hex strings.
% This function converts signed 32-bit integers to hex strings.

if dec_val < 0
    % If so, perform a sign inversion and
    % two's complement operation.
    dec_val = uint32 (-dec_val); % Invert and convert to an unsigned int.
    dec_val = bitcmp (dec_val) + 1; % Perform the two's complement operation.
end

hex_val = dec2hex (dec_val); % Convert the decimal value to a hex string.

end
```

II. *bf_hex2dec.m*

```
function [ dec_val ] = bf_hex2dec( hex_val )
%BF_HEX2DEC Convert a hex number to a signed 32-bit int
% Regular hex2dec will always produce an unsigned int. This function
% produces the signed 32-bit integer represented by the hex string.

dec_val = hex2dec (hex_val); % Determine the unsigned integer value.
dec_val = uint32 (dec_val); % Cast the value to an unsigned integer container.

if dec_val > 2147483647
    % If so, perform a two's complement and
    % sign inversion. 2147483647 == 0x7fffffff.
    dec_val = bitcmp (dec_val); % Determine the complement.
    dec_val = -int32 (dec_val) - 1; % Perform inversion and 'two' the
end % complement (must be done after
% int32 conversion to prevent overflow;
end % int32 == [-2147483648 2147483647] ).
```

III. *bf_bytary2hexstr.m*

```
function [ hex_array ] = bf_bytary2hexstr( byte_array )
%bf_bytary2hexstr Convert an input byte array into a hex-array string.

% Convert each byte into a 2-digit hex value.
hex_array = dec2hex(uint8(byte_array), 2);

% Transpose to put items back in the correct order.
hex_array = transpose(hex_array);

% Reshape to a single row string array.
hex_array = reshape(hex_array, 1, []);

end
```


IV. *bf_hexstr2bytary.m*

```
function [ byte_array ] = bf_hexstr2bytary( hex_array )
%bf_hexstr2bytary Convert an input hex-array string into a byte array

    % Put each pair in a column.
    byte_array = reshape(hex_array, 2, []);

    % Pad each column with whitespace.
    byte_array = [byte_array; repmat(' ', [1, size(byte_array, 2)])];

    % Convert each column from a hex string to its uint8 byte value.
    byte_array = uint8( sscanf(byte_array, '%x') );

    % Restore the array to a single row.
    byte_array = reshape(byte_array, 1, []);

end
```