SNAPPER-24 Library

**(for Snapper-24, Snapper-8,
Snapper-PCI24, Snapper-PCI8,
Snapper-PMC24 and Snapper-PMC8)**

Programmer's Manual

DataCell Limited

v4.0.1

Disclaimer

While every precaution has been taken in the preparation of this manual, DataCell Ltd assumes no responsibility for errors or omissions. DataCell Ltd reserves the right to change the specification of the product described within this manual and the manual itself at any time without notice and without obligation of DataCell Ltd to notify any person of such revisions or changes.

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-SNP24-LIB

Version v4.0.1 September 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-24 function library. These functions allow the capture of video images, using a Snapper-24 module and one of a number of different host hardware platforms, and are independent of the host hardware platform. This manual also applies to the Snapper-PCI24 and Snapper-PMC24 products, which are single board combinations of a PCI-BIB baseboard and a Snapper-24 module.

The Snapper-24 library is also used for Snapper-8 applications. This is because the Snapper-8 is physically a Snapper-24 with only one digitizer / memory channel fitted rather than all three channels. All applications written for Snapper-8 will run identically on a Snapper-24 provided that the function *SNP24_set_format* is called directly. Also, by using the *SNP24_get_ID* function, it is possible to write Snapper-24 applications which can also run in some modes on a Snapper-8. All references in this manual to Snapper-24 also apply to Snapper-8 unless stated otherwise.

Snapper-24 is referenced 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

Snapper-24 has on-board video memory which is arranged as two banks. The *SNP24_capture* function stores data into this memory, and the *SNP24_read_video_data* function reads data out of the memory. The use of two banks of memory allows interlaced images to be automatically deinterlaced, and allows faster acquisition rates than would be possible with one bank. The use of two banks is transparent to the application. Each bank can store approximately 256K pixels.

CONVENTIONAL CAMERAS

"Conventional cameras" are "area scan" cameras, such as standard CCIR (50Hz) or EIA (60Hz, RS-170/170A) formats, which generate images of *m* pixels by *n* lines (e.g. 640 by 480 for EIA cameras). This term includes both commonly available interlaced cameras as well as progressive scan cameras, but does not include "line scan" cameras - see below.

SINGLE CAPTURE MODE

This is the default mode, where a call to *SNP24_capture* only stores one image. Applications are simple to program, but this mode does not take advantage of the speed up possible with two memory banks. This limits the rate at which successive frames can be captured. In general even if a new capture is started as soon as the previous one completes, it is probable that one incoming field or frame will be missed between those captured. This means that a full frame image is limited to half real time acquisition (i.e. 12.5Hz for CCIR, or 15Hz for EIA), but sub-sampled images (i.e. only one field) can be captured real time.

SEQUENCE MODE

To allow real time capture rate Snapper-24 can be used in "sequence mode". This mode uses both memory banks, allowing capture into one bank while the other bank is being read from. There is no point in using sequence capture mode unless fast capture rates are needed because the application code is slightly more complex than for single capture mode.

A significant difference when using sequence mode rather than single capture mode is that interlaced images are always read out field by field, i.e. it is the responsibility of the application to deinterlace the fields into one frame. Conversely in single capture mode the process of deinterlacing is by default hidden from the application. See the TMG Programmer's Manual (section 2 "Concepts", and function *TMG_display_image*) for functions which handle a sequence of fields.

If the read out (and any processing) of images in sequence mode takes longer than the time it takes to acquire a field, the Snapper-24 hardware will automatically drop fields. If the capture mode is a frame (i.e. x1) then field pairs are dropped. For example if in frame mode a field 1 (incoming sequence image 1) has just been read and processed, and this takes longer than one field time, then:

- The following field 2 (incoming sequence image 2) will be captured OK because this will have been capturing into the second Snapper-24 memory bank at the same time as the field 1 was being read.
- The following field 1 (incoming sequence image 3) will be dropped because the first Snapper-24 memory bank was still being read from when this field should have been captured.
- The following field 2 (incoming sequence image 4) will also be dropped to ensure that successive images read alternate between field 1 and field 2.
- Assuming that image sequence 1 has now been read, the following field 1 (incoming sequence image 5) will be captured OK.

Whenever fields are dropped in sequence mode a flag is set (function *SNP24_is_sequence_dropped*).

CALLBACKS METHOD

Callbacks are software interrupts built into the Snapper-24 library that call user application code on specific events, for example, at the end of data transfer or at the start of the vertical sync time from the camera. While programming with callbacks is slightly more complex, it has two main advantages:

- The image acquisition is not tied to the rate at which the data can be processed. Therefore if on average the processing is quicker than the acquisition, but occasionally a frame takes longer to process, the acquisition will still continue and not lose data.
- The libraries can sleep whilst waiting for events to occur and therefore free system resources for other tasks.

Although the callbacks method can be used with both single capture and sequence modes, it is generally used with sequence mode for faster acquisition and processing speeds.

LINE SCAN MODE

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. Line scan cameras generally use RS-422 synchronous control signals, and the timing of these signals varies between cameras. Therefore specific cameras are supported, and Snapper-24 is set up for a camera by calling *SNP24_initialize* with the camera model as a parameter. Cameras which are not supported by *SNP24_initialize* can be set up from the application by direct calls to low level line scan control functions - see the list in section "Function List".

It is expected that line scan applications will use sequence mode to allow continuous capture of lines. Snapper-24 can be set to capture a given number of lines in one bank before switching to the other bank (function *SNP24_set_ROI*). At a minimum, one line can be captured per bank - 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 the programmable ROI width being captured and the fixed bank size. A large number of lines gives a low processing overhead, but a larger latency. Typical values might be 32 or 64 lines per bank.

Most of the Snapper-24 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.

IMPORTANT: Line scan mode on Snapper-24 is not supported in this release of software.

Function Overview

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

INITIALIZATION FUNCTION

The initialization function configures Snapper-24 to default settings. It accepts a parameter indicating the format or model of camera which is connected so that Snapper-24 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 *SNP24_capture* function controls the capture of video data into Snapper-24's video memory. *SNP24_read_video_data* reads the data from Snapper-24's video memory into an image structure in host memory. This image structure is set up by *SNP24_set_image*. *SNP24_reset_read_pointer* allows the data to be read again. Capture status is indicated by *SNP24_is_50Hz*, *SNP24_is_capture_complete*, *SNP24_is_data_ready*, *SNP24_is_field1_captured*, *SNP24_is_field1_incoming*, *SNP24_is_locked*, *SNP24_is_sequence_dropped*, *SNP24_is_sequence_mode*, *SNP24_is_trigger_started* and *SNP24_is_vsync_asserted*.

CONFIGURATION FUNCTIONS

These functions control the configuration of the Snapper-24.

The capture mode used by the next *SNP24_capture* call is controlled by *SNP24_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 *SNP24_set_trigger* function. Selected regions of the image can be captured by a call to *SNP24_set_ROI*.

The video source to be digitized is controlled by *SNP24_set_video_src* and *SNP24_set_format*. The analogue levels at the input to the ADCs are controlled by *SNP24_set_levels*, and the LUTs on the output of the ADCs are controlled by *SNP24_set_LUTs*.

SNP24_set_sync selects the sync source, and *SNP24_set_timer* allows timed pulses to be supplied to the camera to control for example exposure time.

Interrupt control of acquisition is possible using *SNP24_set_interrupts* and *SNP24_set_callback*.

The remaining functions will not get called in a typical application because they are called by *SNP24_initialize* with the correct settings for the camera. These functions are *SNP24_lm1882_prog*, *SNP24_set_active_area*, *SNP24_set_clamp*, *SNP24_set_clk*, *SNP24_set_ctrlout*, *SNP24_set_linescan_ctrl*, *SNP24_set_linescan_freq*, *SNP24_set_pix_per_line*, *SNP24_set_ROI_rounding*, *SNP24_set_TTL422* and *SNP24_set_video_standard*.

PARAMETER READBACK FUNCTIONS

Some of these functions are intended to avoid the need for an application to keep shadow copies of Snapper-24 settings. These are *SNP24_get_active_area*, *SNP24_get_levels*, *SNP24_get_LUTs*, *SNP24_get_ROI* and *SNP24_get_subsample*.

SNP24_get_ROI_max returns maximum settings for the camera in use, and *SNP24_get_property* returns hardware and firmware information about the Snapper-24.

SNP24_get_ID which shows whether a Snapper-8 or Snapper-24 is connected, and *SNP24_get_rev* which returns the hardware revision of the Snapper-24 in use.

Finally *SNP24_get_parameter* returns general information about the Snapper-24.

MISCELLANEOUS FUNCTIONS

SNP24_auto_gain and *SNP24_auto_offset* allow the brightness of an image to be set automatically.

Error Returns

All of the Snapper-24 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-24 - 0xB0 (#defined as *SNP24_FAMILY_ID*)
on Snapper-8 - 0xB8 (#defined as *SNP8_FAMILY_ID*).

Bits 26 to 24 give the hardware revision level. Initial Snapper-24s 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 show a minimal program, a sequence mode example, and a callbacks code fragment. 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.

MINIMAL PROGRAM

The following code is a minimal program to capture an image from a RGB CCIR camera using a Snapper-24 and save it to a file:

```
#include <asl_inc.h>

int main(ui16 argc, char** argv)
{
   Thandle Hsnp24;        /* Handle to Snapper-24 */
   Thandle Hbase;         /* Handle to baseboard */
   Thandle Hvid_image;    /* Handle to captured image */
   Thandle Hrgb24_image;  /* Handle to RGB24 image for TIFF write */

   /* Initialize baseboard and Snapper module */
   Hbase = ASL_get_ret(BASE_create(BASE_AUTO));
   Hsnp24 = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
   Hvid_image = TMG_image_create();
   Hrgb24_image = TMG_image_create();

   /* Set required Snapper mode */
   SNP24_initialize(Hsnp24, SNP24_CCIR_DEFAULT);

   /* Set up image parameters */
   SNP24_set_image(Hsnp24, Hvid_image);

   /* Capture image */
   SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
   SNP24_read_video_data(Hsnp24, Hvid_image, TMG_RUN);

   /* Write image as a TIFF file, so first convert it to RGB24 format */
   TMG_image_convert(Hvid_image, Hrgb24_image, TMG_RGB24, 0, TMG_RUN);
   TMG_image_set_outfilename(Hrgb24_image, "rgb24.tif");
   TMG_image_write(Hrgb24_image, TMG_NULL, TMG_TIFF, TMG_RUN);

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

SEQUENCE MODE EXAMPLE

The following code shows how to use sequence mode to capture images from a RGB EIA camera using a Snapper-24:

```
#include <asl_inc.h>

int main(ui16 argc, char** argv)
{
   Thandle Hsnp24, Hbase, Hvid_image; /* Handles to Snapper-24, baseboard & image */
   Tboolean finished = FALSE;         /* Controls when to stop sequence capture */

   /* Initialize baseboard and Snapper module */
   Hbase = ASL_get_ret(BASE_create(BASE_AUTO));
   Hsnp24 = BASE_get_parameter(Hbase, BASE_MODULE_HANDLE);
   Hvid_image = TMG_image_create();

   /* Set required Snapper mode, including selecting sequence mode */
   SNP24_initialize(Hsnp24, SNP24_EIA_DEFAULT);
   SNP24_set_capture(Hsnp24, SNP24_SEQUENCE_MODE);
   SNP24_set_image(Hsnp24, Hvid_image);

   /* Start continuous capture of images - first prepare SNP24_read_video_data
    * to receive data, then start the actual capture
    */
   SNP24_read_video_data(Hsnp24, Hvid_image, TMG_INIT);
   SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);

   /* Now continuously read in the data as it is captured */
   while (finished == FALSE)
   {
      /* Wait for an image to arrive in video memory */
      while (SNP24_is_data_ready(Hsnp24) == FALSE)
      {}

      /* Now read the image */
      SNP24_read_video_data(Hsnp24, Hvid_image, TMG_STRIP);

      /* 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);
      TMG_image_set_flags(Hvid_image, TMG_LOCKED, TRUE);   /* Speed up loop */
   }

   /* Stop the continuous capturing, then inform SNP24_read_video_data
    * that no more data will be read in this continuous capture
    */
   SNP24_capture(Hsnp24, SNP24_END_SEQUENCE);
   SNP24_read_video_data(Hsnp24, Hvid_image, TMG_RESET);

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

SEQUENCE MODE CODE FRAGMENT – USING CALLBACKS

The following code shows how to use callbacks with sequence mode to capture images from a RGB EIA camera using a Snapper-24:

```
/* Global structure to contain all required info */
struct MyInfo
{
   Thandle Himages[NUM_BUFFERS];
   ui32 dwProcessBuffer;
   ui32 dwAcquisitionBuffer;
}

/* Setup Snapper-24 as appropriate */
SNP24_set_capture(Hsnp24, SNP24_SEQUENCE_MODE);
…
/* Initialise the required image buffers */
for (dwBuffer = 0; dwBuffer < NUM_BUFFERS; dwBuffer++)
{
   SNP24_set_image(Hsnp24, MyInfo.Himages[dwBuffer]);
   SNP24_read_video_data(Hsnp24, MyInfo.Himages[dwBuffer], TMG_INIT);
   TMG_image_set_flags(MyInfo.Himages[dwBuffer], TMG_LOCKED, TRUE);
}

/* Install the interrupt handler and enable the appropriate interrupt */
SNP24_set_callback(Hsnp24, SNP24_CALLBACK_SET, MyCallback, &MyInfo);
SNP24_set_interrupts(Hsnp24, SNP24_INT_DATA_READY, TRUE);

/* Start interrupt driven acquisition */
SNP24_capture(Hsnp24, START_AND_RETURN);

/* This main program thread now waits on a semaphore and then processes the
 * images when signalled from the interrupt handler.
 */

MyInfo.dwProcessBuffer = 0;
while (!bFinished)   /* A global could be used to signal when finished */
{
   /* Wait until image has been acquired into host memory */
   WaitForSingleObject(hSemaImageReady, 500);  /* Win32 API */

   /* Process image */
   ProcessImage(&MyInfo);
   SelectNextProcessBuffer(&MyInfo);
}

/* Disable interrupt handler */
SNP24_set_callback(Hsnp24, SNP24_CALLBACK_INIT, NULL, NULL);
SNP24_capture(Hsnp24, SNP24_END_SEQUENCE);
```

The interrupt handler, called on image acquisition complete (i.e. image data acquired into on-board memory), then reads out the image data from Snapper and signals to the processing thread.

```
/* This is our interrupt handler */
void InterruptHandler(Thandle Hsnp24, ui32 dwIntSrc, void *pMyData)
{
   struct MyInfo *pMyInfo = (struct MyInfo *) pMyData;
   Thandle hThisImage = pMyInfo->Himages[pMyInfo->dwAcquisitionBuffer];

   SNP24_read_video_data(Hsnp24, hThisImage, TMG_STRIP);
   SelectNextAcquisitionBuffer(pMyInfo);
   ReleaseSemaphore(hSemaImageReady, 1, NULL);  /* Win32 API */
}
```

Function List

Functions Supported In All Modes

INITIALIZATION FUNCTION

SNP24_initialize

IMAGE CAPTURE FUNCTIONS

SNP24_capture
SNP24_is_capture_complete
SNP24_is_data_ready
SNP24_is_sequence_dropped
SNP24_is_sequence_mode
SNP24_is_trigger_started
SNP24_read_video_data
SNP24_reset_read_pointer
SNP24_set_image

CONFIGURATION FUNCTIONS

SNP24_set_active_area
SNP24_set_capture
SNP24_set_clamp
SNP24_set_clk
SNP24_set_format
SNP24_set_levels
SNP24_set_LUTs
SNP24_set_parameter
SNP24_set_ROI
SNP24_set_ROI_rounding
SNP24_set_timer
SNP24_set_trigger
SNP24_set_TTL422
SNP24_set_video_src
SNP24_set_video_standard

PARAMETER READBACK FUNCTIONS

SNP24_get_active_area
SNP24_get_ID
SNP24_get_levels
SNP24_get_LUTs
SNP24_get_parameter
SNP24_get_property
SNP24_get_rev
SNP24_get_ROI
SNP24_get_ROI_max
SNP24_get_subsample

Functions Supported In Some Modes

A = AREA SCAN L = LINE SCAN

A *SNP24_is_50Hz*
A *SNP24_is_field1_captured*
A *SNP24_is_field1_incoming*
A *SNP24_is_vsync_asserted*
A *SNP24_is_locked*

A *SNP24_lm1882_prog*
A *SNP24_set_ctrlout*
L *SNP24_set_linescan_ctrl*
L *SNP24_set_linescan_freq*
A *SNP24_set_pix_per_line*
A *SNP24_set_sync*

MISCELLANEOUS FUNCTIONS

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

SNP24_auto_gain

USAGE

Terr SNP24_auto_gain(Thandle Hsnp24, ui8 percent_clip)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
percent_clip	Percentage of image allowed to overshoot peak white.

DESCRIPTION

This function is used to automatically set the white level of the image. It does this by capturing one image, and analysing it to produce a new white level, which it then sets. The white level is chosen to make *percent_clip* percentage of the image overshoot peak white. For typical images a *percent_clip* value between 1 and 5 is recommended, but values between 1 and 99 are accepted by the routine.

RETURNS

The function returns the new white level. It can return the following error codes:

ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_OUT_OF_RANGE	The percentage passed is not between 1 and 99.

It can also return error returns from other SNP24 and TMG functions it calls.

EXAMPLES

See *SNP24_auto_offset*.

BUGS / NOTES

Note that as the function captures one image using the existing capture mode, if external trigger is enabled a trigger event is needed before the function will return. Also, the board must have been initialized before the function is called, otherwise the associated *SNP24_capture* call will fail. Similarly, if the function is called immediately after selecting a new video source the hardware may not have locked to the new source, again causing *SNP24_capture* to fail. If a sequence mode capture is in progress the function will 'steal' four images from the sequence.

A limitation when using an RGB source is that the peak level in the image should be near white, otherwise a greater percentage of the image will be clipped to white than requested (i.e. it is not sufficient to have an image which has full intensity red, green, or blue on their own). An ideal test image for both this function and *SNP24_auto_offset* would be half black and half white. Also note that the image captured by this function will have passed through the currently installed LUTs.

The function adjusts the white level relative to the current setting of black level, therefore the black level should be set (typically to default, or using *SNP24_auto_offset*) before calling this function.

This function is not supported for line scan cameras, and is not supported if sequence mode is selected, but no sequence capture is in progress.

SEE ALSO

SNP24_auto_offset, *SNP24_set_levels*.

SNP24_auto_offset

USAGE

Terr SNP24_auto_offset(Thandle Hsnp24, ui8 percent_clip)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
percent_clip	Percentage of image allowed to undershoot black.

DESCRIPTION

This function is used to automatically set the black level of the image. It does this by capturing one image, and analysing it to produce a new black level, which it then sets. The black level is chosen to make *percent_clip* percentage of the image undershoot black. For typical images a *percent_clip* value between 1 and 5 is recommended, but values between 1 and 99 are accepted by the routine.

RETURNS

The function returns the new black level. It can return the following error codes:

ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_OUT_OF_RANGE	The percentage passed is not between 1 and 99.

It can also return error returns from other SNP24 and TMG functions it calls.

EXAMPLES

The following code will set white, clamp and YCbCr to default levels, then automatically set black and white levels so that 5% of the current image undershoots black and 5% overshoots peak white:

```
SNP24_set_levels(Hsnp24, SNP24_LVL_INIT, levels);
SNP24_auto_offset(Hsnp24, 5);
SNP24_auto_gain(Hsnp24, 5);
```

BUGS / NOTES

Note that as the function captures one image using the existing capture mode, if external trigger is enabled a trigger event is needed before the function will return. Also, the board must have been initialized before the function is called, otherwise the associated *SNP24_capture* call will fail. Similarly, if the function is called immediately after selecting a new video source the hardware may not have locked to the new source, again causing *SNP24_capture* to fail. If a sequence mode capture is in progress the function will 'steal' four images from the sequence.

The function adjusts the black level relative to the current setting of clamp level, therefore the clamp level should be set (typically to default) before calling this function. If both *SNP24_auto_offset* and *SNP24_auto_gain* are called, it is recommended that *SNP24_auto_offset* is called first. Also note that the image captured by this function will have passed through the currently installed LUTs.

This function is not supported for line scan cameras, and is not supported if sequence mode is selected, but no sequence capture is in progress.

SEE ALSO

SNP24_auto_gain, *SNP24_set_levels*.

SNP24_capture

USAGE

Terr SNP24_capture(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required capture mode.

DESCRIPTION

This function is used to initiate a video capture on Snapper-24. The module must be configured with the required video data source, trigger source, sync source, clock source, etc, before this routine is called.

MODE

SNP24_START_AND_WAIT	The function does not return until capture is complete, including waiting for an external trigger if selected. This parameter is not supported when sequence capture is used. When external trigger is enabled the function returns after a timeout period if no trigger occurs.
SNP24_START_AND_RETURN	The capture is initiated and control immediately returns to the calling function. The controlling program must call *SNP24_is_capture_complete* to determine when the captured has completed. This mode allows the time taken by the hardware to store the video data to be used by the software to perform other processing. This method is useful in multitasking systems, or when the capture and display rate must be optimized. In line scan mode this parameter must always be used to start a capture.
SNP24_START_AND_RETURN_TRIG	This is similar to *SNP24_START_AND_RETURN*; it runs slightly faster but can cause image breakup on slow systems. It is generally used with triggered acquisition. In this mode, a second capture is armed before the current image has been read by a call to *SNP24_read_video_data*. This allows the hardware to start acquisition of the second image during the time to read out the current image, which can increase the acquisition rate. However if the incoming video rate is faster than the readout rate, or the time to start the readout is too large, then the current image will be corrupted by data from the subsequent image.
SNP24_END_SEQUENCE	A capture in sequence mode (previously started by *SNP24_START_AND_RETURN*) will be terminated on completion of acquisition of the current image.
SNP24_ABORT_CAPTURE	A capture previously started by *SNP24_START_AND_RETURN* will be terminated immediately.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

ASLERR_NOT_SUPPORTED	*SNP24_START_AND_WAIT* was requested in sequence mode.
ASLERR_TIMEOUT	The capture timed out. There are two possible causes: A trigger event did not occur within the timeout period although the trigger input is enabled, or The required number of pixels were not acquired within the timeout period probably due to the hardware not being locked to the video source.

EXAMPLES

The following code will capture a single image before processing the acquired data:

```
while (DisplayLive == TRUE)
{
   SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
   SNP24_read_video_data(Hsnp24, Himage, TMG_RUN);
   process_image(hImage, ... );
}
```

The following code will process one image whilst acquiring the next, which is a more efficient use of processing resources.

```
SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
while (DisplayLive == TRUE)
{
   while (SNP24_is_capture_complete(Hsnp24) == FALSE)
   {}
   SNP24_read_video_data(Hsnp24, Himage, TMG_RUN);
   SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
   process_image(hImage, ... );
}
```

The following code will start image acquisition before the current image is read, which can improve acquisition rates but at the risk of image corruption on slow systems.

```
SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
while (DisplayLive == TRUE)
{
   while (SNP24_is_capture_complete(Hsnp24) == FALSE)
   {}
   SNP24_capture(Hsnp24, SNP24_START_AND_RETURN_TRIG);
   SNP24_read_video_data(Hsnp24, Himage, TMG_RUN);
   process_image(hImage, ... );
}
```

BUGS / NOTES

If timeouts are required for capture it is recommended that they are controlled within the application by using the *SNP24_START_AND_RETURN* parameter together with the clock functions available within the operating system in use.

SEE ALSO

SNP24_set_capture, *SNP24_is_capture_complete*, *SNP24_is_sequence_dropped*, *SNP24_is_sequence_mode*, *SNP24_is_trigger_started*, *SNP24_set_trigger*.

SNP24_get_active_area

USAGE

Terr SNP24_get_active_area(Thandle Hsnp24, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hsnp24	Handle to Snapper-24.	
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 *SNP24_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. Similarly *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-24 handle is invalid.

EXAMPLES

To display the active area:

```
SNP24_get_active_area(Hsnp24, 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.

SEE ALSO

SNP24_set_active_area, *SNP24_get_ROI*, *SNP24_get_ROI_max*, *SNP24_set_ROI*.

SNP24_get_ID

USAGE

Terr SNP24_get_ID(Thandle Hsnp24)

ARGUMENTS

Hsnp24	Handle to Snapper-24.

DESCRIPTION

This function returns the hardware identifier of the Snapper, so that an application can check whether it is running on a Snapper-8 or a Snapper-24.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

EXAMPLES

The following code sets the format to 'Y8 on blue' for a Snapper-24, or 'Y8 on red' for a Snapper-8:

```
if ( SNP24_get_ID(Hsnp24) == SNP24_ID )
   SNP24_set_format(Hsnp24, SNP24_FORMAT_Y8_ON_BLU);
else if ( SNP24_get_ID(Hsnp24) == SNP8_ID )
   SNP24_set_format(Hsnp24, SNP24_FORMAT_Y8_ON_RED);
else
   /* Error - should never get here */
```

BUGS / NOTES

The <ID> (either *SNP24_ID* or *SNP8_ID*) 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 *SNP24_get_parameter*.

SEE ALSO

SNP24_get_parameter, *SNP24_get_rev*.

SNP24_get_levels

USAGE

Terr SNP24_get_levels(Thandle Hsnp24, ui8 levels[SNP24_LVL_MAX_LEVELS])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
levels	An array of four elements for the current levels to be returned in. The elements are: *levels[SNP24_LVL_WHITE], levels[SNP24_LVL_BLACK], levels[SNP24_LVL_CLAMP], levels[SNP24_LVL_YCBCR]*.

DESCRIPTION

This function returns the current values of the analogue levels at the input to the ADCs, as set by the most recent call to *SNP24_set_levels*.

SNAPPER-8 DIFFERENCES

The Snapper-8 does not allow adjustment of the clamp level. On a Snapper-8 *levels[SNP24_LVL_CLAMP]* is always returned as *SNP24_LVL_CLAMP_INIT*.

RETURNS

This function returns the current values in the *levels[ ]* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

EXAMPLES

The following code will change the black level, leaving the rest unchanged:

```
ui8 levels[SNP24_LVL_MAX_LEVELS];
ui8 black_level;
...
SNP24_get_levels(Hsnp24, levels);
levels[SNP24_LVL_BLACK] = black_level;
SNP24_set_levels(Hsnp24, SNP24_LVL_SET, levels};
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_set_levels.

SNP24_get_LUTs

USAGE

Terr SNP24_get_LUTs(Thandle Hsnp24, int sel252, ui8 lut[SNP24_SIZE_LUT252])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
sel252	Which channel to read.
lut	An array of 256 elements for the required LUT.

DESCRIPTION

This function returns the current values in the look up table (LUT) between the output of the ADC and the input of the frame store, as set by the most recent call to SNP24_set_LUTs.

SEL252 OPTIONS

This controls which of the three LUTs for the red, green and blue channels are affected by this call. The parameter can be one of *SNP24_252_RED, SNP24_252_GRN* or *SNP24_252_BLU*.

SNAPPER-8 DIFFERENCES

On a Snapper-8 *sel252* must be *SNP24_252_RED*.

RETURNS

This function returns the current values in the *lut[ ]* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_BAD_PARAM	All LUTs have been requested simultaneously, ie *sel252* has been set to *SNP24_252_ALL*.
ASLERR_PARAM_CONFLICT	More than one mode parameter has been passed in, which is not allowed because the parameters are mutually exclusive.
ASLERR_NOT_SUPPORTED	A Snapper-8 is being used, and *sel252* has been set to *SNP24_252_GRN* or *SNP24_252_BLU*.

EXAMPLES

The following code will get the red LUT:

```
ui8 lut[SNP24_SIZE_LUT252]
SNP24_get_LUTs(Hsnp24, SNP24_252_RED, lut);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_set_LUTs

SNP24_get_parameter

USAGE

Terr SNP24_get_parameter(Thandle Hsnp24, ui16 parameter)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
parameter	The parameter to return.

DESCRIPTION

This function returns various parameters from the internal structure associated with the Snapper-24 handle.

PARAMETER

SNP24_BASEBOARD_HANDLE	The handle to the baseboard which the Snapper-24 is fitted on. Type *(ui32)*.
SNP24_TIMEOUT_TRIGGER	This returns the timeout value in milliseconds for the period from when *SNP24_capture* is called to when the trigger is received. Type *(ui32)*.
SNP24_SUPPORTED_FORMAT	This returns the output data format of the Snapper, ie 24 bit for Snapper-24, 8 bit for early Snapper-8 and 16 bit for later Snapper-8 which support data pre-packing for improved transfer rates over the host bus. Type *(ui16)*.
SNP24_MAPPER_INTERFACE_TYPE	This is used as part of the initialisation code (see *SNP24_initialize*) to determine which Mapper type is fitted. After initialisation is complete, it is not required again. Type *(ui16)*.
SNP24_ID_VALUE	This returns the ID as read from the hardware, which distinguishes between all the different Snapper-24 and Snapper-8 variants. Type *(ui8)*.
SNP24_REV_VALUE	This returns the board revision as read from the hardware, which distinguishes between the different hardware revisions of the Snapper. Type *(ui8)*.
SNP24_IDREV_VALUE	This returns the ID and board revision as read from the hardware, which distinguishes between the different hardware revisions of the different Snapper variants. Type *(ui8)*.
SNP24_FAMILY_VALUE	This returns either *SNP24_FAMILY_ID* or *SNP8_FAMILY_ID*, which distinguishes between Snapper-24 and Snapper-8 variants but not between individual boards or specific revision levels. Type *(ui8)*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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

The following example checks user input against the Snapper family to determine whether the hardware supports the request.

```
if ( (SNP24_get_parameter(Hsnp24, SNP24_FAMILY_VALUE) != SNP24_FAMILY_ID) &&
     (UserInput == Acquire24Bits) )
{
   /* Flag an error, as Snapper-8's do not support 24 bit acquisition */
}
```

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

SNP24_get_property, *SNP24_set_parameter*.

SNP24_get_property

USAGE

*Terr SNP24_get_property(Thandle Hsnp24, char *property, char *value)*

ARGUMENTS

Hsnp24	Handle to Snapper-24.
property	A character string or name of the property to access.
value	The property result string. (Must point to a buffer of at least 16 bytes.)

DESCRIPTION

This function returns various property strings associated with Snapper-24.

PROPERTY

"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-24 handle is invalid.
ASLERR_BAD_PARAM	The property value is invalid.

EXAMPLES

To print the FPGA date:

```
char string[256];

SNP24_get_property(Hsnp24, "fpgadate", string);
printf("Snapper-24 FPGA date: %s", string);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_get_parameter.

SNP24_get_rev

USAGE

Terr SNP24_get_rev(Thandle Hsnp24)

ARGUMENTS

Hsnp24	Handle to Snapper-24.

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-24 handle is invalid.

EXAMPLES

```
if ( SNP24_get_rev(Hsnp24) == 0 )
   printf("\nRunning on issue 1 Snapper-24");
else if ( SNP24_get_rev(Hsnp24) == 1 )
   printf("\nRunning on issue 2 Snapper-24");
```

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 *SNP24_get_parameter*.

SEE ALSO

SNP24_get_parameter, *SNP24_get_ID*.

SNP24_get_ROI

USAGE

Terr SNP24_get_ROI(Thandle Hsnp24, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hsnp24	Handle to Snapper-24.	
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* values 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 captured per bank.

RETURNS

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

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

EXAMPLES

To display the current ROI:

```
SNP24_get_ROI(Hsnp24, 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.

SEE ALSO

SNP24_set_ROI, *SNP24_get_ROI_max*, *SNP24_set_active_area*.

SNP24_get_ROI_max

USAGE

Terr SNP24_get_ROI_max(Thandle Hsnp24, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
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* values 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 *SNP24_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 can be captured into one memory bank 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 *SNP24_set_active_area*) and the size of the memory bank.

RETURNS

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

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

EXAMPLES

To set the maximum allowable ROI:

```
SNP24_get_ROI_max(Hsnp24, roi);
SNP24_set_ROI(Hsnp24, SNP24_ROI_SET, roi);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_get_ROI, *SNP24_set_ROI*, *SNP24_set_active_area*.

SNP24_get_subsample

USAGE

Terr SNP24_get_subsample(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function returns the current sub-sample factor, as set by *SNP24_set_capture*.

RETURNS

This function returns the sub-sample ratio, i.e. *SNP24_SUB_X1, SNP24_SUB_X2, SNP24_SUB_X4, SNP24_SUB_X8, SNP24_SINGLE_FIELD,* or *SNP24_SUB_X1_FIELD_DUPLICATE*.

Possible error codes:

ASLERR_BAD_HANDLE The Snapper-24 handle is invalid.

EXAMPLES

```
if ( SNP24_get_subsample(Hsnp24) == SNP24_SUB_X2 )
   printf("\Currently using times 2 sub-sample");
else if ( SNP24_get_subsample(Hsnp24) == SNP24_SUB_X4 )
   printf("\Currently using times 4 sub-sample");
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_set_capture.

SNP24_initialize

USAGE

Terr SNP24_initialize(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required initialization mode.

DESCRIPTION

This function is used to initialize the Snapper-24 module to the required settings for the camera in use.

It makes calls to *SNP24_set_active_area*, *SNP24_set_callback*, *SNP24_set_capture*, *SNP24_set_clamp*, *SNP24_set_clk*, *SNP24_set_format*, *SNP24_set_interrupts*, *SNP24_set_levels*, *SNP24_set_LUTs*, *SNP24_set_ROI*, *SNP24_set_ROI_rounding*, *SNP24_set_trigger*, *SNP24_set_TTL422*, *SNP24_set_video_src* and *SNP24_set_video_standard*. Depending on whether the camera in use is area scan or line scan it also makes calls to some of *SNP24_set_ctrlout*, *SNP24_set_linescan_ctrl*, *SNP24_set_linescan_freq*, *SNP24_set_sync*. See the supplied source of *SNP24_initialize* in "snp24ini.c" to see which modes are set for each camera.

MODE

SNP24_CCIR_DEFAULT	This should be used for a standard area scan camera with CCIR timing, i.e. 768 by 576 image size. For a Snapper-24 this sets up RGB capture to a TMG_RGBX32 image, with sync off video. For a Snapper-8 this sets up mono capture to a TMG_Y8 image, with sync off video. The captured image is a full frame which has not been subsampled.
SNP24_EIA_DEFAULT	This should be used for a standard area scan camera with EIA (RS-170) timing, i.e. 640 by 480 image size. For a Snapper-24 this sets up RGB capture to a TMG_RGBX32 image, with sync off video. For a Snapper-8 this sets up mono capture to a TMG_Y8 image, with sync off video. The captured image is a full frame which has not been subsampled.
SNP24_CUSTOM_LINESCAN	This should be used when using a linescan camera which is not listed above. The function still initializes the Snapper-24 in line scan mode, 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.

Note that all line scan cameras, including the mode *SNP24_CUSTOM_LINESCAN*, put the Snapper-24 into "line scan mode". See the Concepts section of this manual for more details.

See the release notes for details of additional cameras supported.

RETURNS

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

EXAMPLES

```
SNP24_initialize(Hsnp24, SNP24_CCIR_DEFAULT);
```

BUGS / NOTES

IMPORTANT : Line scan mode is not supported in this release of software.

SEE ALSO

-

SNP24_is_50Hz

USAGE

Tboolean SNP24_is_50Hz(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to test whether the current sync source is CCIR (50Hz frame rate) or EIA (60Hz frame rate).

RETURNS

This function returns either *TRUE* (for 50Hz) or *FALSE* (for 60Hz).

EXAMPLES

```
if (SNP24_is_50Hz() == TRUE)
   SNP24_set_video_standard(Hsnp24, SNP24_CCIR_DEFAULT);
else
   SNP24_set_video_standard(Hsnp24, SNP24_EIA_DEFAULT);
```

BUGS / NOTES

The function is not supported in line scan mode.

The function actually compares if the number of lines per field is greater or less than 256.

SEE ALSO

SNP24_set_video_standard, *SNP24_set_clk*, *SNP24_set_sync*.

SNP24_is_capture_complete

USAGE

Tboolean SNP24_is_capture_complete(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to test whether the current video capture has completed. It returns *FALSE* from the time that a capture is initiated with *SNP24_capture* until there is valid data in the video memory. *FALSE* is also returned after the capture has been initiated, but before a valid external trigger has occurred. *TRUE* is returned at all other times.

In sequence mode this function always returns *TRUE* because capture is continuously occurring.

RETURNS

This function returns either *TRUE* (for capture completed) or *FALSE* (for capture not yet completed).

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:

```
SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
while (DisplayLive == TRUE)
{
   while (SNP24_is_capture_complete(Hsnp24) == FALSE)
   {}
   SNP24_read_video_data(Hsnp24, Himage, TMG_RUN);
   SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
   process_image( Himage, ... );
}
SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_capture, *SNP24_is_data_ready*, *SNP24_is_field1_captured*, *SNP24_is_trigger_started*.

SNP24_is_data_ready

USAGE

Tboolean SNP24_is_data_ready(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to test whether data is available to be read from Snapper-24. It returns *TRUE* whenever there is data to be read by *SNP24_read_video_data*, and *FALSE* when data is not yet ready.

It is most useful in sequence mode to tell when a memory bank has just been filled and is ready to read.

RETURNS

This function returns either *TRUE* (for data ready) or *FALSE* (for data not yet ready).

EXAMPLES

The following code can be used in sequence mode, and initiates a sequence capture, then uses *SNP24_is_data_ready* to control when *SNP24_read_video_data* can be called.

```
SNP24_read_video_data(Hsnp24, Hvid_image, TMG_INIT);
SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
while (DisplayLive == TRUE)
{
   while (SNP24_is_data_ready(Hsnp24) == FALSE)
   {}
   SNP24_read_video_data(Hsnp24, Himage, TMG_STRIP);
   process_image( Himage, ... );
}
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_capture, *SNP24_is_capture_complete*, *SNP24_is_field1_captured*, *SNP24_is_trigger_started*.

SNP24_is_field1_captured

USAGE

Tboolean SNP24_is_field1_captured(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

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 *SNP24_START_NEXT_FIELD* is selected in *SNP24_set_capture*.

RETURNS

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

BUGS / NOTES

The function is not supported in line scan mode.

There are no known bugs.

SEE ALSO

SNP24_is_capture_complete, *SNP24_set_capture*, *SNP24_is_field1_incoming*.

SNP24_is_field1_incoming

USAGE

Tboolean SNP24_is_field1_incoming(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to determine whether the selected video source is currently generating field 1 or field 2.

RETURNS

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

BUGS / NOTES

The function is not supported in line scan mode.

There are no known bugs.

SEE ALSO

SNP24_is_field1_captured.

SNP24_is_locked

USAGE

Tboolean SNP24_is_locked(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to determine whether the Snapper-24 is locked to the selected video source. If the Snapper is not locked, then an incorrect sync source is selected or the camera may not be correctly connected.

RETURNS

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

BUGS / NOTES

The function is not supported in line scan mode.

There are no known bugs.

SEE ALSO

-

SNP24_is_sequence_dropped

USAGE

Tboolean SNP24_is_sequence_dropped(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used during or after a sequence mode capture to check whether a continuous (i.e. real time) sequence was captured, or if Snapper-24 had to drop incoming data because the software reading and processing the images could not keep up.

The flag is set to *FALSE* at the time that a sequence capture is initiated. It will remain *FALSE* until the Snapper-24 hardware cannot write into a memory bank because the software is still reading that bank, at which point it is set *TRUE*. The flag is not affected by the termination of capture so it is safe to read it at any point until the next capture is initiated.

Note that even if data is dropped the sequence capture will still continue, it simply means that the resulting sequence of images are not continuous, i.e. they were not captured in real time.

In area scan mode the loss of data means that fields or frames had to be dropped; in line scan mode the loss of data means that at least one bank's worth of lines was dropped. See the Concepts section of this manual for more information.

RETURNS

This function returns either *TRUE* (for field or frames dropped) or *FALSE* (for real time sequence captured).

EXAMPLES

The following code shows the use of *SNP24_is_sequence_dropped* within the sequence loop.

```
while (DisplayLive == TRUE)
{
   while (SNP24_is_data_ready(Hsnp24) == FALSE)
   {}
   if (SNP24_is_sequence_dropped(Hsnp24) == TRUE)
   {
      /* Data has been lost, but both banks of memory contain valid sequence
       * images, so two more SNP24_read_video_data calls can be made before
       * a break occurs in the real time sequence.
       * Put error handling code here - may want to abort immediately; or read
       * the last two real time sequence images and then abort; or log at
       * which image the break occurs in the sequence, and then continue
       */
   }
   SNP24_read_video_data(Hsnp24, Himage, TMG_STRIP);
   process_image( Himage, ... );
}
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_is_sequence_mode.

SNP24_is_sequence_mode

USAGE

Tboolean SNP24_is_sequence_mode(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to test whether Snapper-24 has been set to sequence mode. It returns *TRUE* if *SNP24_set_capture* has been called with parameter *SNP24_SEQUENCE_MODE*, or *FALSE* if *SNP24_set_capture* has been called with parameter *SNP24_SINGLE_CAPTURE_MODE*. Note that this function does not show if a sequence capture is actually taking place, only that Snapper-24 is in a mode where any call to *SNP24_capture* results in a sequence capture occurring.

RETURNS

This function returns either *TRUE* (for sequence mode) or *FALSE* (for single capture mode).

EXAMPLES

The following code would be needed in a routine which is called when a "Freeze" button is pressed in an application, and the routine needs to work in both sequence and single capture modes:

```
if (SNP24_is_sequence_mode() == TRUE)
{
   SNP24_capture(Hsnp24, SNP24_END_SEQUENCE);
   SNP24_read_video_data(Hsnp24, Himage, TMG_RESET);
}
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_is_sequence_dropped, *SNP24_set_capture*.

SNP24_is_trigger_started

USAGE

Tboolean SNP24_is_trigger_started(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to test whether an active edge of the external 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.

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

SNP24_is_capture_complete, *SNP24_set_trigger*.

SNP24_is_vsync_asserted

USAGE

Tboolean SNP24_is_vsync_asserted(Thandle Hsnp24)

ARGUMENTS

Hsnp24 Handle to Snapper-24.

DESCRIPTION

This function is used to test whether the vertical sync (VSync) signal, which is output by the video source between fields, is asserted.

RETURNS

This function returns either *TRUE* during the vsync period or *FALSE* at all other times.

BUGS / NOTES

There are no known bugs.

SEE ALSO

-

SNP24_lm1882_prog

USAGE

Terr SNP24_lm1882_prog(Thandle Hsnp24, Tparam mode, ui16 r1882[SNP24_NUM_LM1882_REGS])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required mode.
r1882	An array of *ui16* elements containing the required register settings.

DESCRIPTION

This is a low level function which programs the 74ACT715 sync generator on Snapper-24. For most applications this function should not be called directly, because *SNP24_set_sync* calls it with either *SNP24_LM1882_INIT_EIA* or *SNP24_LM1882_INIT_CCIR* (when called with *SNP24_SYNC_INIT* or *SNP24_SYNC_INTERNAL*). It is documented to allow the sync generator to be configured to special frequencies or timings.

MODE

SNP24_LM1882_INIT_CCIR	This configures the sync generator to CCIR frequencies.
SNP24_LM1882_INIT_EIA	This configures the sync generator to EIA frequencies.
SNP24_LM1882_SET	This copies the values in *r1882[ ]* to the 74ACT715. *r1882[0]* is written to 74ACT715 register 0, *r1882[1]* to register 1 and so on.

RETURNS

This function returns the register values it has written to the 74ACT715 in the *r1882[ ]* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	More than one mode parameter has been passed in, which is not allowed because the parameters are mutually exclusive.

EXAMPLES

The following code sets the 74ACT715 to default EIA, except that it increases the clocks per line by 4 74ACT715 clocks:

```
ui16 r1882[SNP24_NUM_LM1882_REGS];
...
SNP24_lm1882_prog(Hsnp24, SNP24_LM1882_INIT_EIA, r1882);
/* Register 4 now contains the number of clocks per line */
r1882[4] += 4;
SNP24_lm1882_prog(Hsnp24, SNP24_LM1882_SET, r1882);
```

BUGS / NOTES

There are no known bugs.

This chip was originally named LM1882, hence the name *SNP24_lm1882_prog*.

This function directly accesses the hardware at a low level, and therefore it may not be compatible with future versions of Snapper-24. If it is necessary to use the function it is recommended that all calls to it are put into one function, e.g. *SNP24_set_sync_my_camera( )* so that any future changes which need to be made are restricted to the one function.

Support for non-CCIR or EIA cameras may be added to the library as additional mode parameters, therefore it may be worthwhile to check this with your distributor before writing low level code.

The National Semiconductor datasheet for the 74ACT715 gives full details of the internal registers. The 74ACT715 is driven by a 29.50MHz crystal in CCIR mode, or a 14.31818MHz crystal in EIA mode. The crystal is selected by *SNP24_set_video_standard*.

This function is not supported in line scan mode.

SEE ALSO

SNP24_set_sync.

SNP24_read_video_data

USAGE

Terr SNP24_read_video_data(Thandle Hsnp24, Thandle Himage, ui16 TMG_action)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
Himage	Handle to image.
TMG_action	*TMG* mode flag.

DESCRIPTION

This function reads data from Snapper-24 video memory into the image structure referenced by *Himage*.

When in single capture mode the data must already have been stored into video memory with a call to *SNP24_capture*, and *Himage* must have been set up with a call to *SNP24_set_image*. The *TMG_action* flag would normally be set to *TMG_RUN*. As the Snapper-24 memory may contain a large amount of data, the imaging software can be run in strips to reduce the amount of system memory required at any one time. Because of the strip technique it is possible to abort an image transfer at any time with the *TMG_action* flag. Refer to the Concepts section of the TMG Programmer's Manual for more details on strip processing.

When in sequence mode the function is called in three stages by using the *TMG_action* flags *TMG_INIT*, *TMG_STRIP* and *TMG_RESET*. The *Himage* must have been set up with a call to *SNP24_set_image* before calling the function with mode *TMG_INIT*.

TMG_ACTION

TMG_RUN	This parameter should be used when in single capture mode.
TMG_INIT	This parameter is used to initialize the read video data routine before starting a sequence mode capture.
TMG_STRIP	This parameter is used to read one image during a sequence mode capture. It would normally be called multiple times while the sequence capture is running.
TMG_RESET	This parameter is used to reset the read video data routine after the completion a sequence mode capture. It can also be used in single capture mode to reset the routine if an image is being processed in a strip loop and the strip loop is aborted.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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

The following code does a capture, and then reads the new image, in single capture mode:

```
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
SNP24_read_video_data(Hsnp24, Himage1, TMG_RUN);
```

See the sequence mode example in the Sample Applications section at the front of this manual for an example of the use of modes *TMG_INIT*, *TMG_STRIP*, and *TMG_RESET*.

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_reset_read_pointer.

SNP24_reset_read_pointer

USAGE

Terr SNP24_reset_read_pointer(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required read pointer to reset.

DESCRIPTION

This function resets the read pointers to the video memory. This is used when an application requires to re-read the data in video memory.

MODE

SNP24_READ_RESET_AUTO This is the only supported parameter.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

EXAMPLES

The following code does a capture, processes the image and then re-reads the video memory:

```
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
SNP24_read_video_data(Hsnp24, Himage, TMG_RUN);
process_image(Himage, ... );
SNP24_read_reset_pointer(Hsnp24, SNP24_READ_RESET_AUTO);
SNP24_read_video_data(Hsnp24, Himage, TMG_RUN);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_read_video_data.

SNP24_set_active_area

USAGE

Terr SNP24_set_active_area(Thandle Hsnp24, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hsnp24	Handle to Snapper-24.	
roi	ROI array with four elements, with #defined element names:	
	ASL_ROI_X_START	First valid pixel number after line enable (0 = left of image).
	ASL_ROI_Y_START	First valid line number after frame enable (0 = top of image).
	ASL_ROI_X_LENGTH	Number of valid pixels.
	ASL_ROI_Y_LENGTH	Number of valid lines.

DESCRIPTION

This function defines the active area for the camera in use. This is used to allow *SNP24_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 *SNP24_initialize* calls it to set the active area of the selected camera.

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.

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.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_OUT_OF_RANGE	The active area is too large for Snapper-24, or includes negative coordinates, or for area scan cameras the active area is too large compared with the setting of pixels per line.

EXAMPLES

For an area scan 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 */
SNP24_set_active_area(Hsnp24, roi);
```

For an area scan camera which starts outputting data on pixel 0 of line 0, and continues outputting until pixel 523 of line 250:

```
roi[ASL_ROI_X_START] = 0;
roi[ASL_ROI_Y_START] = 0;
roi[ASL_ROI_X_LENGTH] = 523;
```

```
roi[ASL_ROI_Y_LENGTH] = 250
SNP24_set_active_area(Hsnp24, roi);
```

For a line scan camera which starts outputting data on pixel 10, and continues outputting until pixel 4050:

```
roi[ASL_ROI_X_START] = 10;
roi[ASL_ROI_Y_START] = 0;
roi[ASL_ROI_X_LENGTH] = 4041;    /* i.e. 4050 - 10 + 1 */
roi[ASL_ROI_Y_LENGTH] = 0
SNP24_set_active_area(Hsnp24, 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.

SEE ALSO

SNP24_set_ROI, *SNP24_get_ROI_max*, *SNP24_set_ROI_rounding*.

SNP24_set_callback

USAGE

*Terr SNP24_set_callback(Thandle Hsnp24, Tparam mode, void (EXPORT_FN *callback)(Thandle, ui32, void*), void *parameter)*

ARGUMENTS

Hsnp24	Handle to Snapper-24.
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 *SNP24_set_interrupts*.

MODE

SNP24_CALLBACK_INIT	Callback functions are disabled. *callback* and *parameter* should be passed as *NULL*. This is the default as set by *SNP24_initialize*.
SNP24_CALLBACK_SET	Function *callback* together with parameter *parameter* are installed as the callback function.

CALLBACK FUNCTION DEFINITION

*void callback(Thandle Hdig24, ui32 int_source, void *parameter)*

Hsnp24	Handle to Snapper-24. The library sets this to the Snapper-24 which caused the interrupt (i.e. there may be more than one Snapper-24 in the system).
int_source	The interrupt source which has interrupted. The library sets this using the #define parameters as defined in *SNP24_set_interrupts*. These parameters are tested using bitwise operators.
Parameter	The library sets this to the user defined parameter *parameter*, as set by *SNP24_set_callback*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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 read data out of Snapper-24 when data is available (i.e. when acquisition has completed):

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

/* The interrupt handler, which reads out data and saves it to a file */
```

```
void EXPORT_FN MyCallback(Thandle hSnapper, ui32 dwIntSrc, void *pMyData)
{
   /* Test on the correct interrupt */
   if (!(dwIntSrc & SNP24_INT_DATA_READY))
      return;  /* Unexpected interrupt - we could set an error flag here */

   SNP24_read_video_data(hSnapper, hImage, TMG_RUN);
   TMG_image_set_outfilename(hImage, (char*) pMyData);
   TMG_image_write(hImage, NULL, TMG_TIFF, TMG_RUN);
}


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

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

/* Enable the callback function */
SNP24_set_callback(hSnapper, SNP24_CALLBACK_SET, MyCallback, szFilename);

/* Enable transfer complete interrupt, so that the acquired image will
 * automatically get saved as a TIFF file.
 */
SNP24_set_interrupts(hSnapper, SNP24_INT_DATA_READY, 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

SNP24_set_interrupts.

SNP24_set_capture

USAGE

Terr SNP24_set_capture(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required capture mode.

DESCRIPTION

This function configures Snapper-24 for different types of image capture. It accepts up to four different orthogonal options which can be 'OR'ed together. For many applications this function need not be called directly, because *SNP24_initialize* calls it with the parameter *SNP24_CAPTURE_INIT*.

INITIALIZATION

SNP24_CAPTURE_INIT	The first call made to *SNP24_set_capture* must include this parameter. It selects *SNP24_SUB_X1*, *SNP24_START_1ST_FIELD*, *SNP24_TRIG_IN_DISABLE*, *SNP24_SINGLE_CAPTURE_MODE* and *SNP24_READOUT_NO_DEINTERLACE*.

TRIGGER CONTROL

Capture can be delayed until an external trigger event happens:

SNP24_TRIG_IN_ENABLE	Image capture requested by *SNP24_capture* will not start until an active edge of the selected trigger occurs. See *SNP24_set_trigger* for information on control of the trigger source.
SNP24_TRIG_IN_DISABLE	Image capture requested by *SNP24_capture* will occur independent of the trigger status.

Note that if the trigger is infrequent then the capture timeout parameter *SNP24_TIMEOUT_TRIGGER* will need to be extended - see *SNP24_set_parameter*.

DEINTERLACE CONTROL

When a frame of video data is captured the image can be de-interlaced automatically as it is read from video memory (for use with conventional area scan cameras, not line scan cameras). Note that this feature is not available when sequence mode is enabled.

SNP24_READOUT_DEINTERLACE	The image will be de-interlaced automatically as it is read from video memory.
SNP24_READOUT_NO_DEINTERLACE	The image will not be de-interlaced automatically as it is read from video memory, so the fields will be preserved as they were captured.

SUB-SAMPLE CONTROL (AREA SCAN MODE)

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

SNP24_SUB_X1	This enables full resolution image capture of a complete frame.
SNP24_SUB_X1_FIELD_DUPLICATE	This enables full screen image capture at half vertical resolution. This is achieved by capturing only 1 field, but replicating the image data in software by repeating each line. This halves the vertical resolution, but allows fast movement to be captured

without the 'double image' which would result if movement was captured in normal full frame mode from a field exposure camera. If Snapper-24 is used in a system with a slow bus compared to the processor speed (e.g. most ISA systems) then this mode will allow faster transfer rates than *SNP24_SUB_X1*. Conversely, in a system with a fast bus which supports DMA (e.g. PCI or SBus systems) this mode will be slower than *SNP24_SUB_X1*. Field duplication is not supported in sequence mode.

SNP24_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.

SNP24_SUB_X2 — This enables sub-sampled by 2 image capture. This is achieved by capturing every other pixel in the horizontal direction and only one field in the vertical direction.

SNP24_SUB_X4 — This enables sub-sampled by 4 image capture. This is achieved by capturing every fourth pixel in the horizontal direction and every other line of one field in the vertical direction.

Note: If the subsample ratio is changed this function calls *SNP24_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 *SNP24_set_ROI_rounding*. Therefore if an application keeps its own copy of the current ROI is should call *SNP24_get_ROI* to update its copy after calling *SNP24_set_capture*.

SUB-SAMPLE CONTROL (LINE SCAN MODE)

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

SNP24_SUB_X1 — This enables full resolution image capture of a complete line.

SNP24_SUB_X2 — This enables sub-sampled by 2 image capture. This is achieved by capturing every other pixel in the line.

SNP24_SUB_X4 — This enables sub-sampled by 4 image capture. This is achieved by capturing every fourth pixel in the line.

SNP24_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 *SNP24_set_linescan_ctrl* controls the line acceptance ratio independently from the sub-sample ratio.

Note: If the subsample ratio is changed this function calls *SNP24_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 *SNP24_set_ROI_rounding*. Therefore if an application keeps its own copy of the current ROI is should call *SNP24_get_ROI* to update its copy after calling *SNP24_set_capture*.

INITIAL FIELD CONTROL

There are 2 different field control modes for use with conventional area scan cameras:

SNP24_START_1ST_FIELD — Capture starts with a field 1. If a frame is being captured Snapper-24 will wait for a field 1, and then capture this field 1 and the following field 2. If a field is being captured Snapper-24 will wait for a field 1, and then only capture this field.

SNP24_START_NEXT_FIELD — Capture starts with the next field, either field 1 or field 2. If a frame is being captured Snapper-24 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-24 will only capture the next field - either field 1 or field 2.

Note: *SNP24_set_sync* can be called with the parameter *SNP24_SYNC_FIELDS_SWAP* to allow *SNP24_START_1ST_FIELD* to be turned into *"SNP24_START_2ND_FIELD"*.

SEQUENCE CONTROL

SNP24_capture can either capture one image or many images:

SNP24_SINGLE_CAPTURE_MODE Image capture requested by *SNP24_capture* will terminate after one field or frame is acquired.

SNP24_SEQUENCE_MODE Snapper-24 is put into sequence mode where image capture requested by *SNP24_capture* will continuously capture fields or frames. See the Concepts section at the front of this manual for more information on using sequence mode.

The following table details the behaviour of Snapper-24 for the various sequence mode, initial field and sub sample options:

	Sequence Mode	Start 1st Field	Sub x1	Description
(a)				Acquire the next field, either field 1 or field 2, then stop.
(b)			✔	Acquire the next two fields, in either order, then stop. The data will be stored as a single frame of data with the fields in the correct order. However this may consist of field 2 from one frame interlaced with field 1 of the following frame.
(c)		✔		Acquire field 1 only, then stop. As Snapper-24 will have to wait until the next field 1 to start acquisition, the average acquisition rate is likely to be less than (b).
(d)		✔	✔	Acquire field 1 followed by field 2, then stop. This will ensure that both fields are from the same frame, but as Snapper-24 will have to wait until the next field 1 to start acquisition, the average acquisition rate is likely to be less than (a).
(e)	✔			Start acquiring on the next field, either field 1 or field 2, and continue to acquire the same subsequent field until the capture is terminated. Use *SNP24_is_field1_captured* to determine which field has been acquired.
(f)	✔		✔	Start acquiring on the next field, either field 1 or field 2, and continue to acquire every subsequent field until the capture is terminated. Use *SNP24_is_field1_captured* to determine whether the first field acquired was field 1 or field 2. Snapper-24 ensures that if the system cannot read data fast enough, that pairs of fields will be lost to ensure the correct field ordering at all times.
(g)	✔	✔		Start acquiring on the next field 1, and continue to acquire each subsequent field 1 until the capture is terminated.
(h)	✔	✔	✔	Start acquiring on the next field 1, and continue to acquire all subsequent fields until the capture is terminated. Snapper-24 ensures that if the system cannot read data fast enough, that pairs of fields will be lost to ensure the correct field ordering at all times.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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. No change is made to the settings of Snapper-24.

EXAMPLES

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

```
SNP24_set_capture(Hsnp24, SNP24_START_1ST_FIELD | SNP24_SUB_X2);
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_capture, *SNP24_is_sequence_mode*, *SNP24_set_trigger*.

SNP24_set_clamp

USAGE

Terr SNP24_set_clamp(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required clamp mode.

DESCRIPTION

This function controls the Snapper-24 video clamp timing. It accepts two different orthogonal options which can be 'OR'ed together. For many applications this function need not be called directly, because *SNP24_initialize* calls it with the parameters *SNP24_CLAMP_INIT*.

MODE

SNP24_CLAMP_INIT	The clamp position is set to a default value (which is #defined as *SNP24_DEFAULT_CLAMP_START*), which is correct for standard area scan video formats at square pixel sampling frequencies.
SNP24_CLAMP_START	This parameter allows the default clamp position to be changed. The required number of four pixel clocks is 'OR'ed with this parameter, e.g. (SNP24_CLAMP_START | 15). The ROI will need changing after calling this function because the ROI *x_start* position is referenced to the end of the clamp pulse.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_OUT_OF_RANGE	The value passed with *SNP24_CLAMP_START* is too big or too small.

EXAMPLES

The following code will set the clamp start to a value of 80 pixel clocks, (ie 4 x 20):

```
SNP24_set_clamp(Hsnp24, SNP24_CLAMP_START | 20);
```

BUGS / NOTES

There are no known bugs.

There is currently no control of clamp width, but the default width should be suitable.

SEE ALSO

SNP24_set_ROI.

SNP24_set_clk

USAGE

Terr SNP24_set_clk(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required clock mode.

DESCRIPTION

This function controls Snapper-24's phase locked loop (PLL) and 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 *SNP24_initialize* calls it with the *SNP24_CLK_INIT* parameter.

MODE

SNP24_CLK_INIT
The first call made to *SNP24_set_clk* must include this parameter. This sets the PLL to CCIR or EIA operation based on the most recent call to *SNP24_set_video_standard*, selects fast lock, selects the PLL as the clock source, and sets PCLK as an input.

SNP24_PLL_FAST_LOCK
This sets the PLL so that it locks very quickly to a stable sync source. It may not lock to a poor sync such as from a VCR.

SNP24_PLL_SLOW_LOCK
This sets the PLL into a mode where it takes a few seconds to achieve lock. This allows Snapper-24 to lock to poor syncs such as from a VCR. It will also result in slightly lower pixel jitter on stable sync sources, and is therefore recommended for any application where the initial few seconds lock time is acceptable.

SNP24_PLL_PHASE_0,
SNP24_PLL_PHASE_90,
SNP24_PLL_PHASE_180,
SNP24_PLL_PHASE_270
These four parameters vary the time between the active edge of the selected horizontal sync and the active pixel clock edge. As examples, phase 0 makes the two active edges line up, phase 180 makes the horizontal sync edge half way between active edges of pixel clock. *SNP24_CLK_INIT, SNP24_PLL_FAST_LOCK* and *SNP24_PLL_SLOW_LOCK* all select the optimum value of phase for normal use. The phase can adjusted by experimentation to produce the minimum noise from a given camera, that is when the active pixel clock edge is in the middle of each pixel value output from the camera.

SNP24_PLL_VCO_GAIN
This parameter overrides the default setting of the voltage controlled oscillator (VCO) gain in the PLL. The required setting should be 'OR'ed with this parameter, e.g. (*SNP24_PLL_VCO_GAIN* | 3). The setting is a number between 0 and 7, with a higher number corresponding to a higher gain. Recommended values depend on the pixel clock frequency:

Pixel clock, MHz	VCO_GAIN
1.9 ... 5	0
4.4 ... 7.5	1
5.6 ... 10	2
6.9 ... 12.5	3
7.5 ... 16	4
16 ... 20	5

The values 6 and 7 should not be needed. Note that the PLL may not lock

	with some values of VCO gain.
SNP24_PLL_PD_GAIN	This parameter overrides the default setting of the phase detector gain in the PLL. The required setting should be 'OR'ed with this parameter, e.g. (*SNP24_PLL_PD_GAIN* | 3). The setting is a number between 0 and 7, with a higher number corresponding to a higher gain. Note that the PLL may not lock with some values of phase detector gain, but the default value should work for most applications.
SNP24_CLK_PLL	This selects the output of the PLL as Snapper-24's pixel clock.
SNP24_CLK_PCLK_IN_POS	This selects the rising edge of the external PCLK signal as Snapper-24's pixel clock.
SNP24_CLK_PCLK_IN_NEG	This selects the falling edge of the external PCLK signal as Snapper-24's pixel clock.
SNP24_CLK_PCLK_OUT_POS	This drives the external PCLK signal with Snapper-24's internal pixel clock.
SNP24_CLK_PCLK_OUT_NEG	This drives the external PCLK signal with Snapper-24's internal pixel clock inverted.

PARAMETER INTERACTION

When combinations of parameters are passed in one call to *SNP24_set_clk* 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_HANDLE	The Snapper-24 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. See the examples given below. No change is made to the setting of the clock circuits.
ASLERR_NOT_RECOGNIZED	This occurs when *SNP24_CLK_INIT* is passed and the video standard setting is not recognised, possibly because *SNP24_set_video_standard* has not been called.
ASLERR_OUT_OF_RANGE	This occurs when the numeric value passed with a parameter such as *SNP24_PLL_VCO_GAIN* is invalid.

EXAMPLES

In the following call the slow lock option will override the default of fast lock set by *SNP24_CLK_INIT*:

```
SNP24_set_clk(Hsnp24, SNP24_CLK_INIT | SNP24_PLL_SLOW_LOCK);
```

In the following call a VCO gain of 2 and a clock phase of 270° will override the default settings of *SNP24_CLK_INIT*:

```
SNP24_set_clk(Hsnp24, (SNP24_VCO_GAIN | 2) | SNP24_CLK_INIT |
       SNP24_PLL_CLOCK_PHASE_270);
```

The following call will result in a *ASLERR_PARAM_CONFLICT* error because the PLL and the PCLK input cannot both provide the pixel clock at the same time:

```
SNP24_set_clk(Hsnp24, SNP24_CLK_PLL | SNP24_CLK_PCLK_IN_NEG);
```

The following call will also result in a *ASLERR_PARAM_CONFLICT* error. This is because both parameters require a value to be passed, which cannot be done by 'OR'ing the values:

```
SNP24_set_clk(Hsnp24, (SNP24_PLL_VCO_GAIN | 2) | (SNP24_PLL_PD_GAIN | 3));
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_set_pix_per_line, *SNP24_set_sync*, *SNP24_set_TTL422*, *SNP24_set_video_standard*.

SNP24_set_ctrlout

USAGE

Terr SNP24_set_ctrlout(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required control output mode.

DESCRIPTION

This function allows the VSYNC and pixel clock pins to be used as control outputs in applications where they are not being used as normal sync and clock. For many applications this function need not be called directly, because *SNP24_initialize* calls it with the parameter *SNP24_CTRLOUT_INIT*.

The mode parameter should be one of the following:

MODE

SNP24_CTRLOUT_INIT	This sets the control out pins in their default use as sync and clock signals, and initialises the relevant internal structure settings.
SNP24_CTRLOUT_SYNC	This sets the control out pins in their default use as sync and clock signals.
SNP24_CTRLOUT_SET	This sets the control out pins to the specified value. The value is passed by 'OR'ing it with this parameter, e.g. (*SNP24_CTRLOUT_SET* | 2), and must be between 0 and 3. VSYNC and pixel clock must be set as outputs of the required polarity by calls to *SNP24_set_sync* and *SNP24_set_clk*. VSYNC is the LSB and pixel clock the MSB.
SNP24_CTRLOUT_VIDSRC	The control out pins are driven with a binary pattern based on the most recent call to *SNP24_set_video_src*. Subsequent calls to *SNP24_set_video_src* will cause the control out pins to be updated automatically. This automatic mode is cancelled by calling *SNP24_set_ctrlout* with a different parameter. The dual use sync/clk/trigger pins must be enabled as outputs of the required polarity - see above.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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_OUT_OF_RANGE	The value passed with *SNP24_CTRLOUT_SET* is greater than 3.

EXAMPLES

To set the pixel clock pin high and the VSync pin low. Note that *SNP24_set_ctrlout* is called before the pins are enabled as outputs - otherwise a short burst of sync and clock would appear on the outputs.

```
SNP24_set_ctrlout(Hsnp24, SNP24_CTRLOUT_SET | 2);
SNP24_set_sync(Hsnp24, SNP24_SYNC_VSYNC_OUT_POS);
SNP24_set_clk(Hsnp24, SNP24_CLK_PCLK_OUT_POS);
```

BUGS / NOTES

There are no known bugs. Note that if three programmable outputs are required the trigger pin can also be used.

SEE ALSO

SNP24_set_clk, *SNP24_set_sync*, *SNP24_set_trigger*, *SNP24_set_TTL422*.

SNP24_set_format

USAGE

Terr SNP24_set_format(Thandle Hsnp24, Tparam snap_format, ui16 TMG_format)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
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-24 video memory and in the host computer's memory. *SNP24_initialize* calls it with the parameter *SNP24_FORMAT_INIT*.

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:

SNP24_FORMAT_INIT	The first call made to *SNP24_set_format* must include this parameter. This sets the format to *SNP24_FORMAT_RGB* for a Snapper-24, or *SNP24_FORMAT_Y8_ON_RED* for a Snapper-8. It also overrides the *TMG_format* to be *TMG_RGBX32* for a Snapper-24, or *TMG_Y8* for a Snapper-8.
SNP24_FORMAT_RGB	Video data will be read from the red, green and blue ADCs. If sync off video is selected, the sync source will be switched to the current green video source.
SNP24_FORMAT_Y8_ON_RED	Video data will be read from the red ADC. If sync off video is selected, the sync source will be switched to the current red video source. This is the only monochrome format which is supported when Snapper-24 is used on the ISA-BIB or ISA-JPG baseboards, as these baseboards do not have a data mapper.
SNP24_FORMAT_Y8_ON_GRN	Video data will be read from the green ADC. If sync off video is selected, the sync source will be switched to the current green video source.
SNP24_FORMAT_Y8_ON_BLU	Video data will be read from the blue ADC. If sync off video is selected, the sync source will be switched to the current blue video source.

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.

SNAPPER-8 DIFFERENCES

The Snapper-8 does not have a green or blue channel, so the only valid parameters are *SNP24_FORMAT_INIT* and *SNP24_FORMAT_Y8_ON_RED*.

IMPORTANT: When writing code for a Snapper-8 which must also run on a Snapper-24, note that the default format selected by *SNP24_FORMAT_INIT* will be different. Therefore the application must call *SNP24_set_format* with *SNP24_FORMAT_Y8_ON_RED* selected after calling *SNP24_initialize*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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	Either: A Snapper-8 is being used, and an option other than *SNP24_FORMAT_INIT* or *SNP24_FORMAT_Y8_ON_RED* has been requested, or: 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 sets the format to 'Y8 on blue' for a Snapper-24, or 'Y8 on red' for a Snapper-8:

```
if ( SNP24_get_ID(Hsnp24) == SNP24_ID )
   SNP24_set_format(Hsnp24, SNP24_FORMAT_Y8_ON_BLU, TMG_Y8);
else if ( SNP24_get_ID(Hsnp24) == SNP8_ID )
   SNP24_set_format(Hsnp24, SNP24_FORMAT_Y8_ON_RED, TMG_Y8);
else
   /* Error - should never get here */
```

The following code sets the snapper format to 'Y8 on blue', and maps the output to the format *TMG_RGB16* which is suitable for rapid display using both MS-DOS and Microsoft Windows. This conversion is done by the data mapper reading monochrome data from the blue ADC, duplicating this data into its red and blue channels, and finally trimming the resulting 24 bit value down to 16 bits by discarding the lower bits:

```
SNP24_set_format(Hsnp24, SNP24_FORMAT_Y8_ON_BLU, TMG_RGB16);
```

BUGS / NOTES

When a capture is done on a Snapper-24 board with the format set to, for example, SNP24_FORMAT_Y8_ON_RED, the data in the green and blue video memory still gets updated.

If multiple monochrome cameras are needed on a Snapper-8, or an ISA-BIB or ISA-JPG combination is used which does not have a data mapper remember that *SNP24_set_video_src* can be used to select between up to four cameras connected to the red inputs without getting *ASLERR_UNSUPPORTED* problems.

SEE ALSO

-

SNP24_set_image

USAGE

Terr SNP24_set_image(Thandle Hsnp24, Thandle Himage)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
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 (see the TMG Programmer's Manual for an explanation of strip processing).

This function must be called after *SNP24_initialize* has been called, or after the width or height of the ROI, the sub-sample ratio, or the image format has changed, or the capture mode has been changed between sequence and single capture mode, but before *SNP24_read_video_data* is called.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_NOT_RECOGNIZED	The Snapper-24 sub-sample ratio is invalid.

EXAMPLES

```
/* Change the ROI ... */
SNP24_set_ROI(Hsnp24, SNP24_ROI_SET, roi);
/* ... therefore call set image before capturing ... */
SNP24_set_image(Hsnp24, Himage);
/* ... finally acquire the image */
SNP24_capture_to_image(Hsnp24, Himage, SNP24_START_AND_WAIT);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

-

SNP24_set_interrupts

USAGE

Terr SNP24_set_interrupts(Thandle Hsnp24, Tparam type, Tboolean flag)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
type	Required interrupt to control.
flag	*TRUE* / *FALSE* setting for requested interrupt.

DESCRIPTION

This function controls the generation of events and interrupts from Snapper-24. When an enabled interrupt occurs, the function installed by *SNP24_set_callback* is called. Multiple interrupts are set by bitwise ORing the interrupt options in a single call to *SNP24_set_interrupts*.

TYPE

SNP24_INT_INIT	All interrupts are disabled. The flag parameter is ignored. This is the default set by *SNP24_initialize*.
SNP24_INT_VSYNC	When enabled, an interrupt is generated on a video vertical synchronisation pulse (i.e. field sync).
SNP24_INT_DATA_READY	When enabled, an interrupt is generated when an image has been acquired into on-board frame memory. This may be a frame, field or ROI, depending on the current Snapper mode.
SNP24_INT_TRANSFER_COMPLETE	When enabled, an interrupt is generated when all data transfer has completed from the Snapper on-board memory in host memory.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper handle is invalid.
ASLERR_BAD_PARAM	The type parameter is invalid.
ASLERR_NOT_SUPPORTED	A mode other than *SNP24_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 *SNP24_set_callback*, starts acquisition and then processes the image data when signalled to do so from the interrupt thread (using typically a semaphore - Win32 API examples are used here). Full error checking is not shown for simplicity.

```
/* Setup Snapper-24 as appropriate */
…
/* Get valid image data into both buffers to start us off */
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
SNP24_read_video_data(Hsnp24, Himage1, TMG_RUN);
```

```
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
SNP24_read_video_data(Hsnp24, Himage2, TMG_RUN);

/* Install the interrupt handler and enabled the appropriate interrupt */
SNP24_set_callback(Hsnp24, SNP24_CALLBACK_SET, MyCallback, NULL);
SNP24_set_interrupts(Hsnp24, SNP24_INT_DATA_READY, TRUE);

/* Start interrupt driven acquisition */
SNP24_capture(Hsnp24, START_AND_RETURN);

/* This main program thread now waits on a semaphore and then processes the
 * images when signalled from the interrupt handler.
 */

while (!bFinished)   /* A global could be used to signal when finished */
{
   /* Wait until image 1 has been acquired into host memory */
   WaitForSingleObject(hSemaImageReady, 500);  /* Win32 API */

   /* Process image 1 (at this point, image 2 is being acquired) */
   ProcessImage(Himage1);

   /* Now wait for image 2 */
   WaitForSingleObject(hSemaImageReady, 500);

   /* Process image 2 (at this point, image 2 is being acquired) */
   ProcessImage(Himage2);
}

/* Disable interrupt handler */
SNP24_set_callback(Hsnp24, SNP24_CALLBACK_INIT, NULL, NULL);
```

The interrupt handler, called on image acquisition complete (i.e. image data acquired into on-board memory), then reads out the image data from Snapper, signals to the processing thread and starts acquisition of the next image. The pre-processor directive *_CALLBACK_SEQ* shows how sequence mode would be supported.

```
/* This is our interrupt handler */
void InterruptHandler(Thandle Hsnp24, ui32 dwIntSrc, void *pMyData)
{
   static ui32 dwBuffer = 1;

   if ( dwBuffer == 1 )
   {
#ifdef _CALLBACK_SEQ
      SNP24_read_video_data(Hsnp24, Himage1, TMG_STRIP);
#else
      SNP24_read_video_data(Hsnp24, Himage1, TMG_RUN);
      SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
#endif
      dwBuffer = 2;
   }
   else
   {
#ifdef _CALLBACK_SEQ
      SNP24_read_video_data(Hsnp24, Himage2, TMG_STRIP);
#else
      SNP24_read_video_data(Hsnp24, Himage2, TMG_RUN);
      SNP24_capture(Hsnp24, SNP24_START_AND_RETURN);
#endif
      dwBuffer = 1;
   }
   ReleaseSemaphore(hSemaImageReady, 1, NULL);  /* Win32 API */
}
```

BUGS / NOTES

If adding or removing interrupts, disable interrupts using *SNP24_set_callback* (to *NULL*) first and then re-enable after having called *SNP24_set_interrupts*.

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

SEE ALSO

SNP24_set_callback.

SNP24_set_levels

USAGE

Terr SNP24_set_levels(Thandle Hsnp24, Tparam mode, ui8 levels[SNP24_LVL_MAX_LEVELS])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required mode.
levels	An array of four elements for the required levels. The elements are: *levels[SNP24_LVL_WHITE], levels[SNP24_LVL_BLACK], levels[SNP24_LVL_CLAMP], levels[SNP24_LVL_YCBCR].*

DESCRIPTION

This function controls the analogue levels at the input to the ADCs. These are white level, black level, clamp level and YCbCr level. For many applications this function need not be called directly, because *SNP24_initialize* calls it with the *SNP24_LVL_INIT* parameter.

LEVELS

Black level sets the input video voltage which will result in the ADCs generating an output 0x00; similarly white level sets the input video voltage which will result in the ADCs generating an output 0xFF. Clamp level sets the voltage at which DC restoration is performed, i.e. the voltage to which the back porch of the video is set to. YCbCr level only affects SNP-24-YCbCr modules, and it allows the DC restoration voltage for the Cb and Cr components of the video to be set midway between white level and black level.

MODE

SNP24_LVL_INIT	This sets the levels to default values for RGB or monochrome video, which are *SNP24_LVL_WHITE_INIT, SNP24_LVL_BLACK_INIT, SNP24_LVL_CLAMP_INIT,* and *SNP24_LVL_YCBCR_INIT*. The contents of *levels[ ]* passed in are ignored. This gives a maximum dynamic range corresponding to black = 0 and white = 255.
SNP24_LVL_INIT_601	This sets the levels to CCIR601 values for RGB or monochrome video, which are *SNP24_LVL_WHITE_INIT_601, SNP24_LVL_BLACK_INIT_601, SNP24_LVL_CLAMP_INIT_601,* and *SNP24_LVL_YCBCR_INIT_601*. The contents of *levels[ ]* passed in are ignored. This gives a dynamic range corresponding to black = 16 and white = 235.
SNP24_LVL_INIT_YCBCR	This sets the levels to default values for YCbCr video, which are *SNP24_LVL_WHITE_INIT_YCBCR, SNP24_LVL_BLACK_INIT_YCBCR, SNP24_LVL_CLAMP_INIT_YCBCR,* and *SNP24_LVL_YCBCR_INIT_YCBCR*. The contents of *levels[ ]* passed in are ignored.
SNP24_LVL_SET	This sets the levels to the values in the *levels[ ]* array.

PRE-PROCESSING

The function checks the levels before writing them to the hardware. This is because some levels could result in the hardware getting into an invalid state. As an example the white level voltage must be greater than the black level voltage. Also the two least significant bits of the values must be zero.

To simplify the use of the function with interactive software (e.g. level sliders in a GUI application) the function does not treat these invalid levels as an error. Instead it adjusts the levels to prevent the problem, and then returns the actual levels set.

SNAPPER-8 DIFFERENCES

The Snapper-8 does not allow adjustment of the clamp level, and YCbCr video cannot be digitized. A Snapper-8 behaves as if *levels[SNP24_LVL_CLAMP]* has been set to *SNP24_LVL_CLAMP_INIT*.

RETURNS

This function returns the actual levels values used (i.e. after any pre-processing) in the *levels[ ]* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	More than one mode parameter has been passed in, which is not allowed because the parameters are mutually exclusive.
ASLERR_NOT_SUPPORTED	A Snapper-8 is being used, and *SNP24_LVL_INIT_YCBCR* has been requested.

EXAMPLES

The following code will set default levels:

```
ui8 levels[SNP24_LVL_MAX_LEVELS];
...
SNP24_set_levels(Hsnp24, SNP24_LVL_INIT, levels};
printf("White level set to %d\n", levels[SNP24_LVL_WHITE]);
```

The following code will change the white level, leaving the rest at default levels:

```
ui8 levels[SNP24_LVL_MAX_LEVELS];
ui8 white_level;
...
SNP24_set_levels(Hsnp24, SNP24_LVL_INIT, levels};
levels[SNP24_LVL_WHITE] = white_level;
printf("White level passed as %d\n", levels[SNP24_LVL_WHITE]);
SNP24_set_levels(Hsnp24, SNP24_LVL_SET, levels};
printf("White level actually set to %d\n", levels[SNP24_LVL_WHITE]);
```

BUGS / NOTES

There are no known bugs.

Detail notes on the levels: The resolution of the level settings is only six bits because the two LSBs are ignored. For YCbCr operation the Clamp level is increased to give a DC restoration voltage midway between white and black levels for Cb and Cr, then YCbCr level is used to reduce the DC restoration voltage for Y back to the normal level. Therefore an increase in YCbCr level causes a decrease in the DC restoration voltage for the Y component. The actual voltage changes corresponding to increasing a level by four (i.e. the minimum change possible) are:

White Level	25.8mV
Black Level	19.9mV
Clamp Level	18.8mV
YCbCr Level	-9.06mV

All levels give 0V for a setting of zero.

It is important that the clamp level is such that the entire video waveform (including sync) is within the 0 .. 5V range otherwise the input protection circuitry will clip the waveform.

SEE ALSO

SNP24_get_levels, *SNP24_auto_gain*, *SNP24_auto_offset*.

SNP24_set_linescan_ctrl

USAGE

Terr SNP24_set_linescan_ctrl(Thandle Hsnp24, Tparam mode, ui16 width)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required linescan mode
width	Required width of line start out pulse

DESCRIPTION

This function controls parameters needed for line scan cameras. For many applications this function need not be called because *SNP24_initialize* calls it with the required parameters for the selected camera. Even if it is called it is only the line acceptance and trigger controls which should be needed.

The line start controls should only be needed to set up a camera which *SNP24_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.

MODE

SNP24_LSCAN_LINES_X1	Line acceptance ratio - all incoming lines are stored.
SNP24_LSCAN_LINES_X2	Line acceptance ratio - every other incoming line is stored.
SNP24_LSCAN_LINES_X4	Line acceptance ratio - 1 in 4 incoming lines are stored.
SNP24_LSCAN_LINES_X8	Line acceptance ratio - 1 in 8 incoming lines are stored.
SNP24_LSCAN_LTRIG_IN_POS	A positive edge on the line trigger input (trigger) will cause one line to be captured.
SNP24_LSCAN_LTRIG_IN_NEG	A negative edge on the line trigger input (trigger) will cause one line to be captured.
SNP24_LSCAN_LTRIG_IN_TIMER	One line will be captured on each positive edge from the baseboard timer. 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.
SNP24_LSCAN_LSTART_IN_POS	An active high line start in pulse from the camera is used to start Snapper-24 acquiring data. The line start in signal should be connected to the HSYNC pin.
SNP24_LSCAN_LSTART_IN_NEG	An active low line start in pulse from the camera is used to start Snapper-24 acquiring data. The line start in signal should be connected to the HSYNC pin.
SNP24_LSCAN_LSTART_IN_NONE	The camera does not generate a line start in pulse, so the internal line start out pulse is used used to start Snapper-24 acquiring data. Note that this internal pulse is not affected by the setting of *SNP24_LSCAN_LSTART_OUT_POS*, *SNP24_LSCAN_LSTART_OUT_NEG*, or *SNP24_LSCAN_LSTART_OUT_NONE*.
SNP24_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 *SNP24_LSCAN_LSTART_OUT_WIDTH*, and the pulse is generated is output on the VSYNC pin.
SNP24_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 *SNP24_LSCAN_LSTART_OUT_WIDTH*, and the pulse is generated is output on the VSYNC pin.
SNP24_LSCAN_LSTART_OUT_NONE	No line start out pulse will be generated.
SNP24_LSCAN_LSTART_OUT_WIDTH	The width of the line start out pulse is set by the *width* parameter. This value is in pixel clocks.

The *width* parameter is only used if the mode *SNP24_LSCAN_LSTART_OUT_WIDTH* is selected. If this mode is not selected then set *width* to zero.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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:

```
SNP24_set_linescan_ctrl(Hsnp24, SNP24_LSCAN_LINES_X1, 0);
```

To set up the Snapper-24 to capture one line every 100ms:

```
BASE_set_timer(Hbase, BASE_TIMER_ASTABLE, (ui32) 50000L);
SNP24_set_linescan_ctrl(Hsnp24, SNP24_LSCAN_LTRIG_IN_TIMER, 0);
```

To set up the Snapper-24 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:

```
SNP24_set_linescan_ctrl(Hsnp24, SNP24_LSCAN_LSTART_OUT_WIDTH, 40);
SNP24_set_linescan_ctrl(Hsnp24, SNP24_LSCAN_LSTART_OUT_POS |
       SNP24_LSCAN_LSTART_IN_NEG, 0);
```

BUGS / NOTES

IMPORTANT : Line scan mode is not supported in this release of software.

SEE ALSO

SNP24_set_linescan_freq, *SNP24_set_TTL422*.

SNP24_set_linescan_freq

USAGE

Terr SNP24_set_linescan_freq(Thandle Hsnp24, ui16 freq)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
freq	Required clock frequency in kHz.

DESCRIPTION

The *freq* parameter sets the frequency of the clock signal when the Snapper-24 is generating the clock. The value passed in is the required frequency in kilohertz. The phase locked loop on the board is used as a programmable frequency synthesiser in this mode.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_OUT_OF_RANGE	*freq* is less than 475 (i.e. 475 kHz) or more than 20000 (i.e. 20 MHz).

EXAMPLES

To set generate an RS-422 output clock with a frequency of 1 MHz:

```
/* First enable the PLL and drive out the clock in RS-422 mode */
SNP24_set_TTL422(Hsnp24, SNP24_TTL422_PCLK_422);
SNP24_set_clk(Hsnp24, SNP24_CLK_PLL, SNP24_PCLK_OUT_POS);
/* Now set the frequency to 1 MHz */
SNP24_set_linescan_freq(Hsnp24, 1000);
```

BUGS / NOTES

IMPORTANT : Line scan mode is not supported in this release of software.

SEE ALSO

SNP24_set_linescan_ctrl.

SNP24_set_LUTs

USAGE

Terr SNP24_set_LUTs(Thandle Hsnp24, Tparam mode, int sel252, ui8 lut[SNP24_SIZE_LUT252])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required mode.
sel252	Which channels to update.
lut	An array of 256 elements for the required LUT.

DESCRIPTION

This function controls the look up tables (LUTs) which the digitized video data is passed through between the output of the ADCs and the input to the frame store. For many applications this function need not be called directly, because *SNP24_initialize* calls it with *sel252* set to *SNP24_252_ALL* and *mode* set to *SNP24_LUT_INIT*.

SEL252 OPTIONS

This controls which of the three LUTs for the red, green and blue channels are affected by this call. The parameter can be one of *SNP24_252_RED, SNP24_252_GRN* or *SNP24_252_BLU* which only change one LUT; or *SNP24_252_ALL* which writes to all three LUTs in parallel for speed.

MODE

SNP24_LUT_INIT	This writes a linear ramp to the selected LUT(s), effectively bypassing it. That is LUT address 0 contains 0, LUT address 1 contains 1, etc, up to LUT address 255 contains 255. The contents of *lut[ ]* passed in are ignored.
SNP24_LUT_INVERSE	This writes a inverse linear ramp to the selected LUT(s). That is LUT address 0 contains 255, LUT address 1 contains 254, etc, up to LUT address 255 contains 0. The contents of *lut[ ]* passed in are ignored. This mode is useful for some line scan cameras which output inverse video.
SNP24_LUT_SET	This copies the values in *lut[ ]* to the selected LUT(s).

SNAPPER-8 DIFFERENCES

On a Snapper-8 *sel252* must be either *SNP24_252_ALL* or *SNP24_252_RED*, which are treated identically.

RETURNS

This function returns the LUT written in the *lut[ ]* array. Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.
ASLERR_PARAM_CONFLICT	More than one mode parameter has been passed in, which is not allowed because the parameters are mutually exclusive.
ASLERR_NOT_SUPPORTED	A Snapper-8 is being used, and *sel252* has been set to *SNP24_252_GRN* or *SNP24_252_BLU*.

EXAMPLES

The following code will set the red LUT to a ramp:

```
SNP24_set_LUT(Hsnp24, SNP24_LUT_INIT, SNP24_252_RED, lut);
```

The following code will set all three LUTs to binary threshold at level 100:

```
ui8 lut[SNP24_SIZE_LUT252];
ui8 *pLut;

pLut = lut;
for (lut_addr = 0; lut_addr < 100; lut_addr++)
   *pLut++ = 0;

for (lut_addr = 100; lut_addr < SNP24_SIZE_LUT252; lut_addr++)
   *pLut++ = 255;


SNP24_set_LUT(Hsnp24, SNP24_LUT_SET, SNP24_252_ALL, lut);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_get_LUTs

SNP24_set_parameter

USAGE

Terr SNP24_set_parameter(Thandle Hsnp24, ui16 parameter, ui32 value)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
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-24 handle.

PARAMETER

SNP24_TIMEOUT_TRIGGER	This sets the timeout value in milliseconds for the period from when *SNP24_capture* 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.

RETURNS

ASL_OK on success, otherwise *ASLERR_BAD_HANDLE* if the Bus Interface Board's handle is not valid.

EXAMPLES

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

```
SNP24_set_capture(Hsnp24, SNP24_TRIG_IN_ENABLE);
SNP24_set_parameter(Hsnp24, SNP24_TIMEOUT_TRIGGER, (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

SNP24_get_parameter.

SNP24_set_pix_per_line

USAGE

Terr SNP24_set_pix_per_line(Thandle Hsnp24, ui16 pix_per_line)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
pix_per_line	Required pixels per line

DESCRIPTION

The *pix_per_line* parameter overrides the default CCIR or EIA setting of the number of pixels per line for conventional areas scan cameras. This default is set by the *SNP24_CLK_INIT* parameter of *SNP24_set_clk*. Note that required setting is the number of pixel clocks between horizontal sync pulses, and is therefore larger than the number of pixel clocks in the active area; for example the default value for CCIR is 944. The maximum value allowable is 2048, and the minimum value 64. This function controls the phase locked loop (PLL) and is not relevant if the external clock PCLK is being used.

If it is intended to reduce the number of pixels per line it may be necessary to reduce the active area *ASL_ROI_X_LENGTH*, otherwise the ROI may extend beyond the end of the line.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_OUT_OF_RANGE	*pix_per_line* is less than 64 or more than 2048, or the value of *pix_per_line* is too small compared to the setting of the active area *ASL_ROI_X_LENGTH*.

EXAMPLES

To set 1024 pixels per line:

```
SNP24_set_pix_per_line(Hsnp24, 1024);
```

BUGS / NOTES

It may be necessary to adjust the *SNP24_set_clk* parameters *SNP24_PLL_VCO_GAIN* and *SNP24_PLL_PD_GAIN* if the number of pixels per line is changed significantly from the default value.

This function is not supported for line scan cameras - see *SNP24_set_linescan_freq*.

SEE ALSO

SNP24_set_clk.

SNP24_set_ROI

USAGE

Terr SNP24_set_ROI(Thandle Hsnp24, Tparam mode, i16 roi[ASL_SIZE_2D_ROI])

ARGUMENTS

Hsnp24	Handle to Snapper-24.
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 *SNP24_initialize* calls it to set the full ROI for CCIR or EIA cameras.

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* values 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 resolution is 4 pixels, and vertical is one line per field for *ASL_ROI_Y_START* and 2 lines per field for *ASL_ROI_Y_LENGTH*.

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 4 pixels. The coordinate *ASL_ROI_Y_START* should be 0, and *ASL_ROI_Y_LENGTH* defines the number of lines to capture per memory bank (see the Concepts section at the start of the manual for more information on memory banks). 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.

MODE

SNP24_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 *SNP24_ROI_SET* was called, without having to change the setup of the Snapper.
SNP24_ROI_SET	The *roi* passed in is selected.

PRE-PROCESSING

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 *SNP24_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 *SNP24_set_ROI_rounding*).

RETURNS

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

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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 500 pixels wide by 10 lines deep:

```
i16 roi[ASL_SIZE_2D_ROI];

roi[ASL_ROI_X_START] = 0;
roi[ASL_ROI_Y_START] = 0;
roi[ASL_ROI_X_LENGTH] = 500;
roi[ASL_ROI_Y_LENGTH] = 10;
SNP24_set_ROI(Hsnp24, SNP24_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] = 768;
roi[ASL_ROI_Y_LENGTH] = 576;
SNP24_set_active_area(Hsnp24, 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;
SNP24_set_ROI(Hsnp24, SNP24_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 768,
 *                       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.

SEE ALSO

SNP24_get_ROI, *SNP24_get_ROI_max*, *SNP24_set_active_area*, *SNP24_set_linescan_ctrl*, *SNP24_set_ROI_rounding*.

SNP24_set_ROI_rounding

USAGE

Terr SNP24_set_ROI_rounding(Thandle Hsnp24, ui16 x_round, ui16 y_round)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
x_round	Required x direction rounding value
y_round	Required y direction rounding value

DESCRIPTION

x_round and *y_round* allow *SNP24_set_ROI* to automatically adjust ROIs. For many applications this function need not be called directly, because *SNP24_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-24 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 *SNP24_set_ROI* will be a multiple of 2 pixels wide, and a multiple of 8 lines high:

```
SNP24_set_ROI_rounding(Hsnp24, 2, 8);
```

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

```
SNP24_set_capture(Hsnp24, SNP24_SUB_X1);
SNP24_set_ROI_rounding(Hsnp24, 2, 1);
roi[ASL_ROI_X_LENGTH] = 84;
SNP24_set_roi(Hsnp24, SNP24_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:

```
SNP24_set_capture(Hsnp24, SNP24_SUB_X4);
SNP24_set_ROI_rounding(Hsnp24, 2, 1);
roi[ASL_ROI_X_LENGTH] = 86;
SNP24_set_roi(Hsnp24, SNP24_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 *SNP24_set_ROI_rounding* will not take affect until *SNP24_set_ROI* has been called.

SEE ALSO

SNP24_set_ROI, *SNP24_get_ROI_max*.

SNP24_set_sync

USAGE

Terr SNP24_set_sync(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required sync mode.

DESCRIPTION

This function controls Snapper-24's sync circuits for use with conventional area scan cameras. 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 *SNP24_initialize* calls it with the *SNP24_SYNC_INIT* parameter.

MODE

SNP24_SYNC_INIT	The first call made to *SNP24_set_sync* must include this parameter. This selects sync off video, sets the HSYNC and VSYNC pins to be inputs and sets the sync generator to CCIR or EIA operation based on the most recent call to *SNP24_set_video_standard*. It also sets the mode *SNP24_SYNC_FIELDS_STD*.
SNP24_SYNC_OFF_VIDEO	The sync source is selected based on the most recent calls to *SNP24_set_format* and *SNP24_set_video_src*. Subsequent calls to *SNP24_set_format* and *SNP24_set_video_source* will cause the sync source to be updated automatically. This automatic mode is cancelled by calling *SNP24_set_sync* with a different *SNP24_SYNC_OFF_* parameter, or *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_OFF_RED1, SNP24_SYNC_OFF_RED2, SNP24_SYNC_OFF_RED3, SNP24_SYNC_OFF_RED4	One of the four red video inputs is selected as the sync source. The selected input must have a negative 75 Ohm composite sync signal connected, or a video signal including sync.
SNP24_SYNC_OFF_GRN1, SNP24_SYNC_OFF_GRN2, SNP24_SYNC_OFF_GRN3, SNP24_SYNC_OFF_GRN4	One of the four green video inputs is selected as the sync source. The selected input must have a negative 75 Ohm composite sync signal connected, or a video signal including sync.
SNP24_SYNC_OFF_BLU1, SNP24_SYNC_OFF_BLU2, SNP24_SYNC_OFF_BLU3, SNP24_SYNC_OFF_BLU4	One of the four blue video inputs is selected as the sync source. The selected input must have a negative 75 Ohm composite sync signal connected, or a video signal including sync.
SNP24_SYNC_OFF_CSYNC_NEG	The CSYNC pin is selected as the sync source. This input must have a negative 75 Ohm composite sync signal connected, or a video signal including sync.
SNP24_SYNC_OFF_HVSYNC_POS	The HSYNC and VSYNC pins are selected as the sync source. These inputs must have positive TTL or RS-422 horizontal sync and vertical sync signals connected respectively.

SNP24_SYNC_OFF_HSYNC_POS_VSYNC_NEG	The HSYNC and VSYNC pins are selected as the sync source. These inputs must have a positive horizontal sync and a negative vertical sync, TTL or RS-422 signal connected respectively.
SNP24_SYNC_OFF_HVSYNC_NEG	The HSYNC and VSYNC pins are selected as the sync source. These inputs must have negative TTL or RS-422 horizontal sync and vertical sync signals connected respectively.
SNP24_SYNC_OFF_HSYNC_NEG_VSYNC_POS	The HSYNC and VSYNC pins are selected as the sync source. These inputs must have a negative horizontal sync and a positive vertical sync, TTL or RS-422 signal connected respectively.
SNP24_SYNC_INTERNAL	The internal sync generator is selected as the sync source.
SNP24_SYNC_CSYNC_OUT_NEG	This drives the CSYNC pin with a negative 75 Ohm black level composite sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_HSYNC_OUT_POS	This drives the HSYNC pin with a positive TTL or RS-422 horizontal sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_HSYNC_OUT_NEG	This drives the HSYNC pin with a negative TTL or RS-422 horizontal sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_VSYNC_OUT_POS	This drives the VSYNC pin with a positive TTL or RS-422 vertical sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_VSYNC_OUT_NEG	This drives the VSYNC pin with a negative TTL or RS-422 vertical sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_C_ON_H_OUT_POS	This drives the HSYNC pin with a positive TTL or RS-422 composite sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_C_ON_H_OUT_NEG	This drives the HSYNC pin with a negative TTL or RS-422 composite sync signal derived from the selected sync source. This is most useful in conjunction with *SNP24_SYNC_INTERNAL*.
SNP24_SYNC_FIELDS_SWAP	This causes the board to reverse the order in which fields are stored as they are written to video memory, i.e. the first incoming field is stored as the second field and vice-versa. This is intended to allow use with video sources which output their fields reversed; it also can be used to turn *SNP24_capture* mode *SNP24_START_1ST_FIELD* into next 2nd field.

SNP24_SYNC_FIELDS_STD	This is the default mode, and should be used to restore Snapper-24 to standard mode after *SNP24_SYNC_FIELDS_SWAP* has been called.

PARAMETER INTERACTION

When combinations of parameters are passed in one call to *SNP24_set_sync* 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.

SNAPPER-8 DIFFERENCES

On a Snapper-8 the modes *SNP24_SYNC_OFF_GRN* and *SNP24_SYNC_OFF_BLU* are not supported.

IMPORTANT: When writing code for a Snapper-8 which must also run on a Snapper-24, note that the default sync source selected by *SNP24_SYNC_OFF_VIDEO* will be different because the format set by *SNP24_set_format* is different. Therefore the application must call *SNP24_set_format* with *SNP24_FORMAT_Y8_ON_RED* selected after calling *SNP24_initialize*.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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. See the examples given below. No change is made to the setting of the sync circuits.
ASLERR_NOT_RECOGNIZED	This occurs when *SNP24_SYNC_INIT* is passed and the video standard setting is not recognised, possibly because *SNP24_set_video_standard* has not been called.
ASLERR_NOT_SUPPORTED	A Snapper-8 is being used, and one of the *SNP24_SYNC_OFF_GRN* or *SNP24_SYNC_OFF_BLU* modes has been requested.

EXAMPLES

In the following call the *SNP24_SYNC_OFF_RED2* option will override the default of *SNP24_SYNC_OFF_VIDEO* set by *SNP24_SYNC_INIT*:

```
SNP24_set_sync(Hsnp24, SNP24_SYNC_INIT | SNP24_SYNC_OFF_RED2);
```

In the following call the internal sync generator will be selected, and the HSYNC pin will be driven out, again overriding the default settings:

```
SNP24_set_sync(Hsnp24, SNP24_SYNC_INIT | SNP24_SYNC_INTERNAL |
       SNP24_SYNC_HSYNC_OUT_NEG);
```

The following call will result in a *ASLERR_PARAM_CONFLICT* error because the external CSYNC pin cannot function as both the sync source and as an output at the same time:

```
SNP24_set_sync(Hsnp24, SNP24_SYNC_OFF_CSYNC_NEG | SNP24_SYNC_CSYNC_OUT_NEG);
```

BUGS / NOTES

When the external VSYNC and HSYNC signals are selected as the sync source, the CSYNC output is generated by gating HSYNC and VSYNC, therefore the waveform generated is not ideal. If a 'proper'

CSYNC output is needed, either use the internal sync generator, or feed a good composite sync source into one of the video inputs and select it as the sync source.

The function is not supported in line scan mode - see *SNP24_set_linescan_ctrl*.

SEE ALSO

SNP24_set_video_standard, *SNP24_set_clk*, *SNP24_set_TTL422*.

SNP24_set_timer

USAGE

*Terr SNP24_set_timer(Thandle Hsnp24, Tparam mode, ui32 time, Terr (EXPORT_FN *time_fn)(Thandle))*

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required timer mode.
time	Required time pulse width.
time_fn	Required timer control function.

DESCRIPTION

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 *SNP24_initialize* calls it with the parameter *SNP24_TIMER_INIT*.

SNP24_set_timer sets up a timer control function which is automatically called by *SNP24_capture* 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 *SNP24_capture* will automatically call it.

The timer pulse can only be output on the trigger pin, so *SNP24_set_trigger* must be called to set the trigger pin in the required mode.

The timer hardware is fitted to the baseboard. Some baseboards (such as ISA-JPG) do not provide this function, resulting in the first call to *SNP24_capture* 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

SNP24_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.
SNP24_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.
SNP24_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.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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:

```
SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_TIMER_NEG);
SNP24_set_timer(Hsnp24, SNP24_TIMER_SET_EXPOSURE, (ui32) 10000, NULL);
...
SNP24_capture(Hsnp24, SNP24_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:

```
main()
{
   ...
   SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_TIMER_POS);
   SNP24_set_timer(Hsnp24, SNP24_TIMER_SET_TIME_FN, 0, my_timer_function);
   ...
}

Terr EXPORT_FN my_timer_function(Thandle Hsnp24)
{
   Thandle Hbase;

   /* First get baseboard handle so BASE functions can be called */
   Hbase = SNP24_get_parameter(Hsnp24, SNP24_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 */
   SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_LO);
   BASE_set_timer(Hbase, BASE_TIMER_MONOSTABLE | BASE_TIMER_START_AND_WAIT,
       (ui32) 1000000L);
   /* Finally generate second pulse */
   SNP24_set_trigger(Hsnp24, SNP24_TRIG_OUT_TIMER_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.

SEE ALSO

BASE_set_timer, *SNP24_set_trigger*.

SNP24_set_trigger

USAGE

Terr SNP24_set_trigger(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required trigger mode.

DESCRIPTION

This function sets the trigger mode. For many applications this function need not be called directly, because *SNP24_initialize* calls it with the parameter *SNP24_TRIG_INIT*. Note that *SNP24_TRIG_IN_POS* and *SNP24_TRIG_IN_NEG* do not enable triggering - to do this call *SNP24_set_capture* with *SNP24_TRIG_IN_ENABLE*.

Up to four trigger sources can be used; the 'standard' trigger pin is always available, and in addition the VSYNC, HSYNC and pixel clock pins can be used as trigger inputs if they are not being used for normal sync or clock use.

The mode parameter should be one of the following:

MODE

SNP24_TRIG_INIT	This sets the standard trigger pin as an input with the rising edge active, and selects the standard trigger pin as the active trigger source.
SNP24_TRIG_IN_POS	This specifies that the standard trigger pin is an input with the rising edge active.
SNP24_TRIG_IN_NEG	This specifies that the standard trigger pin is an input with the falling edge active.
SNP24_TRIG_OUT_HI	This drives the standard trigger pin to a high level.
SNP24_TRIG_OUT_LO	This drives the standard trigger pin to a low level.
SNP24_TRIG_OUT_TIMER_POS	This drives the standard trigger from the baseboard's timer, with an active high pulse when the timer is in monostable mode.
SNP24_TRIG_OUT_TIMER_NEG	This drives the standard trigger from the baseboard's timer, with an active low pulse when the timer is in monostable mode.
SNP24_TRIG_IN_TRIG1	This selects the standard trigger pin as the active trigger source.
SNP24_TRIG_IN_TRIG2	This selects the VSYNC pin as a trigger, and makes it the active trigger source. The VSYNC pin must first be enabled as an input of the required polarity via a call to *SNP24_set_sync*.
SNP24_TRIG_IN_TRIG3	This selects the pixel clock pin as a trigger, and makes it the active trigger source. The pixel clock pin must first be enabled as an input of the required polarity via a call to *SNP24_set_clk*.
SNP24_TRIG_IN_TRIG4	This selects the HSYNC pin as a trigger, and makes it the active trigger source. The HSYNC pin must first be enabled as an input of the required polarity via a call to *SNP24_set_sync*.
SNP24_TRIG_IN_VIDSRC	The trigger source is selected based on the most recent call to *SNP24_set_video_src*. Subsequent calls to *SNP24_set_video_src* will cause the trigger source to be updated automatically. This automatic

mode is cancelled by calling *SNP24_set_trigger* with a different *SNP24_TRIG_IN_* parameter. The dual use sync/clk/trigger pins must first have been enabled as inputs of the required polarity - see above.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 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 select falling edge trigger on the default trigger input, then acquire the image on a trigger input pulse:

```
SNP24_set_trigger(Hsnp24, SNP24_TRIG_IN_NEG | SNP24_TRIG_IN_TRIG1);
SNP24_set_capture(Hsnp24, SNP24_TRIG_IN_ENABLE);
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
```

To select falling edge trigger on the dual use pixel clock input, then acquire the image on a trigger input pulse. Note the use of two calls to *SNP24_set_clk*, the first to set the pixel clock pin as a falling edge input, the second to restore the required clock mode:

```
SNP24_set_clk(Hsnp24, SNP24_CLK_PCLK_IN_NEG);
SNP24_set_clk(Hsnp24, SNP24_CLK_PLL);
SNP24_set_trigger(Hsnp24, SNP24_TRIG_IN_TRIG3);
SNP24_set_capture(Hsnp24, SNP24_TRIG_IN_ENABLE);
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT);
```

To use three trigger sources in *SNP24_TRIG_IN_VIDSRC* mode, and the pixel clock pin as a normal clock output. With pixel clock in use the three trigger sources free are the standard trigger, VSYNC and HSYNC; therefore the video sources used must be VIDSRC1, VIDSRC2 and VIDSRC4. VIDSRC3 cannot be used because the use of *SNP24_TRIG_IN_VIDSRC* mode would make the pixel clock pin a trigger input: Note the use of two calls to *SNP24_set_sync*, the first to set the VSYNC pin as a falling edge input, the second to restore the required sync mode:

```
SNP24_set_sync(Hsnp24, SNP24_SYNC_OFF_HVSYNC_POS);
SNP24_set_sync(Hsnp24, SNP24_SYNC_OFF_VIDEO);
SNP24_set_trigger(Hsnp24, SNP24_TRIG_IN_POS | SNP24_TRIG_IN_VIDSRC);
SNP24_set_capture(Hsnp24, SNP24_TRIG_IN_ENABLE);
SNP24_set_video_src(Hsnp24, SNP24_VIDSRC_SRC2, SNP24_SEL252_ALL);
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT); /* Active trigger: VSYNC */
...
SNP24_set_video_src(Hsnp24, SNP24_VIDSRC_SRC1, SNP24_SEL252_ALL);
SNP24_capture(Hsnp24, SNP24_START_AND_WAIT); /* Active trigger: trigger pin */
```

BUGS / NOTES

There are no known bugs. Note that if more than one programmable output is required the VSYNC and pixel clock pins can also be used via a call to *SNP24_set_ctrlout*.

SEE ALSO

SNP24_is_capture_complete, *SNP24_is_trigger_started*, *SNP24_set_capture*, *SNP24_set_TTL422*, *SNP24_set_sync*, *SNP24_set_clk*, *SNP24_set_timer*.

SNP24_set_TTL422

USAGE

Terr SNP24_set_TTL422(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required TTL or RS-422 mode.

DESCRIPTION

This function sets the software control of TTL or RS-422 input and output formats for HSYNC, VSYNC, PCLK and trigger. For many applications this function need not be called directly, because *SNP24_initialize* calls it with the parameter *SNP24_TTL422_INIT*. Even if it is called, it need only be called once, because it does not directly control TTL / RS-422 formats, instead it informs the software of the jumper settings on Snapper-24. See the ***Snapper!*** *Installation Guide* for details of setting these jumpers.

The mode parameter should be formed by 'OR'ing the required options from the list below:

MODE

SNP24_TTL422_INIT	This sets the formats to match the default jumper settings on Snapper-24 as shipped, i.e. HSYNC, VSYNC and trigger as TTL; and PCLK as RS-422.
SNP24_TTL422_HSYNC_422	This specifies that the HSYNC format is RS-422.
SNP24_TTL422_HSYNC_TTL	This specifies that the HSYNC format is TTL.
SNP24_TTL422_VSYNC_422	This specifies that the VSYNC format is RS-422.
SNP24_TTL422_VSYNC_TTL	This specifies that the VSYNC format is TTL.
SNP24_TTL422_TRIG_422	This specifies that the trigger format is RS-422.
SNP24_TTL422_TRIG_TTL	This specifies that the trigger format is TTL.
SNP24_TTL422_PCLK_422	This specifies that the PCLK format is RS-422.
SNP24_TTL422_PCLK_TTL	This specifies that the PCLK format is TTL.

PARAMETER INTERACTION

When combinations of parameters are passed in one call to *SNP24_set_TTL422* 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_HANDLE	The Snapper-24 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

Select HSYNC as RS-422 format:

```
SNP24_set_TTL422(Hsnp24, SNP24_TTL422_HSYNC_422);
```

Select PCLK as TTL format, overriding the default set by SNP24_TTL422_INIT:

```
SNP24_set_TTL422(Hsnp24, SNP24_TTL422_INIT | SNP24_TTL422_PCLK_TTL);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

SNP24_set_clk, *SNP24_set_sync*, *SNP24_set_trigger*.

SNP24_set_video_src

USAGE

Terr SNP24_set_video_src(Thandle Hsnp24, Tparam mode, int sel252)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required mode.
sel252	Which channels to update.

DESCRIPTION

This function controls the video source to the ADCs. For many applications this function need not be called directly, because *SNP24_initialize* calls it with *sel252* set to *SNP24_252_ALL* and *mode* set to *SNP24_VIDSRC_INIT*.

SEL252 OPTIONS

This controls which of the three ADCs for the red, green and blue channels are affected by this call. The parameter can be one of *SNP24_252_RED, SNP24_252_GRN* or *SNP24_252_BLU* which only changes the source for one ADC; or *SNP24_252_ALL* which changes the source to all three ADCs in parallel for speed. *SNP24_252_ALL* would normally be used for RGB or YCrCb video sources.

MODE

SNP24_VIDSRC_INIT	This selects the first of the four possible inputs as the video source for all the ADCs. *sel252* must be set to *SNP24_252_ALL*.
SNP24_VIDSRC_SRC1	This selects the first of the four possible inputs as the video source for the selected ADC(s).
SNP24_VIDSRC_SRC2	This selects the second of the four possible inputs as the video source for the selected ADC(s).
SNP24_VIDSRC_SRC3	This selects the third of the four possible inputs as the video source for the selected ADC(s).
SNP24_VIDSRC_SRC4	This selects the fourth of the four possible inputs as the video source for the selected ADC(s).

If sync off video is selected (see *SNP24_set_sync*), and *sel252* is set to *SNP24_252_ALL,* the sync source will be switched to the new video source. Similarly, if the trigger source or the control outputs are set to change with video source (see *SNP24_set_trigger* and *SNP24_set_ctrlout*) they will change automatically.

SNAPPER-8 DIFFERENCES

On a Snapper-8 *sel252* must be either *SNP24_252_ALL* or *SNP24_252_RED*, which are treated identically.

RETURNS

Possible error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.
ASLERR_BAD_PARAM	The mode parameter is invalid.

ASLERR_PARAM_CONFLICT	More than one mode parameter has been passed in, which is not allowed because the parameters are mutually exclusive, or *SNP24_VIDSRC_INIT* has been passed without *sel252* being set to *SNP24_252_ALL*.
ASLERR_NOT_SUPPORTED	A Snapper-8 is being used, and *sel252* has been set to *SNP24_252_GRN* or *SNP24_252_BLU*.

EXAMPLES

The following code will set all ADCs to digitize input channel 2:

```
SNP24_set_vidsrc(Hsnp24, SNP24_VIDSRC_SRC2, SNP24_252_ALL);
```

BUGS / NOTES

There are no known bugs.

SEE ALSO

-

SNP24_set_video_standard

USAGE

Terr SNP24_set_video_standard(Thandle Hsnp24, Tparam mode)

ARGUMENTS

Hsnp24	Handle to Snapper-24.
mode	Required video standard.

DESCRIPTION

This function is used to inform the Snapper-24 software of the video standard in use, so that subsequent calls to functions such as *SNP24_set_sync* work correctly.

MODE

SNP24_CCIR_DEFAULT	Set the video standard to CCIR. This is used in the UK and most of Western Europe, and has a 50Hz frame rate.
SNP24_EIA_DEFAULT	Set the video standard to EIA. This is used in the USA and Japan, and has a 60Hz frame rate. This mode is automatically selected when using line scan mode.

RETURNS

This function returns the following error codes:

ASL_OK	If successful.
ASLERR_BAD_HANDLE	The Snapper-24 handle is invalid.

EXAMPLES

Select EIA operation:

```
SNP24_set_video_standard(Hsnp24, SNP24_EIA_DEFAULT);
```

BUGS / NOTES

It is necessary to call *SNP24_set_clk* and *SNP24_set_sync* with the initialize parameter to reset the PLL and sync generators to the new video standard - *SNP24_set_video_standard* does not make these calls.

SEE ALSO

SNP24_is_50Hz, *SNP24_set_clk*, *SNP24_set_sync*.