SNAPPER-DIG16 Library

(for Snapper-Dig16 and Snapper-PMC-Dig16)

Programmer's Manual

DataCell Limited

v4.0.1

Disclaimer

Copyright Notice

Trademarks

“Apple”, “Macintosh” and “MacOS” are trademarks of Apple Computer Inc. “AMCC” is a registered trademark of Applied Micro Circuits Corporation. “Dallas” is a registered trademark of Dallas Semiconductor Corporation. “Dell” is a registered trademark of Dell Computer Corporation. “Flash Graphics” and “X-32VM” are trademarks of Flashtek Limited. “IBM”, “PC/AT”, “PowerPC” and “VGA” are registered trademarks of International Business Machine Corporation. “MetroWerks” and “CodeWarrior” are registered trademarks of MetroWerks Inc. “Microsoft”, “CodeView”, “MS” and “MS-DOS”, “Windows”, “Windows NT”, “Windows 95”, “Windows 98”, “Win32”, “Visual C++” are trademarks or registered trademarks of Microsoft Corporation. “National Semiconductor” is a registered trademark of National Semiconductor Corporation. “Sun”, “Ultra AX” and “Solaris” are registered trademarks of Sun Microsystems Inc. All “SPARC” trademarks are trademarks or registered trademarks of SPARC International Inc. “VxWorks” and “Tornado” are registered trademarks of Wind River Systems Inc. “Xilinx” is a registered trademark of Xilinx.
All other trademarks and registered trademarks are the property of their respective owners.

Part Information

Part Number: SNP-MAN-DIG16-LIB

Version v4.0.1 March 1999

Printed in the United Kingdom.

Contact Details

Europe & ROW	Web	www.datacell.co.uk
	Sales	info@datacell.co.uk
	Support	techsupport@datacell.co.uk
USA	Web	www.datacell.com
	Sales	info@datacell.com
	Support	techsupport@datacell.com

Head Office:
DataCell Limited.
Falcon Business Park, 40 Ivanhoe Road,
Finchampstead, Berkshire, RG40 4QQ, UK

Tel	+44 (0) 1189 324324
Fax	+44 (0) 1189 324325

Table of Contents

Introduction

This manual describes the Snapper-DIG16 function library. These functions allow the capture of video images, using a Snapper-DIG16 module and one of a number of different host hardware platforms, and are independent of the host hardware platform. The Snapper-DIG16 function library also applies to the Snapper-PMC-DIG16, and unless stated otherwise, all references to Snapper-DIG16 in this manual also apply to Snapper-PMC-DIG16.

Snapper-DIG16 is referred to by a unique handle of type *Thandle* (a 32 bit unsigned integer). It is used by all the software to identify a particular Snapper board and its associated data structures. This handle is automatically generated when a Bus Interface Board detects it has a Snapper module fitted.

Concepts

The Snapper-DIG16 hardware is different from other Snappers (such as Snapper-16 and Snapper-24) because it does not have a frame store. This is because some digital cameras generate very large images, so a frame store would need to be very large to store the image. Instead, Snapper-DIG16 uses fast baseboards such as those for PCI or SBus to transfer data in real time from the Snapper to host computer memory. Therefore the host memory is acting as the frame store. This use of host memory means that *DIG16_capture_to_image* replaces for instance both *SNP24_capture* and *SNP24_read_video_data* except for use with line scan cameras where *DIG16_read_video_data* is still used. Note that *DIG16_capture_to_image* does not support TMG strip processing.

Digital cameras generally use RS-422 synchronous control signals, and the timing of these signals varies between cameras. Therefore specific cameras are supported, and Snapper-DIG16 is set up for a camera by calling *DIG16_initialize* with the camera model as a parameter. Cameras which are not supported by *DIG16_initialize* can be set up from the application by direct calls to low level line scan control functions - see Technical Note 5 "Snapper-DIG16: Porting New Cameras" and the list of functions in the section "Function List".

CONVENTIONAL CAMERAS

"Conventional cameras" are "area scan" cameras which generate images of *m* pixels by *n* lines (e.g. 1024 by 1000). Most digital area scan cameras are progressive scan, but interlaced cameras are also supported.

LINE SCAN CAMERAS

Line scan cameras only capture one line of data at a time, but this line is generally long compared with area scan cameras, for instance a typical line scan image might be 2048 pixels by 1 line.

The application can control how many lines are read in at a time into one 'image' (function *DIG16_set_ROI*). At a minimum, one line can be captured per read - this gives minimum latency between a line being captured and it being ready for processing in host computer memory, but there will be a higher software overhead. The maximum number of lines per bank depends on how much host computer memory is available to read the lines into (e.g. 1024 lines of a 2048 pixel camera needs 2 Mbytes of memory for *TMG_Y8* format). A large number of lines gives a low processing overhead, but a larger latency. Typical values might be 32 or 64 lines per bank.

An additional issue concerning lines per image is that the current Snapper-DIG16 libraries for MS-DOS and Microsoft Windows 3.1 do not transfer data under interrupt control. Therefore under these operating systems, for as long as it takes to process the lines in one image, it is not possible to read any data out of the FIFO buffer on the Snapper-DIG16. Since the camera may be continuously filling up this FIFO it is possible that the FIFO will overflow if the setting of lines per image is too large. The function *DIG16_get_ROI_max* can be used to indicate how many lines will fit into the FIFO as a guide for optimising applications.

Most of the Snapper-DIG16 library functions can be used for both area scan and line scan modes. Where a function can only be used in one mode this is mentioned in the BUGS/ NOTES section of each function description.

See the Camera Specific Installation Notes in the Installation section of the manual for the cable pinouts for supported line scan cameras.

DATA STREAM MODE

In data stream mode the Snapper-DIG16 treats the data from the camera as a raw *data stream*. Data is acquired based on two enable signals, line enable and frame enable, and the function *DIG16_set_data_stream_ctrl* controls which of these signals should be used to enable incoming data to be stored.

This mode is provided to support cameras which are highly programmable, and for instance can output multiple regions of interest from one image, or allow image dimensions to be changed on a frame by frame basis.

Therefore in data stream mode there are no region of interest functions, instead the function *DIG16_set_parameter* sets the number of pixels to capture. It is the responsibility of the application to correctly interpret the data based on the known configuration of the camera.

Function Overview

The functions are split into five sections - Initialization, Image Capture, Configuration, Parameter Readback and Camera Specific.

INITIALIZATION FUNCTION

The initialization function configures Snapper-DIG16 to default settings. It accepts a parameter indicating which camera is connected so that Snapper-DIG16 is configured for that camera.

IMAGE CAPTURE FUNCTIONS

The image capture functions control capture of images and provide functions to test the capture status.

The *DIG16_capture_to_image* routine controls the capture of video data into an image structure in host memory. This image structure is set up by *DIG16_set_image*. Capture status is indicated by *DIG16_get_capture_status*, *DIG16_is_field1_captured* and *DIG16_is_trigger_started*. In line scan mode the function *DIG16_read_video_data* is used in conjunction with *DIG16_capture_to_image* to transfer data into an image structure in host memory.

CONFIGURATION FUNCTIONS

These functions control the configuration of the Snapper-DIG16.

The capture mode used by the next *DIG16_capture_to_image* call is controlled by *DIG16_set_capture*. This allows control of the frame/field mode and sub-sample factor. Image capture can be triggered from external hardware by using the *DIG16_set_trigger* function. Selected regions of the image can be captured by a call to *DIG16_set_ROI*.

The LUTs on Snapper-DIG16 are controlled by *DIG16_set_LUTs*, and the format of the captured image by *DIG16_set_format*. *DIG16_set_timer* allows camera exposure control. *DIG16_set_parameter* allows general parameters to be set.

Interrupt control of acquisition is possible using *DIG16_set_interrupts* and *DIG16_set_callback*.

The remaining functions will not get called in a typical application because they are called by *DIG16_initialize* with the correct settings for the camera. These functions are *DIG16_set_active_area*, *DIG16_set_alignment,* *DIG16_set_camera_info*, *DIG16_set_comms*, *DIG16_set_ctrl_io*, *DIG16_set_clk*, *DIG16_set_data_stream_ctrl*, *DIG16_set_image_data_width*, *DIG16_set_linescan_ctrl*, and *DIG16_set_ROI_rounding*.

PARAMETER READBACK FUNCTIONS

Some of these functions are intended to avoid the need for an application to keep shadow copies of Snapper-DIG16 settings. These are *DIG16_get_active_area*, *DIG16_get_camera_LSB*, *DIG16_get_camera_MSB*, *DIG16_get_camera_type*, *DIG16_get_data_width*, *DIG16_get_ROI* and *DIG16_get_subsample*.

DIG16_get_LUT_max_addr and *DIG16_get_ROI_max* return maximum settings for the camera in use, and *DIG16_get_property* returns hardware and firmware information about the Snapper-DIG16.

DIG16_get_ID and *DIG16_get_rev* return the hardware ID and revision of the Snapper-DIG16 in use.

Finally *DIG16_get_parameter* returns general information about the Snapper-DIG16.

CAMERA SPECIFIC FUNCTIONS

These provide control of operation modes of individual cameras. The list of these functions will be extended as more cameras are supported.

Error Returns

Almost all of the Snapper-DIG16 library functions return a *Terr* apart from several Boolean functions. *Terr* is a 32 bit unsigned integer, with the bit positions defined as follows:

31 to 24	Hardware identifier/revision (returned on error, otherwise 0 is returned). This is used to allow a top level calling function to determine the library in which the error occurred, and is actually read from the hardware itself. Clearing bits 26 to 24 leaves the hardware identifier, which is: on Snapper-DIG16 - 0xC0 (#defined as *DIG16_ID*) on Snapper-PMC-DIG16 - 0xC8 (#defined as *DIG16_PMC_ID*) Bits 26 to 24 give the hardware revision level. Initial Snapper-DIG16s have the value 0x00.
23 to 16	Error number, otherwise 0 if no error.
15 to 0	Function return value.

If a function call is successful, it returns *ASL_OK* (which is #defined as 0) or the requested parameter. If an error occurs, an error number is returned in bits 23 to 16 along with the hardware or library identifier in bits 31 to 24. See the "Snapper Error Handling Programmer's Manual" in the Developer's Guide section of the Snapper Developer's Manual for more details on error returns.

Sample Applications

The following examples are minimal programs for area scan and line scan modes. As with all sample code in this manual, error handling has been omitted for clarity, but apart from not handling errors cleanly these are usable programs. The Snapper SDK includes sample applications, both as executables and as source code, which provide a useful reference of 'real' code and are probably the best starting point for developing custom applications. For examples of how to display images under different operating systems see the examples in the TMG Library Programmer's Manual.

AREA SCAN MODE EXAMPLE

The following program captures an image from a Kodak Megaplus 1.4 camera and saves it to a file:

```
#include <asl_inc.h>

int main(ui16 argc, char** argv)
{
   Thandle Hdig16;       /* Handle to Snapper-DIG16 */
   Thandle Hbase;        /* Handle to baseboard */
   Thandle Hvid_image;   /* Handle to image */

   /* Initialize baseboard and Snapper module */
   Hbase = ASL_get_ret(BASE_create(BASE_AUTO));
   Hdig16 = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
   Hvid_image = TMG_image_create();

   /* Set required Snapper mode */
   DIG16_initialize(Hdig16, DIG16_KODAK_MPLUS14, 8);

   /* Set up image parameters */
   DIG16_set_image(Hdig16, Hvid_image);

   /* Capture image and write it to a file */
   DIG16_capture_to_image(Hdig16, Hvid_image, DIG16_START_AND_WAIT);
   TMG_image_set_outfilename(Hvid_image, "y8.tif");
   TMG_image_write(Hvid_image, TMG_NULL, TMG_TIFF, TMG_RUN);

   /* Free memory and exit */
   BASE_destroy(BASE_ALL_HANDLES);
   TMG_image_destroy(TMG_ALL_HANDLES);
}
```

LINE SCAN MODE EXAMPLE

The following program demonstrates the use of line scan mode using a Dalsa CLCx series camera and synchronous readout of data. For a more detailed example using dual buffers (in host memory) and interrupt driven acquisition, see the 32 bit Windows application example "D16".

```
#include <asl_inc.h>

int main(ui16 argc, char** argv)
{
   Thandle Hdig16, Hbase, Hvid_image; /* Handles to Snapper, baseboard & image */
   Tboolean finished = FALSE;         /* Controls when to stop line scan capture */
   i16 roi_array[ASL_SIZE_2D_ROI];    /* For ROI set and get calls */

   /* Initialize baseboard and Snapper module */
   Hbase = ASL_get_ret(BASE_create(BASE_AUTO));
   Hdig16 = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
   Hvid_image = TMG_image_create();

   /* Set Snapper mode for a Dalsa CLC camera with 2048 pixel CCD. We need to
    * override the default active area set by DIG16_initialize, and then set the ROI
    */
   DIG16_initialize(Hdig16, DIG16_DALSA_CLCX_1CH, 8);
   DIG16_get_active_area(Hdig16, roi_array);
   roi_array[ASL_ROI_X_LENGTH] = 2048;
   DIG16_set_active_area(Hdig16, roi_array);
   DIG16_get_ROI_max(Hdig16, roi_array);
   DIG16_set_ROI(Hdig16, DIG16_ROI_SET, roi_array);
   DIG16_set_image(Hdig16, Hvid_image);
   BASE_set_timer(Hbase, BASE_TIMER_ASTABLE, (ui32) 1000); /* 1 line every 2000us */

   /* Start continuous capture of images - first prepare DIG16_read_video_data
    * to receive data, then start the actual capture
    */
   DIG16_read_video_data(Hdig16, Hvid_image, TMG_INIT);
   DIG16_capture_to_image(Hdig16, Hvid_image, DIG16_START_AND_RETURN);
   TMG_image_set_flags(Hvid_image, TMG_LOCKED, TRUE);   /* Speed up loop */

   /* Now continuously read in the data as it is captured */
   while (finished == FALSE)
   {
      DIG16_read_video_data(Hdig16, Hvid_image, TMG_STRIP);   /* Read the image */

      /* Process or display the image - here an imaginary image processing routine
       * is called - to use the example replace this line with some real code
       */
      finished = process_image(Hvid_image);
   }

   /* Stop the continuous capturing, then inform DIG16_read_video_data that no more
    * data will be read in this continuous capture. Finally free memory and exit
    */
   DIG16_capture_to_image(Hdig16, Hvid_image, DIG16_CAPTURE_END);
   DIG16_read_video_data(Hdig16, Hvid_image, TMG_RESET);

   BASE_destroy(BASE_ALL_HANDLES);
   TMG_image_destroy(TMG_ALL_HANDLES);
}
```

DATA STREAM MODE EXAMPLE

The following program demonstrates the use of data stream mode. This application cannot be used under MS-DOS or Windows 3.1, where explicit calls must be made to *DIG16_read_video_data*. See *DIG16_read_video_data* for more information and example code.

```
#include <asl_inc.h>

int main(ui16 argc, char** argv)
{
   Thandle Hdig16, Hbase, Hvid_image; /* Handles to Snapper, baseboard & image */
   Tboolean finished = FALSE;         /* Controls when to stop capture */

   /* Initialize baseboard and Snapper module */
   Hbase = ASL_get_ret(BASE_create(BASE_AUTO));
   Hdig16 = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
   Hvid_image = TMG_image_create();

   /* Setup Snapper in data stream mode, and request 10000 pixels per capture, and
    * use both line and frame enables to acquire data
    */
   DIG16_initialize(Hdig16, DIG16_AIA_DATA_STREAM, 8);
   DIG16_set_data_stream_ctrl(Hdig16, DIG16_DSTRM_LINE_ACQ_ENABLE |
            DIG16_DSTRM_FRAME_ACQ_ENABLE | DIG16_DSTRM_LINE_START_ENABLE |
            DIG16_DSTRM_FRAME_START_ENABLE);
   DIG16_set_parameter(Hdig16, DIG16_PIXEL_COUNT, 10000);
   DIG16_set_image(Hdig16, Hvid_image);
   TMG_image_set_flags(Hvid_image, TMG_LOCKED, TRUE);   /* Speed up loop */

   /* Now continuously capture data */
   while (finished == FALSE)
   {
      DIG16_capture_to_image(Hdig16, Hvid_image, DIG16_START_AND_RETURN);

      /* Send any commands here needed to make camera send 10000 pixels of data */

      /* Wait for image capture complete - in a typical application
       * the previous image might get processed at this point
       */
      while (ASL_get_ret(DIG16_get_capture_status(Hdig16)) !=
                                  DIG16_CAPTURE_COMPLETE)
      {}

      /* Update ‘finished’ boolean here */
   }

   BASE_destroy(BASE_ALL_HANDLES);
   TMG_image_destroy(TMG_ALL_HANDLES);
}
```

Function List

Functions Supported In All Modes

INITIALIZATION FUNCTION

DIG16_initialize
DIG16_initialize_LUTs

IMAGE CAPTURE FUNCTIONS

DIG16_capture_to_image
DIG16_get_capture_status
DIG16_is_trigger_started
DIG16_set_image

CONFIGURATION FUNCTIONS

DIG16_set_alignment
DIG16_set_callback
DIG16_set_camera_info
DIG16_set_capture
DIG16_set_clk
DIG16_set_comms
DIG16_set_ctrl_io
DIG16_set_format
DIG16_set_image_data_width
DIG16_set_interrupts
DIG16_set_LUTs
DIG16_set_parameter
DIG16_set_timer
DIG16_set_trigger

PARAMETER READBACK FUNCTIONS

DIG16_get_camera_LSB
DIG16_get_camera_MSB
DIG16_get_camera_type
DIG16_get_ctrl_io_status
DIG16_get_data_width
DIG16_get_FIFO_status
DIG16_get_ID
DIG16_get_LUT_max_addr
DIG16_get_parameter
DIG16_get_property
DIG16_get_rev
DIG16_get_subsample
DIG16_is_field1_captured
DIG16_is_trigger_started

Functions Supported In Some Modes

A = AREA SCAN D = DATA STREAM L = LINE SCAN

DL *DIG16_read_video_data*
A *DIG16_is_field1_captured*

AL *DIG16_set_active_area*
D *DIG16_set_data_stream_ctrl*
L *DIG16_set_linescan_ctrl*
AL *DIG16_set_ROI*
AL *DIG16_set_ROI_rounding*

AL *DIG16_get_active_area*
AL *DIG16_get_ROI*
AL *DIG16_get_ROI_max*

The functions are described in alphabetical order in the following pages.

See the end of the manual for camera specific functions.

DIG16_capture_to_image

USAGE

Terr DIG16_capture(Thandle Hdig16, Thandle Himage, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
Himage	Handle to image.
mode	Required capture mode.

DESCRIPTION

This function is used to both initiate a video capture on Snapper-DIG16 and transfer the captured data to an Himage. The module must be configured in the required mode before this routine is called.

If the operating system supports threads (i.e. independent paths of control within a program or process) the data transfer is implemented as a separate thread.

MODE

DIG16_START_AND_WAIT	The function does not return until capture and data transfer is complete, including waiting for an external trigger if selected. This parameter is not supported in line scan mode.
DIG16_START_AND_RETURN	The capture is initiated, a data transfer thread started, then control immediately returns to the calling function. This mode allows the time taken by the data transfer to be used by the software to perform other processing. The controlling program must call *DIG16_get_capture_status* when it has completed its processing. In line scan mode this parameter must always be to start a capture. In area scan mode this option is not supported if the operating system does not support threads (e.g. MS-DOS and Windows 3.1).
DIG16_CAPTURE_END	A capture previously started by *DIG16_START_AND_RETURN* will be terminated on completion of acquisition of the current image. This parameter is needed in line scan mode to stop the acquisition of data.
DIG16_CAPTURE_ABORT	A capture previously started by *DIG16_START_AND_RETURN* will be terminated. This mode must be used to stop the data transfer thread if *DIG16_get_capture_status* has not returned *DIG16_CAPTURE_COMPLETE*. This parameter is not supported if the operating system does not support threads (e.g. MS-DOS and Windows 3.1).

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_NOT_SUPPORTED	*DIG16_START_AND_RETURN* was requested in area scan mode for an operating system which does not support threads.

ASLERR_TIMEOUT	The capture timed out. If external trigger is disabled then this error indicates that the hardware did not complete the required video capture. This is probably due to a lack of video, or the wrong clock source being selected. If external trigger is enabled then this error could also indicate that an active edge of the external trigger source was not detected within the timeout period.

EXAMPLES

The following code will capture and return when capture is complete:

```
while (DisplayLive == TRUE)
{
   DIG16_capture_to_image(Hdig16, Himage, DIG16_START_AND_WAIT);
   process_image(Himage, ... );
}
```

The following code will more efficiently process one image whilst capturing the next:

```
DIG16_capture_to_image(Hdig16, Himage2, DIG16_START_AND_WAIT);
while (DisplayLive == TRUE)
{
   DIG16_capture_to_image(Hdig16, Himage1, DIG16_START_AND_RETURN);
   process_image(Himage2, ... );
   while (ASL_get_ret(DIG16_get_capture_status(Hdig16)) !=
       DIG16_CAPTURE_COMPLETE)
   {}
   DIG16_capture_to_image(Hdig16, Himage2, DIG16_START_AND_RETURN);
   process_image(Himage1, ... );
   while (ASL_get_ret(DIG16_get_capture_status(Hdig16)) !=
       DIG16_CAPTURE_COMPLETE)
   {}
}
```

BUGS / NOTES

If timeouts are required for capture it is recommended that they are controlled within the application by using the *DIG16_START_AND_RETURN* parameter together with the clock functions available within the operating system in use.

If external triggers are enabled (see *DIG16_set_trigger*) then the function will not return until the external trigger event has occurred, even if *DIG16_capture_to_image* is called with *DIG16_START_AND_RETURN*. An alternative method is to disable triggers for acquisition, and use the external trigger input as an interrupt source. The interrupt service routine can then call *DIG16_capture_to_image* with *DIG16_START_AND_RETURN* which will then return immediately.

SEE ALSO

DIG16_set_capture, *DIG16_get_capture_status*, *DIG16_is_trigger_started*, *DIG16_set_trigger*, *DIG16_read_video_data*.

DIG16_get_active_area

USAGE

Terr DIG16_get_active_area(Thandle Hdig16, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.	
roi	ROI array with four elements, with #defined element names:	
	ASL_ROI_X_START	Horizontal start position of ROI (0 = left of image).
	ASL_ROI_Y_START	Vertical start position of ROI (0 = top of image).
	ASL_ROI_X_LENGTH	Horizontal width of ROI.
	ASL_ROI_Y_LENGTH	Vertical height of ROI.
	The values in the *roi* array passed in are ignored.	

DESCRIPTION

This function fetches the active area as set by the most recent call to *DIG16_set_active_area* and returns it in the *roi* array.

For conventional area scan cameras the pixel referenced by *ASL_ROI_X_START* and *ASL_ROI_Y_START* is subsequently used by the ROI functions as pixel [0,0].

In line scan mode *ASL_ROI_Y_START* is not used, and is returned set to 0; and *ASL_ROI_Y_LENGTH* is returned set to 1. The pixel referenced by *ASL_ROI_X_START* is used by the ROI functions as pixel 0.

All the coordinates are based upon raw image sizes in pixels and lines, ***not*** sub-sampled ones. The horizontal and vertical resolutions are 1 pixel and 1 line respectively.

RETURNS

This function returns the current ROI in the *roi* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

To display the active area:

```
DIG16_get_active_area(Hdig16, roi);
printf("\nActive area X start is %d", (int)roi[ASL_ROI_X_START]);
printf("\nActive area Y start is %d", (int)roi[ASL_ROI_Y_START]);
printf("\nActive area X length is %d", (int)roi[ASL_ROI_X_LENGTH]);
printf("\nActive area Y length is %d", (int)roi[ASL_ROI_Y_LENGTH]);
```

BUGS / NOTES

There are no known bugs.

This function is not supported in data stream mode.

SEE ALSO

DIG16_set_active_area, *DIG16_get_ROI*, *DIG16_get_ROI_max*, *DIG16_set_ROI*.

DIG16_get_camera_LSB

USAGE

Terr DIG16_get_camera_LSB(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the bit which the LSB of the camera is connected to, as set by *DIG16_set_camera_info*. For two channel cameras the LSB of the first channel is returned.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
if (ASL_get_ret(DIG16_get_camera_LSB(Hdig16)) == 7)
   printf("\Camera LSB connected to bit MSB-7");
```

BUGS / NOTES

The <camera lsb> is returned in the lower 8 bits, if successful.

There are no known bugs.

SEE ALSO

DIG16_get_camera_MSB, *DIG16_get_data_width*.

DIG16_get_camera_MSB

USAGE

Terr DIG16_get_camera_MSB(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the bit which the MSB of the camera is connected to, as set by *DIG16_set_camera_info*. For two channel cameras the MSB of the first channel is returned.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
if (ASL_get_ret(DIG16_get_camera_MSB(Hdig16)) == 0)
   printf("\Camera MSB connected to bit MSB");
```

BUGS / NOTES

The <camera msb> is returned in the lower 8 bits, if successful.

There are no known bugs.

SEE ALSO

DIG16_get_camera_LSB, *DIG16_get_data_width*.

DIG16_get_camera_type

USAGE

Terr DIG16_get_camera_type(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the camera type in use, as set by *DIG16_initialize*, for example *DIG16_KODAK_MPLUS14, DIG16_PULNIX_TM9700,* etc.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
if (ASL_get_ret(DIG16_get_camera_type(Hdig16)) == DIG16_KODAK_MPLUS14)
   printf(“\Kodak Megaplus 1.4 in use”);
```

BUGS / NOTES

The <camera type> is returned in the lower 16 bits, if successful.

There are no known bugs.

SEE ALSO

DIG16_initialize.

DIG16_get_capture_status

USAGE

Terr DIG16_get_capture_status(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required mode.

DESCRIPTION

This function is used to test whether the current video capture and data transfer initiated by *DIG16_capture_to_image* has completed. This is used in conjunction with *DIG16_capture_to_image* called with the *DIG16_START_AND_RETURN* parameter.

Note that it is important to call *DIG16_get_capture_status* even if you already know that the capture has completed. This is because the function call, on detecting that capture has completed, performs some tidy up code before returning.

MODE

DIG16_START_AND_RETURN	This tests the capture status and returns immediately with the status. If the data transfer has completed it returns *DIG16_CAPTURE_COMPLETE* if the capture was successful, or the error return from the read thread if it was not. It returns *DIG16_CAPTURE_IN_PROGRESS* from the time that a capture is initiated until the data transfer completes. *DIG16_CAPTURE_IN_PROGRESS* is also returned after the capture has been initiated, but before a valid external trigger has occurred.
DIG16_START_AND_WAIT	This does not return until the data transfer has completed, when it returns *DIG16_CAPTURE_COMPLETE* if the capture was successful, or the error return from the read thread if it was not. Note that internally the function uses the operating system's thread synchronisation system to wait for the capture to complete, i.e. it does not use a polling loop, so it uses very little CPU time.

This function is not needed for operating systems which do not support threads (e.g. MS-DOS and Windows 3.1) because they do not support *DIG16_START_AND_RETURN.*

RETURNS

ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_NOT_SUPPORTED	Called for an operating system which does not support threads.
ASLERR_CAPTURE_COMPLETE	The capture has successfully completed.
ASLERR_CAPTURE_IN_PROGRESS	The capture has not yet completed.

The function will also return any errors from the read thread on completion of the thread.

EXAMPLES

The following code initiates a capture, then processes the previous frame whilst capturing the next, and then waits for the capture to complete before reading the new frame:

```
DIG16_capture_to_image(Hdig16, Himage2, DIG16_START_AND_WAIT);
while (DisplayLive == TRUE)
{
        DIG16_capture_to_image(Hdig16, Himage1, DIG16_START_AND_RETURN);
        process_image(Himage2, ... );
        DIG16_get_capture_status(Hdig16, DIG16_START_AND_WAIT);
        DIG16_capture_to_image(Hdig16, Himage2, DIG16_START_AND_RETURN);
        process_image(Himage1, ... );
        DIG16_get_capture_status(Hdig16, DIG16_START_AND_WAIT);
    }
```

BUGS / NOTES

There are no known bugs.

This function also manages the synchronisation of the data transfer thread. Therefore if *DIG16_get_capture_status* does not return *DIG16_CAPTURE_COMPLETE* then *DIG16_capture_to_image* must be called with the parameter *DIG16_ABORT_CAPTURE* to tidy up the data transfer thread.

SEE ALSO

DIG16_capture_to_image.

DIG16_get_ctrl_io_status

USAGE

Terr DIG16_get_ctrl_io_status(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the status of the 4 general purpose control I/O bits; I/O A to I/O D inclusive. The function *DIG16_set_ctrl_io* is used to configure the bits as input or outputs. If an individual bit is set as an output, *DIG16_get_ctrl_io_status* will return the value of that output bit.

Each bit can be masked off using *DIG16_STAT_IO_A_IN* to *DIG16_STAT_IO_D_IN* inclusive.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
bStatus = ASL_get_ret(DIG16_get_ctrl_io_status(Hdig16));
if ((bStatus & DIG16_STAT_IO_A_IN) != 0)
   printf(“Status bit I/O A is High”);
else
   printf(“Status bit I/O A is Low”);
```

BUGS / NOTES

The <control io status> is returned in the lower 8 bits, if successful.

There are no known bugs.

SEE ALSO

DIG16_set_ctrl_io.

DIG16_get_data_width

USAGE

Terr DIG16_get_data_width(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required width parameter.

DESCRIPTION

This function returns the number of valid bits in the requested device.

MODE

DIG16_CAMERA_DATA_WIDTH	This function returns the number of valid bits from the camera, as set by *DIG16_set_camera_info*.
DIG16_IMAGE_DATA_WIDTH	This function returns the number of valid bits in the resulting image, as set by *DIG16_set_image_data_width*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
DataWidth = (ASL_get_ret(DIG16_get_data_width(Hdig16,
       DIG16_CAMERA_DATA_WIDTH));
   printf(“\Camera data width is %d bits”, (int) DataWidth);
```

BUGS / NOTES

The <xxx data width> is returned in the lower 16 bits, if successful.

There are no known bugs.

SEE ALSO

DIG16_get_camera_LSB, *DIG16_get_camera_MSB*.

DIG16_get_FIFO_status

USAGE

Terr DIG16_get_FIFO_status(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the status of the Snapper-DIG16 FIFO. The actual status flags are *DIG16_STAT_FIFO_EMPTY*, *DIG16_STAT_FIFO_ALMOST_EMPTY*, *DIG16_STAT_FIFO_ALMOST_FULL* and *DIG16_STAT_FIFO_FULL*.

The *DIG16_STAT_FIFO_FULL* status is an error condition and is latched in the hardware. Once set, it will not be cleared until the next acquisition is started. The other three bits are not latched and the status values can change throughout a capture. The actual levels of the almost empty and almost full flags are fixed and used by the libraries for control purposes. User applications should not rely on them remaining fixed in the future but they can be used as an indication that the FIFO is filling up rather than having reached a finite level.

With the exception of the latched full flag, the other flags cannot be set together because the FIFO cannot be simultaneously almost full, almost empty and empty.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
bStatus = ASL_get_ret(DIG16_get_FIFO_status(Hdig16));
if ((bStatus & DIG16_STAT_FIFO_EMPTY) != 0)
   printf(“The FIFO is empty”);
else
   printf(“The FIFO is not empty”);
```

BUGS / NOTES

The <FIFO status> is returned in the lower 8 bits, if successful.

There are no known bugs.

SEE ALSO

-

DIG16_get_ID

USAGE

Terr DIG16_get_ID(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the hardware identifier of the Snapper, so that an application can check whether it is running on a Snapper-DIG16 or a Snapper-PMC-DIG16.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
if (ASL_get_ID(DIG16_get_rev(Hdig16)) == DIG16_ID)
   printf(“\nRunning on Snapper-DIG16”);
else if (ASL_get_ret(DIG16_get_rev(Hdig16)) == DIG16_PMC_ID)
   printf(“\nRunning on Snapper-PMC-DIG16”);
...
```

BUGS / NOTES

The <ID> (either *DIG16_ID* or *DIG16_PMC_ID*) is returned in the lower 8 bits, if successful.

This function is included for compatibility with existing applications. All new applications should use *DIG16_get_parameter*.

There are no known bugs.

SEE ALSO

DIG16_get_parameter, *DIG16_get_rev*.

DIG16_get_LUT_max_addr

USAGE

ui32 DIG16_get_LUT_max_addr(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the minimum size of array which is needed when *DIG16_set_LUTs* is called. This is based on the value of the data width set by *DIG16_set_camera_info*.

RETURNS

See above. The value '0' is returned if any error occurs.

EXAMPLES

```
lut_size = DIG16_get_LUT_max_addr(Hdig16);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_set_LUTs.

DIG16_get_parameter

USAGE

Terr DIG16_get_parameter(Thandle Hdig16, ui16 parameter)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
parameter	The parameter to return.

DESCRIPTION

This function returns various parameters from the internal structure associated with the Snapper-DIG16 handle.

PARAMETER

DIG16_BASEBOARD_HANDLE	The handle to the baseboard which the Snapper-DIG16 is fitted on. Type (*Thandle, ui32)*.
DIG16_COMMS_JUMPER	This returns the setting of the RS-232 / RS-422 jumper which sets the comms format. The return value is either *DIG16_COMMS_RS232* or *DIG16_COMMS_RS422*, or on an earlier board where the setting of the jumper cannot be read, *DIG16_COMMS_UNKNOWN.* Type (*Tparam, ui32)*.
DIG16_FIFO_BYTES	This returns the number of bytes which can be stored in the on-board FIFO on Snapper-DIG16. Type (*ui32)*.
DIG16_FIFO_WIDTH	This returns the data width of the on-board FIFO on Snapper-DIG16, which is 16 on earlier boards, and 32 on more recent boards including all Snapper-PMC-DIG16 boards. Type (*ui16)*.
DIG16_PIXEL_COUNT	This returns the number of pixels which get stored per capture when in data stream mode. Type (*ui32)*.
DIG16_TIMEOUT_BEFORE_CAPTURE	This returns the timeout value in milliseconds for the period from when *DIG16_capture_to_image* is called to when the first data is received. Type (*ui32)*.
DIG16_TIMEOUT_DATA_TRANSFER	This returns the timeout value in milliseconds for the data transfer, i.e. following the first data being received. Type (*ui32)*.
DIG16_TIMEOUT_TRIGGER	This returns the time in milliseconds allowed before acquisition times out due to lack of a trigger input. Type (*ui32)*.
DIG16_TWO_CHANNEL	This returns *TRUE* if the camera has two data channels, or *FALSE* if it has one data channel. Type (*Tboolean)*.
DIG16_IS_CAMERA_INTERLACED	This returns *TRUE* if the camera supports interlaced fields output, or *FALSE* otherwise. Type (*Tboolean)*.
DIG16_ID_VALUE	This returns the ID as read from the hardware, which distinguishes between all the different Snapper-DIG16 variants. Type *(ui8)*.
DIG16_REV_VALUE	This returns the board revision as read from the hardware, which distinguishes between the hardware revisions of the Snapper. Type *(ui8)*.
DIG16_IDREV_VALUE	This returns the ID and board revision as read from the hardware, which distinguishes between the different hardware revisions of the Snapper variants. Type *(ui8)*.

DIG16_FAMILY_VALUE	This returns *DIG16_FAMILY_ID*. Although this parameter always returns the same value, it is used to provide a mechanism for distinguishing future Snapper-DIG16 variants, and for compatibility with the *DIG16_get_parameter* call. Type *(ui8)*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The parameter value is invalid.
ASLERR_NOT_RECOGNIZED	The ID value read back from the Snapper hardware is not recognized.

EXAMPLES

```
if (ASL_get_parameter(Hdig16, DIG16_ID_VALUE) == DIG16_ID)
   printf("\nRunning on Snapper-DIG16");
else if (ASL_get_parameter(Hdig16, DIG16_ID_VALUE) == DIG16_PMC_ID)
   printf("\nRunning on Snapper-PMC-DIG16");
...
```

EXAMPLES

See *DIG16_set_timer* for an example code fragment.

BUGS / NOTES

The function returns a type *Terr* (*ui32* - an unsigned 32 bit integer). Therefore a cast may be need depending on the parameter type (given above for each parameter).

There are no known bugs.

SEE ALSO

DIG16_set_parameter, *DIG16_get_property*.

DIG16_get_property

USAGE

*Terr DIG16_get_property(Thandle Hdig16, char *property, char *value)*

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
property	A character string or name of the property to access.
value	The property result string. (Must point to a buffer of at least 128 bytes.)

DESCRIPTION

This function returns various property strings associated with Snapper-DIG16.

The property strings are as follows:

"fpgadate"	**Snapper FPGA Date**: This retrieves the date and time string associated with current control FPGA file in use. It is unlikely that this function will ever be needed, but it can be useful to detect old versions of Snapper control FPGA information. (i.e. the date string is used as a revision level). The format of the returned date string is dd-mmm-yy hh:mm.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The property value is invalid.

EXAMPLES

To print the FPGA date:

```
char string[256];

DIG16_get_property(Hdig16, “fpgadate”, string);
printf(“Snapper-Dig16 FPGA date: %s”, string);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_get_parameter.

DIG16_get_rev

USAGE

Terr DIG16_get_rev(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the hardware revision level of the Snapper.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
if (ASL_get_ret(DIG16_get_rev(Hdig16)) == 0)
   printf("\nRunning on issue 1 Snapper-DIG16");
else if (ASL_get_ret(DIG16_get_rev(Hdig16)) == 1)
   printf("\nRunning on issue 2 Snapper-DIG16");
...
```

BUGS / NOTES

The <rev> is returned in the lower 8 bits, if successful.

There are no known bugs.

This function is included for compatibility with existing applications. All new applications should use *DIG16_get_parameter*.

SEE ALSO

DIG16_get_parameter, *DIG16_get_ID*.

DIG16_get_ROI

USAGE

Terr DIG16_get_ROI(Thandle Hdig16, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.	
roi	ROI array with four elements, with #defined element names:	
	ASL_ROI_X_START	Horizontal start position of ROI (0 = left of image).
	ASL_ROI_Y_START	Vertical start position of ROI (0 = top of image).
	ASL_ROI_X_LENGTH	Horizontal width of ROI.
	ASL_ROI_Y_LENGTH	Vertical height of ROI.
	The values in the *roi* array passed in are ignored.	

DESCRIPTION

This function fetches the current ROI (Region of Interest) and returns it in the *roi* array.

The top left corner of the image is defined by the *ASL_ROI_X_START* and *ASL_ROI_Y_START* coordinates and the image size defined with the *ASL_ROI_X_LENGTH* and *ASL_ROI_Y_LENGTH* values. All the coordinates are based upon raw image sizes in pixels and lines, ***not*** sub-sampled ones.

In line scan mode *ASL_ROI_Y_START* is always 0, and *ASL_ROI_Y_LENGTH* is the number of lines which get read per call to *DIG16_read_video_data*.

RETURNS

This function returns the current ROI in the *roi* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

To display the current ROI:

```
DIG16_get_ROI(Hdig16, roi);
printf(“\nROI X start is %d”, (int)roi[ASL_ROI_X_START]);
printf(“\nROI Y start is %d”, (int)roi[ASL_ROI_Y_START]);
printf(“\nROI X length is %d”, (int)roi[ASL_ROI_X_LENGTH]);
printf(“\nROI Y length is %d”, (int)roi[ASL_ROI_Y_LENGTH]);
```

BUGS / NOTES

There are no known bugs.

This function is not supported in data stream mode.

SEE ALSO

DIG16_set_ROI, *DIG16_get_ROI_max*, *DIG16_set_active_area*, *DIG16_get_active_area*.

DIG16_get_ROI_max

USAGE

Terr DIG16_get_ROI_max(Thandle Hdig16, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
roi	ROI array with four elements, with #defined element names:

ASL_ROI_X_START	Horizontal start position of ROI (0 = left of image).
ASL_ROI_Y_START	Vertical start position of ROI (0 = top of image).
ASL_ROI_X_LENGTH	Horizontal width of ROI.
ASL_ROI_Y_LENGTH	Vertical height of ROI.

The values in the *roi* array passed in are ignored.

DESCRIPTION

This function fetches the maximum usable ROI (Region of Interest) for the camera in use and returns it in the *roi* array.

The maximum size is defined by the *ASL_ROI_X_LENGTH* and *ASL_ROI_Y_LENGTH* values, so the *ASL_ROI_X_START* and *ASL_ROI_Y_START* coordinates are always returned as '0'. The coordinates are based upon raw image sizes in pixels and lines, ***not*** sub-sampled ones.

For conventional area scan cameras the values returned are calculated from the information passed to *DIG16_set_active_area*.

In line scan mode *ASL_ROI_Y_START* is always 0, and the coordinate *ASL_ROI_Y_LENGTH* is the maximum number of lines which will fit into the FIFO on the Snapper assuming that the full line is captured. This is calculated from the full width of the line (from the information passed to *DIG16_set_active_area*) and the size of the FIFO. This is not a limiting value - if sufficient memory is in the host computer to allow large DMA transfers then very large numbers of lines can be captured. The value is given to help optimising applications running under MS-DOS or Windows 3.1. See the Concepts section for more information.

RETURNS

This function returns the maximum ROI in the *roi* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

To set the maximum allowable ROI:

```
DIG16_get_ROI_max(Hdig16, roi);
DIG16_set_ROI(Hdig16, DIG16_ROI_SET, roi);
```

BUGS / NOTES

There are no known bugs.

This function is not supported in data stream mode.

SEE ALSO

DIG16_get_ROI, *DIG16_set_ROI*, *DIG16_set_active_area*, *DIG16_get_active_area*.

DIG16_get_subsample

USAGE

Terr DIG16_get_subsample(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function returns the current sub-sample factor, as set by *DIG16_set_capture*.

RETURNS

This function returns the sub-sample ratio, i.e. *DIG16_SUB_X1, DIG16_SUB_X2, DIG16_SUB_X4* or *DIG16_SUB_X8*. Possible error codes:

ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

```
if (ASL_get_ret(DIG16_get_subsample(Hdig16)) == DIG16_SUB_X2)
   printf("\Currently using times 2 sub-sample");
else if (ASL_get_ret(DIG16_get_subsample(Hdig16)) == DIG16_SUB_X4)
   printf("\Currently using times 4 sub-sample");
...
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_set_capture.

DIG16_initialize

USAGE

Terr DIG16_initialize(Thandle Hdig16, ui16 mode, ui16 required_bits)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required initialization mode.
required_bits	Required data width of final Himage.

DESCRIPTION

This function is used to initialize the Snapper-DIG16 module to the required settings for the camera in use.

It makes calls to *DIG16_set_alignment*, *DIG16_set_callback*, *DIG16_set_capture*, *DIG16_set_clk*, *DIG16_set_ctrl_io*, *DIG16_set_format*, *DIG16_set_image_data_width*, *DIG16_set_interrupts*, *DIG16_set_LUTs*, *DIG16_set_ROI*, and *DIG16_set_trigger*. It also calls camera specific functions such as *DIG16_set_mplus_ctrll*. See the supplied source of *DIG16_initialize* in "dig16ini.c" to see which modes are set for each camera.

The parameter *required_bits* indicates how many bits are needed in the final image. It controls the call to *DIG16_set_format*, the *scaling* parameter in the call to *DIG16_set_LUTs*, and the call to *DIG16_set_image_data_width*. If *required_bits* is greater than eight *DIG16_set_format* is set to Y16 mode generating a *TMG_Y16* image. If *required_bits* is eight or less *DIG16_set_format* is set to Y8 mode generating a *TMG_Y8* image. If *required_bits* is less than the camera's data width the *DIG16_set_LUTs* parameter *scaling* is set to the required non-zero value. If *required_bits* is greater than the camera's data width an error is returned. To get the full data width of the camera, without needing to know what this is, pass the value *DIG16_INIT_FULL_DATA_WIDTH*.

MODE

The *mode* flag identifies the camera in use:

DIG16_AIA_DATA_STREAM	This is a generic set up for AIA pinout cameras which are MSB aligned with up to 16 data bits. The Snapper-DIG16 is put into data stream mode, so there is no concept of region of interest. The application needs to call *DIG16_set_data_stream_ctrl* and *DIG16_set_parameter* passing *DIG16_PIXEL_COUNT* depending on the camera connected.
DIG16_BASLER_L120_1CH *DIG16_BASLER_L120_2CH*	For Basler L120 line scan cameras, with either 1 or 2 channel outputs. These cameras are available with a variety of CCD widths, so DIG16_initialize sets a small width as default which can be overridden by calling *DIG16_set_active_area*. See the example line scan application at the front of the manual.
DIG16_CUSTOM_CAMERA	This should be used when using an area scan camera which is not listed below. The function still initializes the Snapper-DIG16, but does not call the functions listed above. Instead the application must call all these functions with the appropriate setting for the camera in use. See the Snapper Technical Note 5 in the Notes section at the end of the manual for more information. The parameter *required_bits* is ignored.
DIG16_CUSTOM_LINESCAN	This should be used when using a line scan camera which is not listed below. The function still initializes the Snapper-DIG16, but does not call the functions listed above. Instead the application must call all

	these functions with the appropriate setting for the camera in use. See the Snapper Technical Note 5 in the Notes section at the end of the manual for more information. The parameter *required_bits* is ignored.
DIG16_DALSA_CAD1	For Dalsa CA-D1 family area scan cameras. These cameras are available with a variety of CCD widths, so DIG16_initialize sets a small width as default which can be overridden by calling *DIG16_set_active_area*. See the example line scan application at the front of the manual.
DIG16_DALSA_CLCX_1CH *DIG16_DALSA_CLCX_2CH*	For Dalsa CLCx family line scan cameras with either 1 or 2 channel outputs (OS1 and OS2). These cameras are available with a variety of CCD widths, so DIG16_initialize sets a small width as default which can be overridden by calling *DIG16_set_active_area*. See the example line scan application at the front of the manual.
DIG16_DVC_08 *DIG16_DVC_10*	For the DVC DigitEyes 8 or 10 bit interlaced area scan cameras.
DIG16_HAMAMATSU_C4742	For the Hamamatsu Photonics C4742.
DIG16_KODAK_MPLUS14	For the Kodak Megaplus 1.4.
DIG16_KODAK_MPLUS14I	For the Kodak Megaplus 1.4i.
DIG16_KODAK_MPLUS16	For the Kodak Megaplus 1.6 or 1.6i.
DIG16_KODAK_MPLUS42	For the Kodak Megaplus 4.2.
DIG16_PULNIX_TM1000	For the Pulnix TM1000.
DIG16_PULNIX_TM1001	For the Pulnix TM1001.
DIG16_PULNIX_TM9700	For the Pulnix TM9700.
DIG16_PULNIX_TM9701	For the Pulnix TM9701.
DIG16_XILLIX_1400_10BIT *DIG16_XILLIX_1400_12BIT*	For the Xillix MicroImager 1400, in normal clocking mode, with either 10 or 12 bit output.
DIG16_XILLIX_1400_10BIT_BIN2 *DIG16_XILLIX_1400_12BIT_BIN2*	For the Xillix MicroImager 1400, in 2x2 binning mode. In this mode the MicroImager 1400 merges an area of 2x2 photosites into a single output value. The image resolution is therefore reduced by 2 in both the X and Y directions, which results in a reduced image resolution but an increased signal to noise performance, and hence greater dynamic range.
DIG16_XILLIX_1400_PMI	For the Xillix MicroImager 1400 PMI.

Note that all line scan cameras, including the mode *DIG16_CUSTOM_LINESCAN*, put the Snapper-DIG16 into "line scan mode". See the Concepts section of this manual for more details. Similarly all data stream cameras put the Snapper-DIG16 into "data stream mode".

See the release notes for details of additional cameras supported.

When cameras requiring serial communications are initialised, the serial port is configured in the correct mode (ie RS-422 or RS-232), but no communications take place. It is up to the user's application to implement the appropriate interface protocol using the *BASE_serial_xxx* family of functions.

RETURNS

This function will either return one of the following, or an error value from one of the lower level function calls listed above.

ASL_OK	If successful.

ASLERR_OUT_OF_RANGE	The *required_bits* parameter is larger than the camera width.
ASLERR_OUT_OF_MEMORY	There was insufficient memory to setup the array to pass to *DIG16_set_LUTs*.
ASLERR_BAD_PARAM	The *mode* parameter is invalid.

EXAMPLES

To initialize a Kodak Megaplus 1.4, with its full 8 data bits resolution:

```
DIG16_initialize(Hdig16, DIG16_KODAK_MPLUS14, DIG16_INIT_FULL_DATA_WIDTH);
```

To initialize a Kodak Megaplus 1.6, with its full 10 data bits resolution:

```
DIG16_initialize(Hdig16, DIG16_KODAK_MPLUS16, DIG16_INIT_FULL_DATA_WIDTH);
```

To initialize a Kodak Megaplus 1.6, getting only 8 data bits resolution:

```
DIG16_initialize(Hdig16, DIG16_KODAK_MPLUS16, 8);
```

This will return an *ASLERR_OUT_OF_RANGE* because the Kodak Megaplus 1.6 only has 10 data bits resolution:

```
DIG16_initialize(Hdig16, DIG16_KODAK_MPLUS16, 12);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_get_camera_type.

DIG16_initialize_LUTs

USAGE

Terr DIG16_initialize_LUTs(Thandle Hdig16, ui16 mode, ui16 required_bits)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required initialization mode.
required_bits	Required data width of final Himage.

DESCRIPTION

This function is used to initialize the Snapper-DIG16 LUT to the required settings for the camera in use. For many applications this function need not be called directly, because it is called by *DIG16_initialize*.

MODE

The *mode* flag is only used to identify which camera is being configured, so that a meaningful error code can be generated from *DIG16_initialize*. If this function is called directly from an application, then this value can be set to 0.

REQUIRED_BITS

The parameter *required_bits* indicates how many bits are needed in the final image, and controls the *scaling* parameter in the call to *DIG16_set_LUTs*. It must be in the range of 8 to 16 inclusive.

RETURNS

This function will either return one of the following, or an error value from one of the lower level function calls listed above.

ASL_OK	If successful.
ASLERR_OUT_OF_RANGE	The *required_bits* parameter is not between 8 and 16.
ASLERR_OUT_OF_MEMORY	There was insufficient memory to setup the array to pass to *DIG16_set_LUTs*.

EXAMPLES

See the "DIG16ini.c" file for examples.

BUGS / NOTES

There are no known bugs.

Because the function does not use the *mode* parameter, it does not check whether the size of the LUT is appropriate for specific camera.

SEE ALSO

DIG16_initialize.

DIG16_is_field1_captured

USAGE

Tboolean DIG16_is_field 1_captured(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function is used to determine whether field 1 or field 2 of a frame was captured in video memory. This is only valid if the mode *DIG16_NEXT_FIELD* is selected in *DIG16_set_capture*.

RETURNS

This function returns either *TRUE* (for field 1 captured) or *FALSE* (for field 2 captured).

BUGS / NOTES

There are no known bugs.

This function is not supported in data stream or line scan modes.

SEE ALSO

DIG16_set_capture.

DIG16_is_trigger_started

USAGE

Tboolean DIG16_is_trigger_started(Thandle Hdig16)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.

DESCRIPTION

This function is used to test whether an active edge of the external capture trigger has occurred. It returns *TRUE* from the first active edge of the external trigger until the capture has completed. At all other times *FALSE* is returned. Note that this function is only useful if *DIG16_capture_to_image* is called with the *START_AND_RETURN* parameter.

RETURNS

This function returns either *TRUE* (for active edge of trigger has occurred) or *FALSE* (for capture completed, or active edge of trigger has not occurred).

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_set_trigger.

DIG16_read_video_data

USAGE

Terr DIG16_read_video_data(Thandle Hdig16, Thandle Himage, ui16 TMG_action)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
Himage	Handle to image.
TMG_action	*TMG* mode flag.

DESCRIPTION

This function is supported in line scan mode and is used to read individual images containing a block a lines from Snapper-DIG16 into the image structure referenced by *Himage*. It is also supported with MS-DOS and Windows 3.1 to allow data stream mode to be supported on operating systems without thread support.

The function is called in three stages by using the *TMG_action* flags *TMG_INIT*, *TMG_STRIP* (or *TMG_ASYNC*) and *TMG_RESET*. The *Himage* must have been set up with a call to *DIG16_set_image* before calling the function with mode *TMG_INIT*.

TMG_ACTION

TMG_INIT	This parameter is used to initialize the read video data routine before starting a line scan mode capture.
TMG_STRIP	This parameter is used to read one image during a line scan mode capture. It would normally be called multiple times while the line scan capture is running. When called with this parameter, the *DIG16_read_video_data* call will not return until all the required data has been read.
TMG_ASYNC	This parameter is an asynchronous version of *TMG_STRIP*; that is, having initiated the data transfer the function returns before all the data has been read. The function *DIG16_get_capture_status* must be called to determine that the transfer has completed.
TMG_RESET	This parameter is used to reset the read video data routine after the completion a line scan mode capture.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_INCOMPATIBLE	The mode *TMG_STRIP* was used without previously using *TMG_INIT*.
ASLERR_OUT_OF_MEMORY	There is insufficient memory available to process the video data. If this occurs, either there is a memory leak within the application, or the strip size is too large (consult the TMG Programmer's Manual for further details).

EXAMPLES

For line scan see the line scan mode example in the Sample Applications section at the front of this manual.

For data stream mode under Windows 3.1 or MS-DOS, the following loop replaces the equivalent one in the data stream example at the front of the manual:

```
/* Now continuously capture data */
```

```
while (finished == FALSE)
{
   DIG16_capture_to_image(Hdig16, Hvid_image, DIG16_START_AND_RETURN);
   DIG16_read_video_data(Hdig16, Hvid_image, TMG_INIT);

   /* Send any commands here needed to make camera send 10000 pixels of data
       */

   DIG16_read_video_data(Hdig16, Hvid_image, TMG_STRIP);
  /*
   * Note: The use of TMG_STRIP above is equivalent to:
   * DIG16_read_video_data(Hdig16, Hvid_image, TMG_ASYNC);
   * DIG16_get_capture_status(Hdig16, DIG16_START_AND_WAIT);
   */
   DIG16_read_video_data(Hdig16, Hvid_image, TMG_RESET);

   /* Process image here */

   /* Update ‘finished’ boolean here */
}
```

BUGS / NOTES

TMG_INIT and *TMG_STRIP* are only necessary under MS-DOS, Windows 3.1 and MacOS. Other operating systems need not use them. Similarly *TMG_ASYNC* is not supported under MS-DOS, Windows 3.1 and MacOS.

This function is not supported for area scan cameras, where *DIG16_capture_to_image* captures and reads.

SEE ALSO

DIG16_capture_to_image.

DIG16_set_active_area

USAGE

Terr DIG16_set_active_area(Thandle Hdig16, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
roi	ROI array with four elements, with #defined element names:

ASL_ROI_X_START	Horizontal start position of ROI (0 = left of image).
ASL_ROI_Y_START	Vertical start position of ROI (0 = top of image).
ASL_ROI_X_LENGTH	Horizontal width of ROI.
ASL_ROI_Y_LENGTH	Vertical height of ROI.

DESCRIPTION

This function defines the active area for the camera in use. This is used to allow *DIG16_set_ROI* to automatically adjust invalid ROIs (Regions of Interest) so that they do not exceed the camera's active area. For many applications this function need not be called directly, because *DIG16_initialize* calls it to set the active area of the selected camera.

For conventional area scan cameras the pixel referenced by *ASL_ROI_X_START* and *ASL_ROI_Y_START* is subsequently used by the ROI functions as pixel [0,0].

In line scan mode *ASL_ROI_Y_LENGTH* should be set to 1 to indicate one line, and *ASL_ROI_Y_START* is not used, and should be set to 0. The pixel referenced by *ASL_ROI_X_START* is subsequently used by the ROI functions as pixel 0.

All the coordinates are based upon raw image sizes in pixels and lines, ***not*** sub-sampled ones. The horizontal and vertical resolutions are 1 pixel and 1 line respectively.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_OUT_OF_RANGE	The active area is too large for Snapper-DIG16, or includes negative coordinates.

EXAMPLES

For a camera with an active CCD area of 1024 by 512 pixels, which starts outputting data on pixel 20 of line 10:

```
roi[ASL_ROI_X_START] = 20;
roi[ASL_ROI_Y_START] = 10;
roi[ASL_ROI_X_LENGTH] = 1024;
roi[ASL_ROI_Y_LENGTH] = 512;
DIG16_set_active_area(Hdig16, roi);
```

For a camera which starts outputting data on pixel 33 of line 2, and continues outputting until pixel 1042 of line 788:

```
roi[ASL_ROI_X_START] = 33;
roi[ASL_ROI_Y_START] = 2;
roi[ASL_ROI_X_LENGTH] = 1010;    /* i.e. 1042 - 33 + 1 */
roi[ASL_ROI_Y_LENGTH] = 787; ;   /* i.e.  788 -  2 + 1 */
```

```
    DIG16_set_active_area(Hdig16, roi);
```

BUGS / NOTES

Currently for area scan cameras *roi[ASL_ROI_X_START]* must be at least 5 and *roi[ASL_ROI_Y_START]* must be at least 1.

This function is not supported in data stream mode.

SEE ALSO

DIG16_get_active_area, *DIG16_set_ROI*, *DIG16_get_ROI_max*, *DIG16_set_ROI_rounding*.

DIG16_set_alignment

USAGE

Terr DIG16_set_alignment(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required mode.

DESCRIPTION

This function controls how the data processed by the Snapper-DIG16 is aligned before being transferred to host memory. For many applications this function need not be called directly, because *DIG16_initialize* calls it with *mode* set to *DIG16_ALIGN_LSB*.

If the processed data is between 9 and 15 bits per pixel, each 16 bit processed pixel will contain some unused bits. Most image processing software will expect these unused bits to be at the high end of the pixel, ie the processed data is LSB aligned.

However by shifting the data such that the unused bits are at the low end of the processed pixel, a fully saturated image would appear to have nominally the same brightness irrespective of the depth of the camera output, ie the processed data is MSB aligned. This mode can be useful when viewing an image from a number of different cameras on the same display using custom imaging software. (Note that whilst the TMG libraries expect the processed data to be LSB aligned, they automatically MSB align the data for display).

As there are no unused bits within an 8 or 16 bit processed pixel, the alignment setting is ignored under these conditions.

MODE

DIG16_ALIGN_LSB	The LSB of the processed pixel is aligned to be always on D0.
DIG16_ALIGN_MSB	The MSB of the processed pixel is aligned to be always on D15.
DIG16_ALIGN_NONE	The data from the camera is passed directly to the output without any alignment being performed.

The alignment requested is performed by programming a shift operation in the LUT the next time *DIG16_set_LUTs* is called. The size of the shift is calculated from the values of *msb* and *lsb* set by *DIG16_set_camera_info*. The requested alignment can be overridden by the *scaling* parameter of *DIG16_set_LUTs*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.

EXAMPLES

See *DIG16_set_LUTs* for example code.

BUGS / NOTES

There are no known bugs.

The modes *DIG16_ALIGN_LSB* and *DIG16_ALIGN_MSB* can be set regardless of whether the physical wiring of the camera to Snapper-DIG16 is MSB or LSB aligned, i.e. an MSB aligned camera can be read into system memory either MSB or LSB aligned, and with no affect on system performance.

The TMG library functions expect a LSB aligned image, so this is the recommended format. Certain TMG functions, such as *TMG_write_raw_data_file*, can be used with any alignment.

SEE ALSO

-

DIG16_set_callback

USAGE

*Terr DIG16_set_callback(Thandle Hdig16, Tparam mode, void (EXPORT_FN *callback)(Thandle, ui32, void*), void *parameter)*

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required callback mode.
callback	The callback function to install.
parameter	Pointer to optional user defined parameter to pass to the installed callback function. This would typically be a pointer to a user defined structure.

DESCRIPTION

This function installs a callback function that is called on certain events or interrupts which have been set up by *DIG16_set_interrupts*.

TYPE

DIG16_CALLBACK_INIT	Callback functions are disabled. *callback* and *parameter* should be passed as *NULL*. This is the default as set by *DIG16_initialize*.
DIG16_CALLBACK_SET	Function *callback* together with parameter *parameter* are installed as the callback function.

CALLBACK FUNCTION DEFINITION

*void function(Thandle Hdig16, ui32 int_source, void *parameter)*

Hdig16	Handle to Snapper-DIG16. The library sets this to the Snapper-DIG16 which caused the interrupt (i.e. there may be more than one Snapper-DIG16 in the system).
int_source	The interrupt source which has interrupted. The library sets this using the #define parameters as defined in *DIG16_set_interrupts*. These parameters are tested using bitwise operators.
parameter	The library sets this to the user defined parameter *parameter*, as set by *DIG16_set_callback*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.

EXAMPLES

This example shows how to enable interrupts and set up a callback to save a file on completion of image acquisition (see also the example under *DIG16_set_interrupts*):

```
/* Callback prototype */
void EXPORT_FN MyCallback(Thandle, ui32, void*);

/* The interrupt handler - which saves a file in this example */
```

```
void EXPORT_FN MyCallback(Thandle hSnapper, ui32 dwIntSrc, void *pMyData)
{
   /* Test on the correct interrupt */
   if (!(dwIntSrc & DIG16_INT_MAIN_TRANSFER_COMPLETE))
      return;  /* Unexpected interrupt - we could set an error flag here */

   /* Complete the acquisition and get the capture status */
   if (DIG16_get_capture_status(hSnapper, DIG16_START_AND_WAIT) != ASL_OK)
   {
/* This code assumes that the TMG image format of hImage is suitable
 * for saving as a TIFF file.  Also note that, for this example,  hImage
 * is a global.
       */
      TMG_image_set_outfilename(hImage, (char*) pMyData);
      TMG_image_write(hImage, NULL, TMG_TIFF, TMG_RUN);
   }
}


/* The following initialisation code would go into the main program thread */

/* We pass the name of the file in the optional parameter */
static char *pMyData = "test.tif";

/* Enable the callback function */
DIG16_set_callback(hSnapper, DIG16_CALLBACK_SET, MyCallback, NULL);

/* Enable transfer complete interrupt, so that the acquired image will
        * automatically get saved as a TIFF file.
 */
DIG16_set_interrupts(hSnapper, DIG16_INT_MAIN_TRANSFER_COMPLETE, TRUE);
```

BUGS / NOTES

There are no known bugs.

This function is not supported under MS-DOS , Windows 3.1, Windows 95 or Windows 98 operating systems.

SEE ALSO

DIG16_set_interrupts.

DIG16_set_camera_info

USAGE

Terr DIG16_set_camera_info(Thandle Hdig16, Tparam mode, ui8 lsb, ui8 msb)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required camera settings.
lsb	Bit position of the LSB from camera.
msb	Bit position of the MSB from camera.

DESCRIPTION

This function informs Snapper-DIG16 of general information about the camera or video source in use. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the appropriate parameters for the camera in use (see the supplied source to "dig16ini.c").

MODE

DIG16_CAMERA_CONTINUOUS_CLK	The camera outputs a clock for more than the period when data is being transferred. This mode should be set if there are four or more clocks from the camera following the last line of data in an image. If it is not clear from the camera's data sheet whether to set this parameter or *DIG16_CAMERA_DATA_CLK*, set this parameter. Change to *DIG16_CAMERA_DATA_CLK* if a data timeout occurs on attempting to read the full ROI.
*DIG16_CAMERA_*DATA_CLK	The camera only outputs a clock while data is being transferred. This mode should be set if there are less than four clocks from the camera following the last line of data in an image.
DIG16_CAMERA_INTERLACED	An interlaced video source is connected.
DIG16_CAMERA_NONINTERLACED	A non-interlaced or line scan video source is connected.
DIG16_CAMERA_TWO_CHANNEL	The video source connected has two data channels, i.e. it outputs two consecutive pixels in parallel. In this mode both channels must be 8 bit, with one channel on MSB ... MSB-7 and the other on MSB-8 ... MSB-15. Therefore the only acceptable values for MSB and LSB are 0 and 7, or 7 and 0. Channel order: Pixel t_n should be connected to MSB-8 ... MSB-15 and pixel t_{n+1} to MSB ... MSB-7.

MSB

This parameter is used by the hardware to know which bit is the MSB. For standard MSB aligned AIA cameras a setting of 0 should be used indicating that the MSB from the camera connects to the MSB pin on Snapper-DIG16.

LSB

This parameter is used by the hardware to know which bit is the LSB. As an example, for standard MSB aligned AIA cameras if the LSB is set to 11 then pins MSB..MSB-11 contain camera data, with MSB-11 the LSB; and pins MSB-12 to MSB-15 can be ignored.

Cameras such as the Kodak Megaplus 1.4 connected via cable CBL-68-37D-A-2M, results in LSB justified data needing to set *lsb* to a lower value than *msb*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other. See the examples given below. No change is made to the setting of the board.
ASLERR_OUT_OF_RANGE	*lsb* or *msb* is more than 15, or the larger of *msb* and *lsb* is not between 7 and 15, or the camera data width (i.e. the difference between *msb* and *lsb* + 1) is not between 2 and 16.
ASLERR_NOT_IMPLEMENTED	*DIG16_CAMERA_TWO_CHANNEL* has been selected with either *msb* or *lsb* not 0 or 7.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

The camera in use is non-interlaced, with it's MSB on pin MSB, and it's LSB on pin MSB-09 (i.e. it is MSB aligned and has 10 data bits), and does not stop the clock as soon as data transfer is complete:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED |
       DIG16_CAMERA_CONTINUOUS_CLK, 9, 0);
```

The camera in use is interlaced, with it's MSB on pin MSB-13, and it's LSB on pin MSB-00 (i.e. it is LSB aligned and has 14 data bits), and does not stop the clock as soon as data transfer is complete:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_INTERLACED |
       DIG16_CAMERA_CONTINUOUS_CLK, 0, 13);
```

The camera in use is non-interlaced, with it's MSB on pin MSB-11, and it's LSB on pin MSB-02 (i.e. it is LSB aligned, but with a 2 bit offset, and has 10 data bits), and stops the clock as soon as data transfer is complete:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_DATA_CLK |
       DIG16_CAMERA_NONINTERLACED, 2, 11);
```

The camera in use is two channel, non-interlaced, with one channel's MSB on pin MSB and LSB on pin MSB-07 (i.e. it is MSB aligned), and the second channel's MSB on pin MSB-08 and LSB on pin MSB-15, and does not stop the clock as soon as data transfer is complete:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED |
       DIG16_CAMERA_TWO_CHANNEL | DIG16_CAMERA_CONTINUOUS_CLK, 7, 0);
```

The following will result in a error *ASL_PARAM_CONFLICT* because it is not possible for a camera to be both interlaced and non-interlaced:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_INTERLACED |
       DIG16_CAMERA_NONINTERLACED, 0, 7);
```

BUGS / NOTES

There are no known bugs.

The function calculates the data width of the camera from the difference between the setting of *msb* and *lsb*. For MSB aligned cameras *MSB* must be 0.

SEE ALSO

DIG16_get_camera_LSB, *DIG16_get_camera_MSB*, *DIG16_get_data_width*, *DIG16_get_parameter*.

DIG16_set_capture

USAGE

Terr DIG16_set_capture(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required capture mode.

DESCRIPTION

This function configures Snapper-DIG16 for different types of image capture. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the parameter *DIG16_CAPTURE_INIT*, along with the appropriate field/frame capture mode for the camera in use.

INITIALIZE

DIG16_CAPTURE_INIT	This selects next frame, x1 subsample, and disables trigger.

SUB-SAMPLE CONTROL (AREA SCAN MODE)

There are 5 different sub-sample ratios, which also control whether a field or frame is captured:

DIG16_SUB_X1	This enables full resolution image capture of a complete frame.
DIG16_SUB_X1_SINGLE_FIELD	This enables full resolution image capture of a single field. Note that because the resulting image only contains one field it will have the wrong aspect ratio - it will be half height. This mode can only be used for an interlaced source.
DIG16_SUB_X2	This enables sub-sampled by 2 image capture. For non-interlaced sources this is achieved by capturing every other pixel in the horizontal direction and every other line in the vertical direction. Similarly, for interlaced sources every other pixel in the horizontal direction and only one field in the vertical direction is captured.
DIG16_SUB_X4	This enables sub-sampled by 4 image capture. For non-interlaced sources this is achieved by capturing every fourth pixel in the horizontal direction and every fourth line in the vertical direction. Similarly, for interlaced sources every fourth pixel in the horizontal direction and every other line of one field in the vertical direction is captured.
DIG16_SUB_X8	This enables sub-sampled by 8 image capture. For non-interlaced sources this is achieved by capturing every eighth pixel in the horizontal direction and every eighth line in the vertical direction. Similarly, for interlaced sources every eighth pixel in the horizontal direction and every fourth line of one field in the vertical direction is captured.

Note: If the subsample ratio is changed this function calls *DIG16_set_ROI* with the current ROI. This allows the ROI to be checked and if necessary adjusted so that it complies with the settings given by *DIG16_set_ROI_rounding*. Therefore if an application keeps its own copy of the current ROI is should call *DIG16_get_ROI* to update its copy after calling *DIG16_get_ROI*.

SUB-SAMPLE CONTROL (LINE SCAN MODE)

There are 4 different sub-sample ratios which control sub-sampling of data across the line:

DIG16_SUB_X1	This enables full resolution image capture of a complete line.
DIG16_SUB_X2	This enables sub-sampled by 2 image capture. This is achieved by capturing every other pixel in the line.
DIG16_SUB_X4	This enables sub-sampled by 4 image capture. This is achieved by capturing every fourth pixel in the line.
DIG16_SUB_X8	This enables sub-sampled by 8 image capture. This is achieved by capturing every eighth pixel in the line.

Note that the function *DIG16_set_linescan_ctrl* controls the line acceptance ratio independently from the sub-sample ratio.

Note: If the subsample ratio is changed this function calls *DIG16_set_ROI* with the current ROI. This allows the ROI to be checked and if necessary adjusted so that it complies with the settings given by *DIG16_set_ROI_rounding*. Therefore if an application keeps its own copy of the current ROI is should call *DIG16_get_ROI* to update its copy after calling *DIG16_set_capture*.

SUB-SAMPLE CONTROL (DATA STREAM MODE)

There are 4 different sub-sample ratios which control sub-sampling of the raw data stream:

DIG16_SUB_X1	This enables full resolution image capture.
DIG16_SUB_X2	This enables sub-sampled by 2 image capture. This is achieved by capturing every other pixel in the data stream.
DIG16_SUB_X4	This enables sub-sampled by 4 image capture. This is achieved by capturing every fourth pixel in the data stream.
DIG16_SUB_X8	This enables sub-sampled by 8 image capture. This is achieved by capturing every eighth pixel in the data stream.

INITIAL FIELD CONTROL

There are 3 different field control modes for use with interlaced area scan cameras:

DIG16_START_1ST_FIELD	Capture starts with a field 1. If a frame is being captured Snapper-DIG16 will wait for a field 1, and then capture this field 1 and the following field 2. If a field is being captured Snapper-DIG16 will wait for a field 1, and then only capture this field 1.
DIG16_START_2ND_FIELD	Capture starts with a field 2. If a frame is being captured Snapper-DIG16 will wait for a field 2, and then capture this field 2 and the following field 1. If a field is being captured Snapper-DIG16 will wait for a field 2, and then only capture this field 2.
DIG16_START_NEXT_FIELD	Capture starts with the next field, either field 1 or field 2. If a frame is being captured Snapper-DIG16 will capture the next field and the following field - this may be field 1 and field 2 of the same frame, or field 2 of one frame and field 1 of the following frame. In either case the frame will be correctly deinterlaced. If a field is being captured Snapper-DIG16 will only capture the next field - either field 1 or field 2.

SEQUENCE CONTROL

There are 2 different operating modes:

DIG16_SINGLE_CAPTURE_MODE	After completing an acquisition of the required amount, ie a single frame in area scan or n pixels in data stream modes, the hardware resets and does not attempt to acquire any more data.

DIG16_SEQUENCE_MODE	In this mode, the hardware will continue to acquire valid data until *DIG16_capture_to_image* is called with *DIG16_CAPTURE_END*. A single *DIG16_capture_to_image* is used to start the acquisition and then all subsequent data is read with calls to *DIG16_read_video_data*. Because there is no software intervention to re-arm the hardware between each field, the acquisition rate can be faster but there is the possibility that if the software processing takes too long then the FIFO may overflow. This mode is not yet supported in data stream.

TRIGGER CONTROL

Capture can be delayed until an external capture trigger event happens:

DIG16_TRIG_IN_ENABLE	Image capture requested by *DIG16_capture* will not start until an active edge of the selected trigger occurs. See *DIG16_set_trigger* for information on control of the trigger source. Note that if the trigger rate is slow, the timeout before capture may need to be increased to prevent *DIG16_capture_to_image* timing out before the trigger is received. This timeout is set by *DIG16_set_parameter*.
DIG16_TRIG_IN_DISABLE	Image capture requested by *DIG16_capture_to_image* will occur independent of the trigger status.

Note that if the trigger is infrequent then the capture timeout parameter *DIG16_TIMEOUT_BEFORE_CAPTURE* will need to be extended - see *DIG16_set_parameter*.

PARAMETER INTERACTION

When combinations of parameters are passed in one call to *DIG16_set_capture* the routine will interpret the combinations in a 'sensible' way whenever possible, or return an error if the combinations are invalid. See the examples given below.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other. See the examples given below. No change is made to the setting of the capture circuits.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_INCOMPATIBLE	A field capture mode is requested for a non-interlaced camera.

EXAMPLES

The following code will capture the next first field using 2 times sub-sampling:

```
DIG16_set_capture(Hdig16, DIG16_START_1ST_FIELD | DIG16_SUB_X2);
DIG16_capture_to_image(Hdig16, Himage, DIG16_START_AND_WAIT);
```

In the following call a subsample factor of 8 and capturing the next first field will override the default settings of *DIG16_CLK_INIT*:

```
DIG16_set_capture(Hdig16, DIG16_START_1ST_FIELD | DIG16_CAPTURE_INIT |
       DIG16_SUB_X8);
```

The following call will result in a *DIG16_PARAM_CONFLICT* error because Snapper-DIG16 cannot capture at both x2 and x4 sub-sampling at the same time:

```
DIG16_set_capture(Hdig16, DIG16_SUB_X4 | DIG16_SUB_X2);
```

BUGS / NOTES

There are no known bugs.

The function *DIG16_set_camera_info* informs the software whether the video source is interlaced or non-interlaced.

SEE ALSO

DIG16_capture_to_image, *DIG16_set_trigger*.

DIG16_set_clk

USAGE

*Terr DIG16_set_clk(Thandle Hdig16, Tparam mode, ui16 *frequency)*

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required clock mode.
frequency	Required frequency in kilohertz of internal clock generator.

DESCRIPTION

This function controls Snapper-DIG16's external clock line (PCLK). The mode parameter is formed by 'OR'ing the required options from the list defined below. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the appropriate parameter for the camera in use.

MODE

DIG16_CLK_IN_POS	This selects the rising edge of the external PCLK signal as Snapper-DIG16's pixel clock.
DIG16_CLK_IN_NEG	This selects the falling edge of the external PCLK signal as Snapper-DIG16's pixel clock.
DIG16_CLK_OUT_POS	This selects the internal clock generator as Snapper-DIG16's pixel clock, and drives the external PCLK signal (area scan mode) or the output B signal (line scan mode) with that clock.
DIG16_CLK_OUT_NEG	This selects the internal clock generator as Snapper-DIG16's pixel clock, and drives the external PCLK signal (area scan mode) or the output B signal (line scan mode) with that clock inverted.
DIG16_CLK_IO_POS	This drives the output B signal with the output from the internal clock generator, and selects the rising edge of the external PCLK signal as Snapper-DIG16's pixel clock. It is needed for cameras which must be supplied with a clock, but also provide a clock which should be used to sample camera data. Note that there is no requirement that the input and output clocks have a known phase relationship - they may be completely different frequencies.
DIG16_CLK_IO_NEG	This drives the output B signal with the output from the internal clock generator, and selects the falling edge of the external PCLK signal as Snapper-DIG16's pixel clock. It is needed for cameras which must be supplied with a clock, but also provide a clock which should be used to sample camera data. Note that there is no requirement that the input and output clocks have a known phase relationship - they may be completely different frequencies.
DIG16_CLK_OUT_FREQUENCY	This sets the frequency of Snapper-DIG16's internal clock generator to the nearest achievable value to that requested with the parameter *frequency*. With a standard Snapper-DIG16 the following frequencies are supported by the hardware: 251 kHz, 501 kHz, 1.003 MHz , 2.00 MHz, 2.50 MHz, 3.133 MHz, 4.00 MHz, 5.00 MHz, 6.265 MHz, 8.00 MHz, 10.00 MHz, 12.53 MHz, 16.00 MHz, 20.00 MHz, 25.06 MHz, 33.29MHz, 40MHz and 50.11MHz. The parameter *frequency* is updated with the frequency actually set. Note that RS-422 is only specified up to 10 MHz, so higher frequencies must be used with

caution, and short cables will probably be needed. The frequencies over 25MHz are provided for those cameras which must be supplied with a clock but also output a clock (usually half rate) which is used to sample camera data.

The parameter *frequency* is only used in mode *DIG16_CLK_OUT_FREQUENCY*. For all other modes it should be set to *NULL*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

Set the falling edge of incoming pixel clock as the active edge:

```
DIG16_set_clk(Hdig16, DIG16_CLK_IN_NEG, NULL);
```

Use the internal clock generator at 12.5 MHz, and drive it out inverted:

```
ui16 frequency = 12500;

DIG16_set_clk(Hdig16, DIG16_CLK_OUT_NEG | DIG16_CLK_OUT_FREQUENCY,
       &frequency);
/* The following will print 12530, which is the nearest achievable frequency
       to 12500 requested */
printf("\nFrequency set is %d", frequency);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

-

DIG16_set_comms

USAGE

Terr DIG16_set_comms(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required comms mode.

DESCRIPTION

This function provides control of the serial communication hardware drivers. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the appropriate parameter for the camera in use.

The mode parameter should be one of the following:

MODE

DIG16_COMMS_DEFAULT	This should be selected for cameras which do not use serial communications. In this mode, the serial communications are set to the same mode as the jumper J4 on the Snapper-DIG16 or J2 on Snapper-PMC-DIG16. (see the ***Snapper! Installation Guide*** for details).
DIG16_COMMS_RS422	This should be selected for cameras which use the RS-422 communication standard. Note that jumper J4 on the Snapper-DIG16 or jumper J2 on Snapper-PMC-DIG16 must be set for RS-422. (see the ***Snapper! Installation Guide*** for details).
DIG16_COMMS_RS232	This should be selected for cameras which use the RS-232 communication standard. Note that jumper J4 on the Snapper-DIG16 or jumper J2 on Snapper-PMC-DIG16 must be set for RS-232. (see the ***Snapper! Installation Guide*** for details). The RS-232 output and input signals (as viewed by Snapper-DIG16) are available on pins 23 and 22 respectively of the front panel connector.
DIG16_COMMS_RS232_ON_NEG	This mode is similar to *DIG16_COMMS_RS232*, except that the RS-232 output and input signals (as viewed by Snapper-DIG16) are available on pins 57 and 56 respectively of the front panel connector. This parameter is currently only supported in area scan mode.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_INCOMPATIBLE	The setting of jumper J2 or J4 (see above) does not match the requested option. Note that this error can only be generated on recent boards which support readback of the jumper setting.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.

EXAMPLES

To use RS-422 interface levels:

```
DIG16_set_comms(Hdig16, DIG16_COMMS_RS422);
```

BUGS / NOTES

There are no known bugs.

IMPORTANT: Check the specification of the camera before deciding on the setting of this function.

SEE ALSO

-

DIG16_set_ctrl_io

USAGE

Terr DIG16_set_ctrl_io(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required control I/O mode.

DESCRIPTION

This function provides direct control of the general purpose I/O lines. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the parameter *DIG16_CTRLOUT_INIT*, and camera specific functions such as *DIG16_set_mplus_ctrl* provide a higher level control of the I/O lines.

The mode parameter should be a combination of the following:

MODE

DIG16_CTRL_INIT	This sets the I/O lines to inputs and tri-state disables the outputs. If enabled, all outputs would be ‘0’.
DIG16_CTRL_IO_A_IN	This sets I/O line A as an input.
DIG16_CTRL_IO_A_OUT_0	This drives I/O line A as a low output.
DIG16_CTRL_IO_A_OUT_1	This drives I/O line A as a high output.
DIG16_CTRL_IO_B_IN	This sets I/O line B as an input.
DIG16_CTRL_IO_B_OUT_0	This drives I/O line B as a low output.
DIG16_CTRL_IO_B_OUT_1	This drives I/O line B as a high output.
DIG16_CTRL_IO_C_IN	This sets I/O line C as an input.
DIG16_CTRL_IO_C_OUT_0	This drives I/O line C as a low output.
DIG16_CTRL_IO_C_OUT_1	This drives I/O line C as a high output.
DIG16_CTRL_IO_D_IN	This sets I/O line D as an input.
DIG16_CTRL_IO_D_OUT_0	This drives I/O line D as a low output.
DIG16_CTRL_IO_D_OUT_1	This drives I/O line D as a high output.
DIG16_CTRL_OUT_A_0	This sets output line A low. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_A_1	This sets output line A high. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_A_EXP_POS	This drives output line A with the exposure signal from the baseboard’s timer, with an active high pulse when the timer is in monostable mode. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_A_EXP_NEG	This drives output line A with the exposure signal from the baseboard’s timer, with an active low pulse when the timer is in monostable mode. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.

DIG16_CTRL_OUT_B_0	Area scan only: This sets output line B low. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_B_1	Area scan only: This sets output line B high. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_C_0	Area scan only: This sets output line C low. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_C_1	Area scan only: This sets output line C high. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_D_0	This sets output line D low. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_D_1	This sets output line D high. The line is tri-state controllable using *DIG16_CTRL_OUT_EN* and *DIG16_CTRL_OUT_DIS*.
DIG16_CTRL_OUT_EN	This tri-state enables the four control out lines, A to D.
DIG16_CTRL_OUT_DIS	Area scan only: This tri-state disables the four control out lines, A to D.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.

EXAMPLES

To set the I/O A high, I/O C an input, and output A high, and pulse I/O B low.

```
DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_IO_A_1 | DIG16_CTRL_IO_B_1 |
      DIG16_OUT_A_1 | DIG16_CTRL_IO_C_IN | DIG16_CTRL_OUT_EN);
DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_IO_B_0);
DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_IO_B_1);
```

The following will result in a parameter conflict error because I/O A cannot be an input and an output at the same time:

```
DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_IO_A_1 | DIG16_CTRL_IO_A_IN);
```

BUGS / NOTES

There are no known bugs.

Note that all output values are written to hardware simultaneously, regardless of the order of parameters in the function call.

In line scan mode output lines B and C are dedicated for camera control functions, therefore direct control of these lines is not possible. Also note that in line scan mode it is not possible to tri-state output A and output D because the dedicated lines B and C must be driven, and independent control is not available.

IMPORTANT: I/O pins should not be enabled as outputs without checking with the camera cable in use to ensure that the pin will not clash with an output from the camera.

SEE ALSO

DIG16_set_timer, also camera specific functions - see the end of the manual.

DIG16_set_data_stream_ctrl

USAGE

Terr DIG16_set_data_stream_ctrl(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required data stream mode

DESCRIPTION

This function controls parameters needed for data stream cameras. For many applications this function need not be called because *DIG16_initialize* calls it with the required parameters for the selected camera.

MODE

DIG16_DSTRM_LINE_ACQ_ENABLE	Data is only acquired while the line enable signal is asserted.
DIG16_DSTRM_LINE_ACQ_DISABLE	Data is only acquired regardless of the line enable signal.
DIG16_DSTRM_FRAME_ACQ_ENABLE	Data is only acquired while the frame enable signal is asserted.
DIG16_DSTRM_FRAME_ACQ_DISABLE	Data is only acquired regardless of the frame enable signal.
DIG16_DSTRM_LINE_START_ENABLE	Start of data capture occurs on the edge of the line enable signal when it becomes asserted.
DIG16_DSTRM_LINE_START_DISABLE	Start of data capture occurs regardless of the line enable signal.
DIG16_DSTRM_FRAME_START_ENABLE	Start of data capture occurs on the edge of the frame enable signal when it becomes asserted.
DIG16_DSTRM_FRAME_START_DISABLE	Start of data capture occurs regardless of the frame enable signal.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.

EXAMPLES

For a camera with line and frame enables, the normal setting would be to use both line and frame enable to qualify both capture start and data acquisition. In this mode, capture would not start until both frame and line had been detected low together, and then pixels would be acquired while frame and line were simultaneously high:

```
DIG16_set_data_stream_ctrl(Hdig16, DIG16_DSTRM_LINE_ACQ_ENABLE |
       DIG16_DSTRM_FRAME_ACQ_ENABLE | DIG16_DSTRM_LINE_START_ENABLE |
       DIG16_DSTRM_FRAME_START_ENABLE);
```

Similarly, for a camera with only a line enable, the normal setting would be to use the line enable to qualify both capture start and data acquisition. The frame controls should be disabled unless the Snapper-DIG16 frame enable signal is tied to a known level:

```
DIG16_set_data_stream_ctrl(Hdig16, DIG16_DSTRM_LINE_ACQ_ENABLE |
       DIG16_DSTRM_FRAME_ACQ_DISABLE | DIG16_DSTRM_LINE_START_ENABLE |
       DIG16_DSTRM_FRAME_START_DISABLE);
```

BUGS / NOTES

There are no known bugs.

This function is not supported in area scan or line scan modes.

SEE ALSO

-

DIG16_set_format

USAGE

Terr DIG16_set_format(Thandle Hdig16, Tparam snap_format, ui16 TMG_format)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
snap_format	Required format of Snapper.
TMG_format	Required TMG format of resulting Himage.

DESCRIPTION

This function controls the format in which data is stored in both Snapper-DIG16 video memory and in the host computer's memory. *DIG16_initialize* calls it with the appropriate parameter for the camera in use.

The parameter *TMG_format* should be a TMG library pixel format (see the TMG Programmer's Manual).

SNAP_FORMAT

The parameter *snap_format* should be one of the following:

DIG16_FORMAT_Y16	Video data will be read from the Snapper-DIG16 as 16 bit words. This mode can be used with cameras which have between 9 and 16 bits of valid data, based on the values of *lsb* and *msb* set by *DIG16_set_camera_info*. This will give images with the maximum image quality when used in conjunction with *TMG_Y16*. Cameras with 8 data bits should always use *DIG16_FORMAT_Y8*.
DIG16_FORMAT_Y8	Video data will be read from the Snapper-DIG16 as 8 bit words. This mode can be used with all cameras. For cameras which give between 9 and 16 bits of valid data, based on the values of *lsb* and *msb* set by *DIG16_set_camera_info*, the upper 8 bits will be transferred (as default). This will result in reduced image quality, but allows twice the data transfer rate compared to Y16.

If colour output modes such as *TMG_RGB16* are required the *snap_format* should always be set to *DIG16_FORMAT_Y8*. This is because none of the TMG colour modes use more than 8 bits per colour, so there would be no change in quality in selecting *DIG16_FORMAT_Y16,* and using *DIG16_FORMAT_Y8* allows a faster transfer rate.

Many combinations of *snap_format* and *TMG_format* are only supported on baseboards with a data mapper, because it is the data mapper which performs the necessary format conversions. Even with a data mapper some combinations are not supported, for instance current data mappers do not support colour space conversion, so *TMG_YUV422* cannot be used as an output format. When a combination is not supported this function returns an error, and a *TMG_image_convert* call could be made to perform the conversion in software.

Note that while selecting an output mode such as *TMG_RGBX32* to allow direct copying of data to a display card may allow faster display update, it also quadruples the bandwidth required on the host computer's bus relative to using *TMG_Y8*. With slow computer busses, or with digital cameras with a fast clock, this may result in a FIFO overflow on Snapper-DIG16. If this happens try using *TMG_Y8* format and then doing the format conversion using *TMG_image_convert*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

ASLERR_PARAM_CONFLICT	More than one *snap_format* parameter has been passed in, which is not allowed because the parameters are mutually exclusive.
ASLERR_NOT_SUPPORTED	The combination of parameters selected is not supported by the baseboard. Either the baseboard may not have a data mapper, or the format conversion required is too complex for the data mapper to perform.

EXAMPLES

The following code could be used with a 12 bit camera to transfer all 12 bits of data to an Himage:

```
DIG16_set_alignment(Hdig16, DIG16_ALIGN_LSB);
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 11, 0);
...
DIG16_set_format(Hdig16, DIG16_FORMAT_Y16, TMG_Y16);
```

The following code could be used with the same 12 bit camera to generate an output of the format *TMG_RGB16* which is suitable for rapid display using both MS-DOS and Microsoft Windows. This is done by firstly the Snapper-DIG16 mapping D11..D4 to D7..D0 using its LUTs, and secondly the data mapper duplicating this data into its red, green and blue channels, and finally trimming the resulting 24 bit value down to 16 bits by discarding the lower bits:

```
DIG16_set_format(Hdig16, DIG16_FORMAT_Y8, TMG_RGB16);
```

BUGS / NOTES

Because some of the mapping functions are performed by the LUTs on the Snapper-DIG16, if *snap_format* has been changed the function *DIG16_set_LUTs* must be called after calling *DIG16_set_format* and before calling *DIG16_capture_to_image*. If only *TMG_format* has changed there is no need to call *DIG16_set_LUTs*.

SEE ALSO

DIG16_set_LUTs.

DIG16_set_image

USAGE

Terr DIG16_set_image(Thandle Hdig16, Thandle Himage)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
Himage	Handle to image.

DESCRIPTION

This function is used to initialize the image structure into which the captured image will be stored. It sets the image size based upon the ROI (Region of Interest), the sub-sample ratio, and the image type based on the requested format. It also initializes the image to process the video data in one strip as is required for the Snapper-DIG16 (see the TMG Programmer's Manual for an explanation of strip processing).

This function must be called after *DIG16_initialize* has been called, or after the width or height of the ROI, the sub-sample ratio, or the image format or image data width has changed, but before *DIG16_capture_to_image* is called.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_NOT_IMPLEMENTED	The Snapper-DIG16 sub-sample ratio is not recognised.

EXAMPLE

```
/* Change the ROI ... */
DIG16_set_ROI(Hdig16, DIG16_ROI_SET, roi);
/* ... therefore call set image before capturing ... */
DIG16_set_image(Hdig16, Himage);
/* ... finally acquire the image */
DIG16_capture_to_image(Hdig16, Himage, DIG16_START_AND_WAIT);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

-

DIG16_set_image_data_width

USAGE

Terr DIG16_set_image_data_width(Thandle Hdig16, ui16 width)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
Width	Data width of image.

DESCRIPTION

This function is used to inform the library of the data width in bits of the image structure into which the captured image will be stored. For many applications this function need not be called directly, because *DIG16_initialize* calls it with *width* set to the value of *required_bits*.

The function may need to be called after *DIG16_set_LUTs* has been called, depending on how the LUT is set up. See *DIG16_set_LUTs* for more information.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_OUT_OF_RANGE	The value of *width* is less than 2 or greater than 16.

EXAMPLE

```
/* Set new LUT which generates 9 bit data */
DIG16_set_LUTs(Hdig16, DIG16_LUT_SET, 0, Plut);
/* Therefore update data width */
DIG16_set_image_data_width(Hdig16, 9);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_set_LUTs

DIG16_set_interrupts

USAGE

Terr DIG16_set_interrupts(Thandle Hdig16, Tparam type, Tboolean flag)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
type	Required interrupt to control.
flag	*TRUE* / *FALSE* setting for requested interrupt.

DESCRIPTION

This function controls the generation of events and interrupts from Snapper-DIG16. When an enabled interrupt occurs the function installed by *DIG16_set_callback* is called. Multiple interrupts are set by bitwise ORing the interrupt options in a single call to *DIG16_set_interrupts*.

TYPE

DIG16_INT_INIT	All interrupts are disabled. The flag parameter is ignored. This is the default set by *DIG16_initialize.*
DIG16_INT_MAIN_TRANSFER_COMPLETE	When enabled, an interrupt is generated when all data transfer has completed.
DIG16_INT_CAPTURE_COMPLETE	When enabled, an interrupt is generated when the Snapper hardware has finished acquiring data from the camera. This will be slightly before *DIG16_INT_MAIN_TRANSFER_COMPLETE* as image data will still be in the hardware FIFO and bus interface etc.
DIG16_INT_START_OF_FRAME	When enabled, an interrupt is generated at the start of the frame.
DIG16_INT_END_OF_LINE	When enabled, an interrupt is generated at the end of a line.
DIG16_INT_ACQ_TRIGGER	When enabled, an interrupt is generated when the acquisition trigger is asserted.
DIG16_INT_FIFO_OVERFLOW	When enabled, an interrupt is generated if the FIFO overflows.
DIG16_INT_INITIAL_DATA	When enabled, an interrupt is generated when initial data first enters Snapper-DIG16. This is likely to be just after *DIG16_INT_START_OF_FRAME.*

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The type parameter is invalid.
ASLERR_NOT_SUPPORTED	A mode other than *DIG16_INT_INIT* has been selected under an operating system which does not support interrupts.

EXAMPLES

This example shows how to perform interrupt driven acquisition and processing using two buffers in host memory. The benefit in making the acquisition interrupt driven is that CPU's processing time is maximised.

The program's main thread installs the interrupt callback function as described in the example under *DIG16_set_callback* and starts acquisition as shown below. Full error checking is not shown for simplicity.

```
/* Setup Snapper-DIG16 as appropriate */
…
/* Prepare image2 for processing by capturing an image */
DIG16_capture_to_image(D16.m_hSnapper, D16.m_hSrcImage2,
       DIG16_START_AND_WAIT);

/* Install the interrupt handler and enabled the appropriate interrupt */
DIG16_set_callback(hSnapper, DIG16_CALLBACK_SET, InterruptHandler, NULL);
DIG16_set_interrupts(hSnapper, DIG16_INT_TRANSFER_COMPLETE, TRUE);

/* Start interrupt driven acquisition */
DIG16_capture_to_image(hSnapper, hImage1, DIG16_START_AND_RETURN);

/* This main program thread now sleeps/waits or returns to the message loop */
```

The interrupt handler now performs acquisition completion, starts the next acquisition and processes the current image:

```
/* This is our interrupt handler */
void InterruptHandler(Thandle hSnapper, ui32 dwIntSrc, void *pMyData)
{
   static ui32 dwBuffer = 1;

   /* Complete the read - if acquisition has failed then switch off interrupts
    * and finish.
    */
   if (DIG16_get_capture_status(hSnapper, DIG16_START_AND_WAIT) != ASL_OK)
   {
      DIG16_set_callback(hSnapper, DIG16_CALLBACK_SET, NULL, NULL);
      return;
   }
   else if ( dwBuffer == 1 )
   {
      DIG16_capture_to_image(hSnapper, hImage2, DIG16_START_AND_RETURN);
      ProcessImage(hImage1);
      dwBuffer = 2;
   }
   else
   {
      /* Complete the read and start the next one */
      DIG16_get_capture_status(hSnapper, DIG16_START_AND_WAIT);
      DIG16_capture_to_image(hSnapper, hImage1, DIG16_START_AND_RETURN);
      ProcessImage(hImage2);
      dwBuffer = 1;
   }

   /* Test on a global (or whatever) to determine if we have finished */
   if (bFinish == TRUE)
      DIG16_set_callback(hSnapper, DIG16_CALLBACK_SET, NULL, NULL);
}
```

BUGS / NOTES

The interrupt *DIG16_INT_MAIN_TRANSFER_COMPLETE* is not generated when 384 bytes or less is acquired per frame. In this instance use the interrupt *DIG16_INT_CAPTURE_COMPLETE* instead.

If adding or removing interrupts, disable interrupts using *DIG16_set_callback* (to *NULL*) first and then re-enable after having called *DIG16_set_interrupts*.

This function is not supported under MS-DOS, Windows 3.1, Windows 95 or Windows 98 operating systems.

SEE ALSO

DIG16_set_callback.

DIG16_set_linescan_ctrl

USAGE

Terr DIG16_set_linescan_ctrl(Thandle Hdig16, Tparam mode, ui16 width)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required linescan mode.
width	Required width of line start out pulse in pixel clocks.

DESCRIPTION

This function controls parameters needed for line scan cameras. For many applications this function need not be called because *DIG16_initialize* calls it with the required parameters for the selected camera. Even if it is called it is only the line acceptance controls and line trigger controls which should be needed.

The line start controls should only be needed to set up a camera which *DIG16_initialize* does not support. Many cameras take a line start in signal and send it back as a line start out. This automatically compensates for any propagation delays in buffers and cabling. It is necessary to have technical information on the camera signal connections and timing to set these line start parameters correctly.

The *width* parameter is only used if the mode *DIG16_LSCAN_LSTART_OUT_WIDTH* is selected. If this mode is not selected then set *width* to zero.

MODE

DIG16_LSCAN_LINES_X1	Line acceptance ratio - all incoming lines are stored.
DIG16_LSCAN_LINES_X2	Line acceptance ratio - every other incoming line is stored.
DIG16_LSCAN_LINES_X4	Line acceptance ratio - 1 in 4 incoming lines are stored.
DIG16_LSCAN_LINES_X8	Line acceptance ratio - 1 in 8 incoming lines are stored.
DIG16_LSCAN_LSTART_IN_POS	An active high line start in pulse from the camera is used to start Snapper-DIG16 acquiring data. The line start in signal should be connected to the LINE_EN pin.
DIG16_LSCAN_LSTART_IN_NEG	An active low line start in pulse from the camera is used to start Snapper-DIG16 acquiring data. The line start in signal should be connected to the LINE_EN pin.
DIG16_LSCAN_LSTART_IN_NONE	The camera does not generate a line start in pulse, so the internal line start out pulse is used to start Snapper-DIG16 acquiring data. Note that this internal pulse is not affected by the setting of *DIG16_LSCAN_LSTART_OUT_POS*, *DIG16_LSCAN_LSTART_OUT_NEG*, or *DIG16_LSCAN_LSTART_OUT_NONE*.
DIG16_LSCAN_LSTART_OUT_POS	An active high line start out pulse will be generated to tell the camera to start sending data. The width of this pulse will be determined by the most recent call with the parameter *DIG16_LSCAN_LSTART_OUT_WIDTH*, and the pulse is generated is output on the OUT_C pin.
DIG16_LSCAN_LSTART_OUT_NEG	An active low line start pulse will be generated to tell the camera to start sending data. The width of this pulse will be determined by the most recent call with the parameter *DIG16_LSCAN_LSTART_OUT_WIDTH*, and the pulse is

	generated is output on the OUT_C pin.
DIG16_LSCAN_LSTART_OUT_NONE	No line start out pulse will be generated.
DIG16_LSCAN_LSTART_OUT_WIDTH	The width of the line start out pulse is set by the *width* parameter. This value is in pixel clocks.
DIG16_LSCAN_LTRIG_IN_TIMER	The line trigger, which is used to capture each line, is obtained from the baseboard timer. Each rising edge from the baseboard timer results in one line being captured. This mode would normally be used in conjunction with the astable mode of the timer, as controlled by *BASE_set_timer*. This allows lines to be captured on a regular interval.
DIG16_LSCAN_LTRIG_IN_POS	The line trigger, which is used to capture each line, is obtained from the line trigger input. Each rising edge of the line trigger input results in one line being captured.
DIG16_LSCAN_LTRIG_IN_NEG	The line trigger, which is used to capture each line, is obtained from the line trigger input. Each falling edge of the line trigger input results in one line being captured.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.

EXAMPLES

To set the line acceptance ratio to every other line:

```
DIG16_set_linescan_ctrl(Hdig16, DIG16_LSCAN_LINES_X1, 0);
```

To set up the Snapper-DIG16 for a camera which needs to receive an active high line start pulse 40 clocks wide, and which returns an active low line start pulse:

```
DIG16_set_linescan_ctrl(Hdig16, DIG16_LSCAN_LSTART_OUT_WIDTH, 40);
DIG16_set_linescan_ctrl(Hdig16, DIG16_LSCAN_LSTART_OUT_POS |
      DIG16_LSCAN_LSTART_IN_NEG, 0);
```

BUGS / NOTES

There are no known bugs.

This function is not supported in area scan or data stream modes.

The line trigger should be connected to pins FRAME+ and FRAME-. See the Camera Specific Release Notes in the Installation section of the manual for cable connections and Snapper-DIG16's pinout.

SEE ALSO

-

DIG16_set_LUTs

USAGE

*Terr DIG16_set_LUTs(Thandle Hdig16, Tparam mode, i16 scaling, IM_UI16 *Plut)*

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required mode.
scaling	Scaling factor for LUT.
Plut	Pointer to an array of ui16 elements for the required LUT.

DESCRIPTION

This function controls the look up tables (LUTs) which the video data is passed through. For many applications this function need not be called directly, because *DIG16_initialize* calls it with *scaling* set to zero and *mode* set to *DIG16_LUT_INIT*.

MODE

DIG16_LUT_INIT	This writes a linear ramp to the selected LUT(s), effectively bypassing it. The contents of *Plut* are ignored on entry, but are used to return the values written to the LUT by *DIG16_set_LUTs*.
DIG16_LUT_SET	This copies the values in *Plut* to the LUT.

In both cases the LUT is scaled as determined by *DIG16_set_alignment* and the parameter *scaling* - see below.

ALIGNMENT AND SCALING

The LUT is used to automatically scale data from the camera to have the required bit alignment in the resulting Himage, as determined by the requested alignment from *DIG16_set_alignment*. *DIG16_set_LUTs* does this based on the most recent calls to *DIG16_set_camera_info* and *DIG16_set_format*, as well as *DIG16_set_alignment*. If the contents of the LUT passed in override this bit alignment it is necessary to call *DIG16_set_image_data_width* to ensure than the resulting image is processed and displayed correctly.

If *scaling* is positive the data from the camera is additionally shifted by *scaling* places to the left, i.e. an image will become brighter. Similarly, if *scaling* is negative the data from the camera is additionally shifted by *scaling* places to the right, i.e. an image will become darker. Note that the TMG function *TMG_LUT_generate* can be used to generate LUTs giving proper brightness, contrast and gamma control, (see the examples given below).

LUT ARRAY SIZE

It is safe to always declare the array pointed to by *Plut* with 65536 elements; or the minimum size of the LUT can be determined by a call to *DIG16_get_LUT_max_addr*. *DIG16_set_LUTs* only writes the number of elements needed based on *DIG16_get_LUT_max_addr* - this greatly speeds up LUT programming for small LUTs. Note that the array is always *ui16*, even for 8 bit cameras.

RETURNS

When called with *DIG16_LUT_INIT* the function fills in the values written to the LUT in the array pointed to by *Plut*. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_OUT_OF_RANGE	The value of *scaling* would result in the data being shifted so far that no image would result.

EXAMPLES

The following code will result in the data from a 12 bit camera being passed unchanged through the LUT so that D11 from the camera ends up on D11 in the Himage:

```
DIG16_set_alignment(Hdig16, DIG16_ALIGN_LSB);
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 11, 0);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y16, TMG_Y16);
DIG16_set_LUTs(Hdig16, DIG16_LUT_INIT, 0, lut);
```

The following code will result in the data from a 12 bit camera being shifted left by 4 places so that D11 from the camera ends up on D15 in the Himage:

```
DIG16_set_alignment(Hdig16, DIG16_ALIGN_MSB);
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 11, 0);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y16, TMG_Y16);
DIG16_set_LUTs(Hdig16, DIG16_LUT_INIT, 0, lut);
```

The following code will result in the data from the same camera being shifted left by 3 places so that D11 from the camera ends up on D14 in the Himage:

```
DIG16_set_alignment(Hdig16, DIG16_ALIGN_MSB);
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 11, 0);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y16, TMG_Y16);
DIG16_set_LUTs(Hdig16, DIG16_LUT_INIT, -1, lut);
```

The following code will result in the data from a 10 bit camera being shifted right by 2 places so that D9 from the camera ends up on D7 in the Himage:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 9, 0);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y8, TMG_RGBX32);
DIG16_set_LUTs(Hdig16, DIG16_LUT_INIT, 0, lut);
```

The following code will result in the data from a 8 bit camera which is connected to D9..D2 being shifted right by 2 places so that D9 from the camera ends up on D7 in the Himage:

```
DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 2, 9);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y8, TMG_RGBX32);
DIG16_set_LUTs(Hdig16, DIG16_LUT_INIT, 0, lut);
```

The following code will set the LUT to binary threshold at level 100:

```
ui16 lut[256];

DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 7, 0);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y8, TMG_RGBX32);
for (lut_addr = 0; lut_addr <= 100; lut_addr++)
   lut[lut_addr] = 0;
for (lut_addr = 100; lut_addr <= 256; lut_addr++)
   lut[lut_addr] = 255;
DIG16_set_LUTs(Hdig16, DIG16_LUT_SET, 0, lut);
```

The following code will generate a LUT with brightness, contrast and gamma correction, etc:

```
ui8 pLut;

Thandle hLUT;

/* Create a 256 element LUT structure, and generate a LUT with the supplied
 * brightness, contrast and gamma settings.
 */
hLUT = TMG_LUT_create(256, 1);
TMG_LUT_generate(hLUT,  wBrightness, wContrast, wGamma, 0, 0, 0);
```

```
pLut = (ui8 *) TMG_LUT_get_ptr(hLUT, TMG_GRAY);

DIG16_set_camera_info(Hdig16, DIG16_CAMERA_NONINTERLACED, 7, 0);
DIG16_set_format(Hdig16, DIG16_FORMAT_Y8, TMG_RGBX32);
DIG16_set_LUTs(Hdig16, DIG16_LUT_SET, 0, pLut);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

DIG16_set_alignment, *DIG16_set_camera_info*, *DIG16_set_format*, *DIG16_set_image_data_width*, *DIG16_get_LUT_max_addr*.

DIG16_set_parameter

USAGE

Terr DIG16_set_parameter(Thandle Hdig16, ui16 parameter, ui32 value)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
parameter	The parameter to set.
value	The value to set the parameter to.

DESCRIPTION

This function sets various parameters in the internal structure associated with the Snapper-DIG16 handle.

PARAMETER

DIG16_PIXEL_COUNT	This sets the number of pixels which get stored per capture when in data stream mode. The parameter is only used in data stream mode. *DIG16_set_image* must then be called.
DIG16_TIMEOUT_BEFORE_CAPTURE	This sets the timeout value in milliseconds for the period from when *DIG16_capture_to_image* is called to when the first data is received. The default value is 2 seconds. Note that in external trigger mode if the trigger rate is slower than 2 seconds then this will need to be increased.
DIG16_TIMEOUT_DATA_TRANSFER	This sets the timeout value in milliseconds for the data transfer, i.e. following the first data being received. The default value is 4 seconds. Note that very large image transfers (especially in data stream mode) will need larger values.
DIG16_TIMEOUT_TRIGGER	This sets the timeout value in milliseconds for the period from when *DIG16_capture_to_image* is called to when the trigger is received. The default value is 4 seconds. Note that in external trigger mode if the trigger rate is slower than 4 seconds then this will need to be increased. This timeout is automatically increased as necessary if *DIG16_TIMEOUT_BEFORE_CAPTURE* is set to a value larger than *DIG16_TIMEOUT_TRIGGER*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

EXAMPLES

To request 40,000 pixels get stored per capture in data stream mode:

```
DIG16_set_parameter(Hdig16, DIG16_PIXEL_COUNT, (ui32) 40000L);
```

To increase the timeout before capture to 15 seconds when switching to external trigger mode:

```
DIG16_set_capture(Hdig16, DIG16_TRIG_IN_ENABLE);
DIG16_set_parameter(Hdig16, DIG16_TIMEOUT_BEFORE_CAPTURE, (ui32) 15000L);
```

BUGS / NOTES

There are no known bugs.

For the timeout parameters the granularity depends on the operating system in use, but will not be more than 1 second.

SEE ALSO

DIG16_get_parameter.

DIG16_set_ROI

USAGE

Terr DIG16_set_ROI(Thandle Hdig16, Tparam mode, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
roi	ROI array with four elements, with #defined element names:

ASL_ROI_X_START	Horizontal start position of ROI (0 = left of image).
ASL_ROI_Y_START	Vertical start position of ROI (0 = top of image).
ASL_ROI_X_LENGTH	Horizontal width of ROI.
ASL_ROI_Y_LENGTH	Vertical height of ROI.

DESCRIPTION

This function defines the ROI (Region of Interest). For many applications this function need not be called directly, because *DIG16_initialize* calls it to set the full ROI of the selected camera.

For conventional area scan cameras the top left corner of the image is defined with the *ASL_ROI_X_START* and *ASL_ROI_Y_START* coordinates and the image size defined with the *ASL_ROI_X_LENGTH* and *ASL_ROI_Y_LENGTH* values. All the coordinates are based upon raw image sizes in pixels and lines, ***not*** sub-sampled ones. This allows the image sub-sampling ratio to be varied for fast update or image resolution, without varying the ROI as well. The horizontal and vertical resolutions are 1 pixel and 1 line respectively.

For line scan mode the first pixel in the line is defined with the *ASL_ROI_X_START* value and the line width defined with the *ASL_ROI_X_LENGTH* value. These X coordinates are based upon raw image sizes in pixels, ***not*** sub-sampled ones, with a resolution of 1 pixel. The coordinate *ASL_ROI_Y_START* should be 0, and *ASL_ROI_Y_LENGTH* defines the number of lines to read per image (see the Concepts section at the start of the manual for more information on lines per image). The Y coordinate has a resolution of one line, but note that this controls how many lines are *captured*, not how many are *incoming*, therefore a constant number of lines are stored even if the line acceptance ratio is changed.

The benefit of using a reduced ROI compared to a full screen image is that the frame readout rate can be significantly faster because there is less data to read out.

The function checks the ROI before setting it in hardware. If the ROI requested is outside the valid range for the camera in use (as set by *DIG16_set_active_area*) the function does not return an error. Instead it trims the ROI, and returns the actual ROI set. This is done to simplify the use of the function with interactive software (e.g. window sizing).

Similarly, the ROI is adjusted to satisfy the rounding requirements (as set by *DIG16_set_ROI_rounding*).

MODE

DIG16_ROI_CHECK	The *roi* passed in is pre-processed (see below), but not actually set. This allows an application to check what ROI would get used if *DIG16_ROI_SET* was called, without having to change the setup of the Snapper.
DIG16_ROI_SET	The *roi* passed in is selected.

RETURNS

This function returns the actual ROI set in the *roi* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.

ASLERR_BAD_PARAMETER The mode parameter is invalid.

EXAMPLES

To select an ROI starting in the top left hand corner of the active area which is 1000 pixels wide by 10 lines deep:

```
roi[ASL_ROI_X_START] = 0;
roi[ASL_ROI_Y_START] = 0;
roi[ASL_ROI_X_LENGTH] = 1000;
roi[ASL_ROI_Y_LENGTH] = 10;
DIG16_set_ROI(Hdig16, DIG16_ROI_SET, roi);
```

The following shows the effect of the parameter pre-processing:

```
roi[ASL_ROI_X_START] = 15;
roi[ASL_ROI_Y_START] = 1;
roi[ASL_ROI_X_LENGTH] = 800;
roi[ASL_ROI_Y_LENGTH] = 600;
DIG16_set_active_area(Hdig16, roi);

roi[ASL_ROI_X_START] = 0;
roi[ASL_ROI_Y_START] = 0;
roi[ASL_ROI_X_LENGTH] = 1000;
roi[ASL_ROI_Y_LENGTH] = 400;
DIG16_set_ROI(Hdig16, DIG16_ROI_SET, roi);
/* roi[ASL_ROI_X_START] will still contain 0
 * roi[ASL_ROI_Y_START] will still contain 0
 * roi[ASL_ROI_X_LENGTH] will now contain 800, i.e. the value of 1000 has been
       clipped
 * roi[ASL_ROI_Y_LENGTH] will still contain 400
 */
```

BUGS / NOTES

There are no known bugs.

This function is not supported in data stream mode.

SEE ALSO

DIG16_get_ROI, *DIG16_get_ROI_max*, *DIG16_set_active_area*, *DIG16_set_ROI_rounding*, *DIG16_get_active_area*.

DIG16_set_ROI_rounding

USAGE

Terr DIG16_set_ROI_rounding(Thandle Hdig16, ui16 x_round, ui16 y_round)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
x_round	Required x direction rounding value
y_round	Required y direction rounding value

DESCRIPTION

x_round and *y_round* allow *DIG16_set_ROI* to automatically adjust ROIs. For many applications this function need not be called directly, because *DIG16_initialize* calls it with values *x_round* of 2 and *y_round* of 1.

If *x_round* is not one the horizontal width of an image is reduced to be a multiple of the number specified. For example, if it is required that all Himages have a width which is exactly divisible by 8 then *x_round* should be set to 8.

Similarly, if *y_round* is not zero the number of lines in an image is reduced to be a multiple of the number specified. For example, if it is required that all Himages have a height which is exactly divisible by 64 then *y_round* should be set to 64.

Note that it is the resulting image size allowing for subsampling which is adjusted, and not the x1 subsampling parameters of the ROI.

Both *x_round* and *y_round* can have values of 1, 2, 4, 8, 16, 32, 64, 128 or 256.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_OUT_OF_RANGE	*x_round* or *y_round* is not a supported value as listed above.

EXAMPLES

All ROIs set by subsequent calls to *DIG16_set_ROI* will be a multiple of 2 pixels wide, and a multiple of 8 lines high:

```
DIG16_set_ROI_rounding(Hdig16, 2, 8);
```

In this example the ROI specified does not need adjusting because 84 is divisible by 2:

```
DIG16_set_capture(Hdig16, DIG16_SUB_X1);
DIG16_set_ROI_rounding(Hdig16, 2, 1);
roi[ASL_ROI_X_LENGTH] = 84;
DIG16_set_roi(Hdig16, DIG16_ROI_SET, roi);
```

In this example the ROI specified will be adjusted because 21 (the resulting image width at x4 subsampling, i.e. 84 / 4) is not divisible by 2:

```
DIG16_set_capture(Hdig16, DIG16_SUB_X4);
DIG16_set_ROI_rounding(Hdig16, 2, 1);
roi[ASL_ROI_X_LENGTH] = 86;
DIG16_set_roi(Hdig16, DIG16_ROI_SET, roi);
/* roi[ASL_ROI_X_LENGTH] now 80 giving an image width of 20 */
```

BUGS / NOTES

There are no known bugs.

It is recommended that *x_round* is set to 2 or greater because some image processing libraries, including many functions in the TMG library, cannot cope with odd width images.

Any changes in the settings of *DIG16_set_ROI_rounding* will not take affect until *DIG16_set_ROI* has been called.

This function is not supported in data stream mode.

SEE ALSO

DIG16_set_ROI, *DIG16_get_ROI_max*.

DIG16_set_timer

USAGE

*Terr DIG16_set_timer(Thandle Hdig16, Tparam mode, ui32 time, Terr (EXPORT_FN *time_fn)(Thandle))*

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required timer mode.
time	Required time pulse width.
time_fn	Required timer control function.

DESCRIPTION - AREA SCAN, DATA STREAM MODES

This function controls the timer on the baseboard to generate pulses, typically for camera exposure control. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the parameter *DIG16_TIMER_INIT*.

DIG16_set_timer sets up a timer control function which is automatically called by *DIG16_capture_to_image* to generate a timed pulse for the camera. A default timer control function provides a single pulse of programmable width. If a sequence of pulses is required then a custom function can be written, and *DIG16_capture_to_image* will automatically call it.

The hardware timer pulse can only be output on the OUT_A pin, so *DIG16_set_ctrl_io* must be called to set the OUT_A pin in the required mode. However custom software functions can control other I/O or output pins and be utilised by calling *DIG16_TIMER_SET_TIMER_FN.*

The timer hardware is fitted to the baseboard. Some baseboards do not provide this function, resulting in the first call to *DIG16_capture_to_image* with the timer enabled failing with error *ASLERR_NOT_SUPPORTED* from function *BASE_set_timer*.

The mode parameter should be one of the following:

MODE	
DIG16_TIMER_INIT	This selects the default timer function, but disables the timer by setting the exposure time to zero. The values of *time* and *time_fn* are ignored.
DIG16_TIMER_SET_EXPOSURE	This controls the exposure time used by the default timer function. The exposure time is passed in the *time* parameter in microseconds. The value of *time_fn* is ignored.
DIG16_TIMER_SET_TIMER_FN	This allows a custom timer function to be called. This might be used if a sequence of pulses is required for a camera. A pointer to the custom function is passed in the *time_fn* parameter. The value of *time* is ignored.

DESCRIPTION - LINE SCAN MODE

This function controls a hardware timer on the Snapper-DIG16 to generate pulses, typically for camera exposure control. A pulse of the requested width is generated once per line captured, following the active edge of the selected line start signal. The baseboard timer is not used, and is therefore free to generate a line trigger (see *DIG16_set_linescan_ctrl*).

The timer pulse can only be output on the OUT_A pin, so *DIG16_set_ctrl_io* must be called to set the OUT_A pin in the required mode.

MODE

DIG16_TIMER_SET_EXPOSURE	This controls the exposure time used by the default timer function. The exposure time is passed in the *time* parameter in microseconds. The value of *time_fn* is ignored. The timer has a resolution of about 53us, with a maximum period of 13.6ms.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.

EXAMPLES

To enable a single active low pulse on the trigger pin, of width 10ms, to be automatically generated on each capture from then on:

```
DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_OUT_A_EXP_NEG);
DIG16_set_timer(Hdig16, DIG16_TIMER_SET_EXPOSURE, (ui32) 10000, NULL);
...
DIG16_capture_to_image(Hdig16, Himage, DIG16_START_AND_WAIT;
```

To set up a custom timer function which enables an active high pulse of 1ms, followed by a low pulse of 1s, followed by a high pulse of 1ms, to be automatically generated on each capture from then on (area scan and data stream modes only):

```
main()
{
   ...
   DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_OUT_A_EXP_POS);
   DIG16_set_timer(Hdig16, DIG16_TIMER_SET_TIME_FN, 0, my_timer_function);
   ...
}

Terr EXPORT_FN my_timer_function(Thandle Hdig16)
{
   Thandle Hbase;

   /* First get baseboard handle so BASE functions can be called */
   Hbase = DIG16_get_parameter(Hdig16, DIG16_BASEBOARD_HANDLE);

   /* Generate first pulse */
   BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000);
   /* Hold exposure line low for 1 second */
   DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_OUT_A_0);
   BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000000L);
   /* Finally generate second pulse */
   DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_OUT_A_EXP_POS);
   BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000);
   return( ASL_OK );
}
```

BUGS / NOTES

There are no known bugs.

If the exposure is very long then the capture timeout parameter *DIG16_TIMEOUT_BEFORE_CAPTURE* will need to be extended - see *DIG16_set_parameter*.

SEE ALSO

BASE_set_timer, *DIG16_set_ctrl_io*.

DIG16_set_trigger

USAGE

Terr DIG16_set_trigger(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required trigger mode.

DESCRIPTION

This function sets the capture trigger mode. A pulse on the hardware trigger input results in a single frame being captured in area scan mode, or allows a group of lines to be captured in line scan mode. For many applications this function need not be called directly, because *DIG16_initialize* calls it with the parameter *DIG16_TRIG_INIT*. Note that *DIG16_TRIG_IN_POS* and *DIG16_TRIG_IN_NEG* do not enable triggering - to do this use call *DIG16_set_capture* with *DIG16_TRIG_IN_ENABLE*.

The mode parameter should be one of the following:

MODE

DIG16_TRIG_INIT	The first call to *DIG16_set_trigger* should include this parameter. It selects IO_B as the trigger pin is an input with the rising edge active.
DIG16_TRIG_IN_POS	This specifies that the selected trigger pin has the rising edge active if it is an edge trigger, or active high if it is a level trigger.
DIG16_TRIG_IN_NEG	This specifies that the selected trigger pin has the falling edge active if it is an edge trigger, or active low if it is a level trigger.
DIG16_TRIG_IN_LEVEL	This specifies that the trigger occurs when the required level occurs on the selected trigger pin following the start of capture.
DIG16_TRIG_IN_EDGE	This specifies that the trigger occurs when the required edge occurs on the selected trigger pin following the start of capture.
DIG16_TRIG_GATED	This specifies that the trigger pin is also used to gate data acquisition on a line by line basis. Once the trigger condition has occurred, the selected trigger input pin is sampled at the start of each line, and used to determine whether the current line of data is acquired. The polarity of the gating is common with that of the trigger, i.e. if *DIG16_TRIG_IN_POS* is used, then the system will trigger on a rising edge or high level, and the acquisition will also be gated with a high level on the trigger input. Note that this feature currently only applies to line scan mode.
DIG16_TRIG_NON_GATED	This specifies that the trigger pin is only used to start acquisition. Once the trigger condition has occurred, acquisition continues until the software terminates capture. Note that this feature is the default setting in area scan and data stream modes.
DIG16_TRIG_IN_IO_A	This specifies that the IO_A pin is used as the trigger in. *DIG16_set_ctrl_io* should be called to set this pin as an input.
DIG16_TRIG_IN_IO_B	This specifies that the IO_B pin is used as the trigger in. *DIG16_set_ctrl_io* should be called to set this pin as an input.
DIG16_TRIG_IN_IO_C	This specifies that the IO_C pin is used as the trigger in. *DIG16_set_ctrl_io* should be called to set this pin as an input.

DIG16_TRIG_IN_IO_D	This specifies that the IO_D pin is used as the trigger in. *DIG16_set_ctrl_io* should be called to set this pin as an input.
DIG16_TRIG_IN_TTL1	This specifies that the TTL_TRIG1 pin is used as the trigger in. Note that this pin is only available on recent boards.
DIG16_TRIG_IN_TTL2	This specifies that the TTL_TRIG2 pin is used as the trigger in. Note that this pin is only available on recent boards.
DIG16_TRIG_IN_FRAME	This specifies that the FRAME pin is used as the trigger in. This is currently only available in area scan mode.
DIG16_TRIG_IN_LINE	This specifies that the LINE pin is used as the trigger in. This is currently only available in area scan mode.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.
ASLERR_NOT_SUPPORTED	A TTL trigger is selected on a board which does not support it.

EXAMPLES

To select falling edge trigger on pin IO_A, then acquire the image on a trigger input pulse:

```
DIG16_set_ctrl_io(Hdig16, DIG16_CTRL_IO_A_IN);
DIG16_set_trigger(Hdig16, DIG16_TRIG_IN_IO_A | DIG16_TRIG_IN_NEG);
DIG16_set_capture(Hdig16, DIG16_TRIG_IN_ENABLE);
DIG16_capture_to_image(Hdig16, Himage, DIG16_START_AND_WAIT);
```

BUGS / NOTES

There are no known bugs.

See the Camera Specific Release Notes in the Installation section of the manual for cable connections and Snapper-DIG16's pinout.

SEE ALSO

DIG16_is_trigger_started, *DIG16_set_capture*.

Camera Specific Functions

DIG16_set_C4742_ctrl
DIG16_set_mplus_ctrl
DIG16_set_X1400_ctrl

The functions are described in alphabetical order in the following pages.

DIG16_set_C4742_ctrl

USAGE

Terr DIG16_set_C4742_ctrl(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required control mode.

DESCRIPTION

This function allows control of Hamamatsu Photonics C4742 cameras by making appropriate calls to *DIG16_set_ctrl_io*. *DIG16_initialize(..., DIG16_HAMAMATSU_C4742)* calls it with the parameter *DIG16_C4742_MODE* | 4.

Note that the camera mode must be switched to '0' for this function to have any effect, and the shutter switch on the camera control unit needs to be set to 'ENB'. Also switch SW1-2 in the camera control unit must be set to 'Off' to get the correct active area. See the camera's manual for details of how to do this and for full descriptions of the available camera modes.

The mode parameter should be a combination of the following:

MODE

DIG16_C4742_MODE	The required C4742 mode should be ORed with this parameter, e.g. (*DIG16_C4742_MODE* | 2) to set mode 2. The mode number should be one of the modes supported by the camera, i.e. one of 0, 2, 3, 4, 6 or 7. See the camera's manual for details of these modes.
DIG16_C4742_PULSE_EXT_IN	This pulses the EXT_IN line to the camera which will cause it to start an exposure when it is in modes 2, 3, 6 or 7.

In mode 7, where the exposure time is controlled by the EXT_IN line, this line driven by the baseboard timer. *DIG16_set_timer* must be called to set the required exposure time.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_OUT_OF_RANGE	The C4742 mode is not supported by the camera.

EXAMPLES

To set the a C4742 camera in mode 4:

```
DIG16_set_C4742_ctrl(Hdig16, DIG16_C4742_MODE | 4);
```

BUGS / NOTES

There are no known bugs.

The function relies on the pin connections in the camera cable matching those of the standard cable part number CBL-68-37D-A-2M. A cable with different I/O connections could be used, but direct calls to *DIG16_set_ctrl_io* would be needed.

SEE ALSO

DIG16_set_ctrl_io.

DIG16_set_mplus_ctrl

USAGE

Terr DIG16_set_mplus_ctrl(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required control mode.

DESCRIPTION

This function allows control of Kodak Megaplus 1.4, 4.2 and similar cameras by making appropriate calls to *DIG16_set_ctrl_io*. *DIG16_initialize(..., DIG16_KODAK_MPLUSXX)* calls it with the parameters *DIG16_MPLUS_CONTINUOUS* and *DIG16_MPLUS_SHUTTER_ON.*

Note that the camera control unit must be switched to 'Computer' for this function to have any effect, and the shutter switch on the camera control unit may be need to be set to 'Off'. See the camera's manual for details of how to do this and for full descriptions of the available camera modes.

The mode parameter should be a combination of the following:

MODE

DIG16_MPLUS_CONTINUOUS	This sets the camera mode as 'Continuous', so it continuously starts a self-timed expose-transfer-idle sequence, regardless of the setting of frame-reset/exposure.
DIG16_MPLUS_TRIGGERED	This sets the camera mode as 'Triggered', so it continuously starts a self-timed expose-transfer-idle sequence if frame-reset/exposure is low (i.e. expose).
DIG16_MPLUS_CONTROLLED	This sets the camera mode as 'Controlled', so it continuously starts a expose-transfer-idle sequence if frame-reset/exposure is low (i.e. expose), and the exposure time is controlled by frame-reset/exposure, which is driven by the baseboard timer. *DIG16_set_timer* must be called to set the required exposure time.
DIG16_MPLUS_SHUTTER_RUN	This sets the camera shutter running.
DIG16_MPLUS_SHUTTER_OPEN	This locks the camera shutter open.
DIG16_MPLUS_FRAME_EXPOSE	This drives the camera frame-reset/exposure line low, so an exposure can start (depending on the mode selected above).
DIG16_MPLUS_FRAME_RESET	This drives the Megaplus frame-reset/exposure line high.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.

EXAMPLES

To set the a Megaplus camera in Continuous mode with the shutter running:

```
DIG16_set_mplus_ctrl(Hdig16, DIG16_MPLUS_CONTINUOUS | DIG16_MPLUS_SHUTTER_ON);
```

The following will result in a parameter conflict error because the camera cannot be in Continuous mode and Triggered mode at the same time:

```
DIG16_set_ctrl_io(Hdig16, DIG16_MPLUS_CONTINUOUS | DIG16_MPLUS_TRIGGERED);
```

BUGS / NOTES

There are no known bugs.

The function relies on the pin connections in the camera cable matching those of the standard cable part number CBL-68-37D-A-2M. A cable with different I/O connections could be used, but direct calls to *DIG16_set_ctrl_io* would be needed.

SEE ALSO

DIG16_set_ctrl_io.

DIG16_set_X1400_ctrl

USAGE

Terr DIG16_set_X1400_ctrl(Thandle Hdig16, Tparam mode)

ARGUMENTS

Hdig16	Handle to Snapper-DIG16.
mode	Required control mode.

DESCRIPTION

This function allows control of the Xillix MicroImager 1400 camera by making appropriate calls to *DIG16_set_ctrl_io*. *DIG16_initialize(..., DIG16_XILLIX_1400)* calls it with the parameter *DIG16_X1400_FREQ_8MHZ.*

The camera exposure is set by the *DIG16_set_timer* function.

The mode parameter should be one of the following:

MODE	
DIG16_X1400_FREQ_500KHZ	This sets the camera clock frequency to 500kHz.
DIG16_X1400_FREQ_1MHZ	This sets the camera clock frequency to 1MHz.
DIG16_X1400_FREQ_4MHZ	This sets the camera clock frequency to 4MHz.
DIG16_X1400_FREQ_8MHZ	This sets the camera clock frequency to 8MHz.

Lower readout frequencies are used to improve noise characteristics whereas higher readout frequencies are used when acquisition speed is more important.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-DIG16 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	Two or more of the mode parameters conflict with each other.

EXAMPLES

To set the MicroImager 1400 camera to run at 4MHz clock frequency:

```
DIG16_set_X1400_ctrl(Hdig16, DIG16_X1400_FREQ_4MHZ);
```

The following will result in a parameter conflict error because the camera cannot be set to run at 4MHz and 1MHz at the same time:

```
DIG16_set_X1400_io(Hdig16, DIG16_X1400_FREQ_4MHZ | DIG16_X1400_FREQ_1MHZ);
```

BUGS / NOTES

There are no known bugs.

Different camera types are defined to allow use of the two MicroImager 1400 clocking modes. These are *DIG16_XILLIX_1400* for normal clocking mode and *DIG16_XILLIX_1400BIN2* for 2x2 binning mode. The

DIG16_initialize function must be called with the appropriate parameter to reconfigure the camera for the required binning option.

After the CCD has been exposed the MicroImager 1400 camera outputs the full frame of data and cannot be stopped. If the ROI is set to be much less than the full image height, it is possible for the image data to be processed before the camera has finished outputting the current frame. If another capture is then initiated, the CCD will be re-exposed and will generate a multiple image. The application software must therefore ensure that the camera has completed outputting its data before another capture is initiated, by use of a suitable delay. The delay value must take into account the number of lines of data not stored as part of the ROI, the MicroImager 1400 camera clock frequency, and the binning mode selected.

SEE ALSO

DIG16_set_ctrl_io.