Advanced Micro Peripherals Ltd.

MPEG4-CPCI and MPEG4000-XLP

Video Recording SDK

Software Reference Manual

Document Version 1.22

30 October 2008

Revision History

Document Release	***Date***	***Description***	***Approved***
1.00	06/12/06	Initial release	
1.01	07/03/07	Added OSD, analogue output and decode functions	
1.02	20/03/07	Added Linux instructions	
1.03	23/03/07	Added MPG4KX_SetDigitalPath function Added ID_PIP_MODE encoding Added MPG4K_SetPIPPosition Added hardware block diagram	
1.04	11/04/07	Added MPG4K_SetMotionThreshold, MPG4K_EnableMotion, MPG4K_SetPreviewMode, MPG4K_RegisterPrivate, MPG4K_WritePrivateData, MPG4K_SetPreviewInputPath, MPG4KX_GetDecodeSource	
1.05	20/04/07	Added MPG4KX_SetMotioPreProcessing, MPG4KX_SetMotionMask, MPG4KX_AddMotionMask, MPG4KX_SetPIPSize, MPG4KX_SetAlwaysOpenFile	
1.06	20/06/07	Corrected minimum recommended bit rates, changed return value of MPG4KX_StartDecode, added MPG4KX_SetFrameSkip, MPG4KX_SetEncoderLocation, MPG4KX_SetInputScale, MPG4K_SetAudioPath, MPG4KX_VideoDetected, MPG4KX_ForceFont, MPG4KX_DecodeCommandEx, MPG4KX_GetDecodeSourceID	
1.07	31/07/07	Corrected MPG4KX_SetFrameRate comments Added MPG4KX_SetNumChannels	
1.08	02/08/07	Added MP4 file format Added MPG4KX_EnableFilters and MPG4KX_SetFilterLevel, MPG4KX_EnableQTCompatibility, MPG4K_SetThreadAttr	
1.09	30/08/07	Minor typo fixes Added MPG4KX_GetPreviewHeight, MPG4KX_GetPreviewWidth and MPG4KX_SingleShotTrigger	
1.10	26/10/07	Corrected MPG4K_AddMotionCallback, MPG4K_AddVideoCallback and MPG4K_AddAudioCallback. Added MPG4KX_QueueDecode, MPG4KX_AVIFileOpen and MPG4KX_SetAVICloseFrameCount Updated MPG4KX_StartDecode	
1.11	29/10/07	Added filename selection of file type to MPG4KX_SingleShotTrigger	
1.12	05/11/07	Added MPG4KX_SetChannelScale and MPG4KX_SetMuxInputSequence	

1.13	11/12/07	Added MPG4KX_ShowPlayback and MPG4KX_SetFrameMode	
1.14	24/01/08	Added ID_DUAL_MODE_SINGLE to MPG4K_SetMode and MPG4K_SetPreviewMode. Added MPG4K_Rectangle, MPG4K_SetRectangleColour, MPG4KX_SetOSDCustomColour, MPG4KX_SetOSDTextColour	
1.15	05/02/08	Added MPG4KX_SetNumPreviewBuffers and MPG4KX_SetPreviewFrameRate	
1.16	10/06/08	Fixed MPG4K_EnableChannel prototype Fixed MPG4KX_VideoDetected description Added MPG4K_RemovePreviewCallback Added ID_OSD_ODDFIELD, ID_OSD_EVENFIELD flags to MPG4KX_PrintOSD	
1.17	20/06/08	Added QNX support	
1.18	28/07/08	Clarified return value from MPG4K_PostTriggering Corrected return values for MPG4X_GetDecodeDuration and MPG4KX_GetDecodeLocation Added return value to MPG4K_SetPreviewLocation Fixed some index entries Changed return value from MPG4KX_SingleShotTrigger Added MPG4KX_StopSingleShotTrigger Added MPG4K_GetFIFOLevel	
1.19		Fixed MPG4K_AddPreviewCallback *Callback parameter Fixed preview FOURCC RGB2	
1.20	15/10/08	Added MPG4KX_EnableExtraFrameInfo Clarified MPG4K_Stop and MPG4K_StopEncoder Clarified the functions that cannot be called while encoding is happening	
1.21	24/10/08	Corrected return values for MPG4K_SetQuality	
1.22	30/10/08	Clarified OSD character grid sizes	

AMP does not recommend the use of any of its products in life support applications wherein a failure or malfunction of the product may directly threaten life or injury. The user of AMP products in life support applications assumes all risk of such use and indemnifies AMP against all damages. All information contained in this document is proprietary to AMP and may not be reproduced or disclosed to any third parties without the written consent of AMP

The information contained in this document has been carefully checked and is believed to be reliable. However, Advanced Micro Peripherals Ltd (AMP) makes no guarantee or warranty concerning the accuracy of said information and shall not be responsible for any loss or damage of whatever nature resulting from the use of, or reliance upon, it. AMP does not guarantee that the use of any information herein will not infringe upon the patent, trademark, copyright, or other rights of third parties, and no patent or other license is implied hereby.

AMP reserves the right to make changes in the products or specifications, or both, presented in this document at any time and without notice.

Table of Contents

Introduction

This is the API reference for the Advanced Micro Peripherals Ltd MPEG4-CPCI and MPEG4000-XLP MPEG4 hardware encoder card.
The API is the same for all supported operating systems. Functions that are specific to an OS are marked as such.

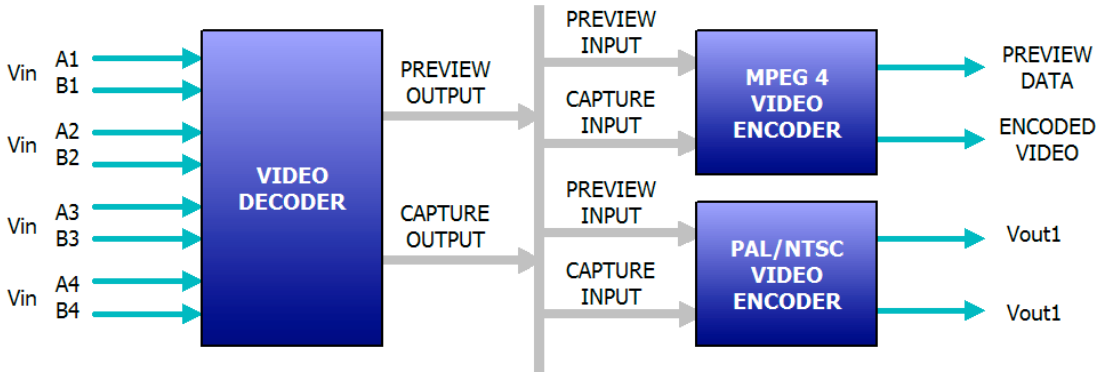
Related Documentation

Other Documentation that may be of use whilst reading this document are described in the table below:

Document Name	**Source**
MPEG4-CPCI Hardware Reference Manual	MPEG4-CPCI SDK
MPEG4000-XLP Hardware Reference Manual	MPEG4000-XLP SDK

Hardware block diagram

The video decoder has two digital video output paths, one is the preview output and the other the capture output.
The MPEG4 video encoder has two digital video input paths, one is the preview input and the other the capture input.
It is possible to route either of the video decoder outputs to either of the MPEG4 video encoder inputs.
The data routed to the capture input to the MPEG4 video encoder is encoded by the MPEG4 encoding engine and output as encoded data.
The data routed to the preview input to the MPEG4 video encoder is passed through as raw video data and sent to the preview call back chain, where it optionally displayed on screen.



Overview of writing Applications

The MPEG4-CPCI and MPEG4000-XLP SDKs provides functions to control every aspect of the MPEG4-CPCI and MPEG4000-XLP respectively.

The MPG5K.H header file must be included in all applications using the MPEG4-CPCI or MPEG4000-XLP API.

The SDK library should be initialised for each card by calling the **MPG4K_InitCard** (page **14**) function. The return value should be checked to confirm that the operation succeeded. If the return is not ID_OK then the application should not continue. This must be done before any other functions from the SDK are called.

Windows Library files

The application should be linked against the MPG5K.DLL file. This file is found in *software/lib*
The MGP5K.DLL should be placed in the System directory on the target machine. Under Windows NT 4 and Windows 2000 this is */winnt/system32*. Under Windows XP this is */windows/system32*
For preview under Windows, DirectX support is required.
To use the software on systems without DirectX, link against the MPG5K-np.LIB and use the MGP5K-np.DLL

Linux Library files

The application should be linked against the libmpeg5k file. This file is found in software/lib, named libmpeg5k-glibcXX.so for SDL based preview, libmpeg5k-glibcXXxv.so for X11/XVideo based preview and libmpeg5k-glibcXXnp.so for no onscreen preview support, where XX is the target glibc version.
The libmpeg5k file should be placed in an appropriate library directory on the target machine. In general, either /usr/local/lib or /usr/lib should be used. Having copied the file, run ldconfig after verifying that the library path is present in the /etc/ld.so.conf file.

QNX library files

The application should be linked against the libmpeg5k file. This file is found in software/lib, named libmpeg5k.so for Photon Graphics based preview and libmpeg5k-np.so for no onscreen preview support.
The libmpeg5k file should be placed in the /usr/lib directory. The install.sh shell script will do this for you.

Firmware file

The firmware files is called *mpeg5k.bin* and can be found in the *software/lib* directory.
It should be copied to the working directory of the target application. This is normally the directory the application is run from.

Preview

The MPEG4-CPCI and MPEG4000-XLP both have a preview feature where the video being encoded can be viewed on screen.
Under Windows the on screen preview feature uses DirectX surfaces.
Under Linux the on screen preview feature uses SDL or X11/XVideo, depending the library linked against.

Under QNX the on screen preview feature uses the Photon Graphics library. An overlay scaler is a requirement for the on screen preview to function properly.

A callback function can also be registered with the SDK library that will be called with the raw video data for each preview frame.

Where possible colour keying will be used for the preview. This means that the preview video will only display in areas that are the colour key. The default colour key is magenta (Red=255, Blue=0, Green=255).

Decoding

The decoding feature of the MPEG4000XLP uses external plugin libraries to parse the various container formats into a form the decoder understands.
Under Windows these DLLs should be placed on the System directory on the target machine.
Under Linux, the .so files should be placed in /usr/local/lib
Under QNX decoding is currently not supported.

On the MPEG4-CPCI only card 1 can be used for decode. The output of the decode goes to card 0. To view the decode on either the analogue output or on the VGA preview, the playback source must be enabled using the **MPG4KX_ShowPlayback** (page **52**) function. The MPG4XDecodePreview example application is given in the MPEG4-CPCI SDK to demonstrate this.

Structure Reference

```
typedef struct tagErrorCallback
{
	int nCard;
	int nChannel;
	int nSource;
	int nError;
} tErrorCallback;
```

This structure is used to pass information to the error callback function.
The nCard and nChannel elements specify which card and channel have generated the error.
The nSource element specifies the source of the error.
The nError element specifies the error number for the error.

```
typedef struct tagDecodeCommand
{
	int nSize;
	unsigned long ulCommand;
	unsigned long ulFlags;
	union
	{
		unsigned char bParam;
		unsigned short usParam;
		int nParam;
		unsigned long ulParam;
		long lParam;
#ifdef WIN32
		__int64 i64Param;
#else
		long long i64Param;
#endif
	}Param;
} tDecodeCommand;
```

This structure is used to pass commands to the decode engine.
The nSize element specifies the size of the tDecodeCommand instance and must be filled in.
The ulCommand element specifies the command.
The ulFlags element specifies flags specific to the command. It also contains flags to specify which of the Param variables is to be used.
The Param element contains an option parameter value for the command. The ulFlags element must contain the correct flag specifying which value to use.

```
#define ID_DECODE_BYTE 0x10000
Use the bParam entry in the Param element.

#define ID_DECODE_USHORT 0x20000
Use the usParam entry in the Param element.

#define ID_DECODE_ULONG 0x30000
Use the ulParam entry in the Param element.

#define ID_DECODE_INT 0x40000
Use the nParam entry in the Param element.

#define ID_DECODE_LONG 0x50000
Use the lParam entry in the Param element.
```

```
#define ID_DECODE_LONGLONG 0x60000
Use the i64Param entry in the Param element.
```

Function Reference

Initialisation functions

MPG4K_InitCard

int MPG4K_InitCard (nCard)

int *nCard*; /* */

The **MPG4K_InitCard** function initialises the MPEG4000 SDK and hardware.

Parameter	*Description*
nCard	Specifies which card to initialise

Returns

The return value is one of the following values:

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	The MPEG4000 could not be opened or was not detected.
ID_ERR_NOFWFILE	The firmware file was not found.
ID_ERR_MEMALLOC	Memory allocation for the firmware failed
ID_ERR_FWUPLOAD	An error occurred uploading the firmware
ID_ERR_INVALID_CHANNEL	Invalid card number specified

Comments

The application should only continue if the return was ID_OK. All other errors are fatal.

MPG4K_DeInitCard

int MPG4K_DeInitCard (*nCard)*

int *nCard*; /* */

The **MPG4K_DeInitCard** function de-initialises the specified card.

Parameter	*Description*
nCard	Specifies the card to de-initialise

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	The MPEG4000 could not be opened or was not detected.

Comments

MPG4K_SetParserThreadAttr

int MPG4K_SetParserThreadAttr (nCard, *attr)

int *nCard*; /* */
pthread_attr **attr*; /* */

The **MPG4K_SetParserThreadAttr** function sets the desired parser thread priority attributes under POSIX operating systems.

Parameter	*Description*
nCard	Specifies which card to set the parser thread priority
**attr*	Pointer to a pthread_attr structure that contains the desired thread attributes

Returns

The return value is one of the following values:

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	The MPEG4000 could not be opened or was not detected.

Comments

This function is only available under POSIX based operating systems such as Linux and QNX.

This function should be called before the encoding is started with

Encoding control functions

MPG4K_Start

int MPG4K_Start (*nCard)*

int *nCard*; /* */

The **MPG4K_Start** function starts the encoding.

Parameter	*Description*
nCard	Specifies which MPEG4000 card to start the encoding on.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_ENCODING	The encoder is already running
ID_ERR_SAVE_FAILE	Failed to start saving data

Comments

MPG4K_Stop

int MPG4K_Stop (*nCard)*

int *nCard*; /* */

The **MPG4K_Stop** function stops the encoding for the specified card.

Parameter	*Description*
nCard	Specifies which MPEG4000 card to stop

Returns

The return value is

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_ENCODING	The encoder is not running

Comments

This function is equivalent to calling **MPG4K_EnableChannel** (page **26**) to disable all channels on the specified card.

MPG4K_StopEncoder

int MPG4K_Stop (*nCard)*

int *nCard*; /* */

The **MPG4K_StopEncoder** function stops the encoding for the specified card.

Parameter	*Description*
nCard	Specifies which MPEG4000 card to stop

Returns

The return value is

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_ENCODING	The encoder is not running

Comments

This function stops the encoding process. No more data will be received until **MPG4K_Start** (page **17**.) is called again.

MPG4K_SetInputStandard

int MPG4K_SetInputStandard (*nCard*, *nStandard)*

int *nCard*; /* */
int *nStandard*; /* */

The **MPG4K_SetInputStandard** function sets the standard for the input video.

Parameter	*Description*
nCard	Specifies the card to set the standard for
nStandard	Specifies the video standard for the input. This can be one of the following values.

Value	*Meaning*
ID_PAL	PAL
ID_NTSC	NTSC

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_STANDARD	An invalid video standard was passed
ID_ERR_ENCODING	The encoder is running

Comments

The input video standard can only be changed when the encoder is in a stopped start.

This function should not be called after **MPG4K_SetFrameRate** (page **32**) or **MPG4KX_SetFrameRate** (page) **33**

MPG4K_GetInputStandard

int MPG4K_GetInputStandard (*nCard)*

int *nCard*; /* */

The **MPG4K_GetInputStandard** function returns the current input standard for the specified card.

Parameter	*Description*
nCard	Specifies which card to retrieve the standard for.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_PAL	PAL
ID_NTSC	NTSC

Comments

MPG4K_SetMode

int MPG4K_SetMode (*nCard*, *nMode)*

int *nCard*; /* */
int *nMode*; /* */

The **MPG4K_SetMode** function sets the mode for the encoder.

Parameter	*Description*
nCard	Specifies the card to set the mode for.
nMode	Specifies the mode to use. This can be one of the following values.

Value	*Meaning*
ID_QUAD_MODE	Encode the four channels as four separate files, each at CIF resolution
ID_QUAD_MODE_SINGLE	Encode the four channels arranged in four quadrants as a single file at D1 resolution with each channel being CIF
ID_SINGLE_MODE	Capture only one channel at D1 resolution
ID_MUXD1_MODE	Capture up to four channels at D1 resolution ¼ frame rate to four separate files.
ID_PIP_MODE	Capture one channel at D1 resolution and one channel at reduced resolution
ID_DUAL_MODE_SINGLE	Encode the first two channels arranged in two halves as a single file at D1 resolution

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_MODE	An invalid mode was specified
ID_ERR_ENCODING	The encoder is running and an unsupported mode change was requested

Comments

This function sets both preview and capture paths to the same mode. To set only the preview mode, use **MPG4K_SetPreviewMode** (page **117**).

This function overrides previous capture and preview mode settings.

It is not possible to change modes that would result in a change to the number of encoding channels. For example, changing from ID_SINGLE_MODE to ID_QUAD_MODE_SINGLE is allowed because both modes output a single MPEG4 stream. Changing from ID_SINGLE_MODE to ID_QUAD_MODE is not supported because that requires changing from a single stream to four streams.

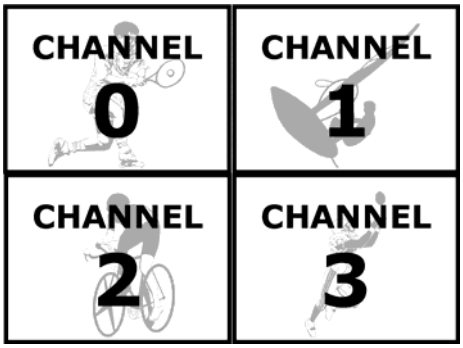
ID_SINGLE_MODE

In this mode a single channel fills the D1 capture window. The channel can be selected from on of the 4 possible decoders by using **MPG4K_SetInputPath** (page **28**). Further selection of the particular input to the selected decoder is possible using. **MPG4K_SelectInput** (page **27**)

ID_SINGLE_MODE

ID_QUAD_MODE_SINGLE

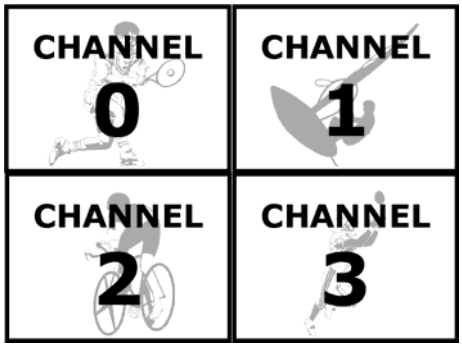
In this mode all 4 channels are scaled to equally sized quadrants and merged into a single D1 capture window. The selection of the particular input displayed and in which position is controlled by using **MPG4K_SetInputPath** (page **28**) and **MPG4K_SelectInput** (page **27**)



ID_QUAD_MODE_SINGLE

ID_QUAD_MODE

In this mode all 4 channels are scaled to equally sized quadrants. Four capture windows of CIF size are encoded. Encoding for each capture window must be enabled using **MPG4K_EnableChannel** (page **26**). Only channel 0 is currently supported in ID_QUAD_MODE.
The selection of the particular input displayed and in which position is controlled by using **MPG4K_SetInputPath** (page **28**) and **MPG4K_SelectInput** (page **27**).



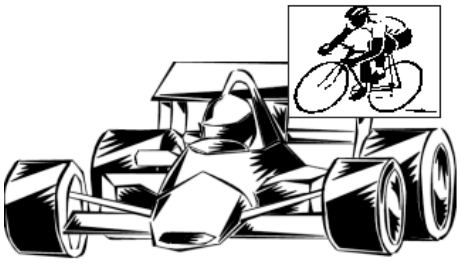
ID_QUAD_MODE

ID_PIP_MODE

In this mode channel 0 is set to full D1 size and channel 1 to a reduced size.

The capture and preview paths handle ID_PIP_MODE differently. In capture mode the smaller window will be CIF. In preview mode the smaller window is QCIF.

In order to capture ID_PIP_MODE with the smaller window as QCIF, it is necessary to route the digital video differently using the **MPG4KX_SetDigitalPath** (page **38**)function.



ID_PIP_MODE

ID_MUXD1_MODE

In this mode, all channels are encoded at full D1 size into separate files but time multiplexed. This means that the maximum frame rate per channel will depend on the number of channels being encoded.

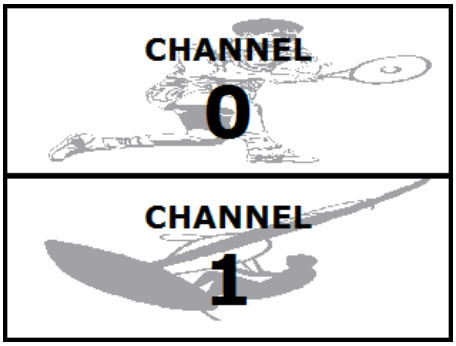
When using ID_MUXD1_MODE, the encoder will default to encoding all four channels at D1 to separate. files. To reduce the number of channels (and increase the maximum frame rate achievable per channel) calling the **MPG4KX_SetNumChannels** (page **49**) function. The maximum frame rate can be calculated using

$$MaxFPS = \frac{InputFPS}{nChannels}$$

InputFPS is determined by MPG4KX_**SetMuxInputSequence** (page **51**)

ID_DUAL_MODE_SINGLE

In this mode the first two channels are scaled vertically to be half height and merged into a single D1 capture window. The selection of the particular input displayed and in which position is controlled by using **MPG4K_SetInputPath** (page **28**) and **MPG4K_SelectInput** (page **27**)



ID_DUAL_MODE_SINGLE

MPG4K_SetEncodingType

int MPG4K_SetEncodingType (*nCard*, *nChannel*, *nType)*

int *nCard*; /* */
int *nChannel*; /* */
int *nType*; /* */

The **MPG4K_SetEncodingType** function sets the type of encoding.

Parameter	***Description***
nCard	Specifies the card to set the encoding type for
nChannel	Specifies the channel to set the encoding type for
nType	Specifies the encoding type. This can be one of the following values

Value	***Meaning***
ID_SYSTEM	Audio and video are recorded
ID_VIDEO	Video only is recorded

Returns

The return value is one of the following values.

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_MODE	An invalid mode was specified

Comments

This only affects the recording to AVI file.

This function only takes effect the next time an AVI file is created.

MPG4K_EnableChannel

int MPG4K_EnableChannel (*nCard*, *nChannels, nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4K_EnableChannel** function enables the specific channels for capture.

Parameter	***Description***	
nCard	Specifies the card to enable the channels for	
nChannel	Specifies the channel to enable.	
nEnable	Specifies whether to enable or disable the channel. This can be one of the following values.	
	Value	***Meaning***
	ID_ENABLE	Enable the channel
	ID_DISABLE	Disable the channel

Returns

The return value is one of the following values.

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified.

Comments

If the encoder is running, enabling the channel using this function has no effect until the next I-frame is received.
Disabling a channel has the effect of stopping all data being passed into the callback chain. This can be used to pause saving data to disk.
The files are not closed while the channel is been disabled.

MPG4K_SelectInput

int MPG4K_SelectInput (*nCard*, *nChannel*, *nInput)*

int *nCard*; /* */
int *nChannel*; /* */
int *nInput*; /* */

The **MPG4K_SelectInput** function selects the input to the specified channel.

Parameter	*Description*
nCard	Specifies which card to set the input routing for.
nChannel	Specifies the major input group..
nInput	Specifies the input to the group

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified

Comments

On the MPEG4-CPCI, nChannel should always be 0. This is because group A on the cable corresponds to card 0, group B to card 1 etc.

nChannel 0 corresponds to group A on the cable

nChannel 1 corresponds to group B on the cable

nChannel 2 corresponds to group C on the cable

nChannel 3 corresponds to group D on the cable

nInput 0 corresponds to input 1 in the cable group.

nInput 1 corresponds to input 2 in the cable group.

MPG4K_SetInputPath

int MPG4K_SetInputPath (*nCard*, *nChannel*, *nInput)*

int *nCard*; /* */
int *nChannel*; /* */
int *nInput*; /* */

The **MPG4K_SetInputPath** function sets the routing for the video inputs.

Parameter	*Description*
nCard	Specifies which card to set the input routing for.
nChannel	Specifies the destination channel.
nInput	Specifies the input to route to the specified channel

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_MODE	The current mode is MUXD1

Comments

This function is only available on the MPEG4000-XLP.

Input 0 corresponds to input group A

Input 1 corresponds to input group B

Input 2 corresponds to input group C

Input 3 corresponds to input group D

It is possible to route the same input to multiple channels.

The input routing can be modified without stopping the encoding.

In ID_SINGLE_MODE only channel 0 is encoded. Combining this function with **MPG4K_SelectInput** (page **27**) allows a single channel encoding of any of the 8 possible inputs.

This function cannot be used if using ID_MUXD1_MODE.

This function sets the routing for both the capture and preview paths. To set the input routing for only the preview path use the **MPG4K_SetPreviewInputPath** (page **118** function.

MPG4K_SetEncodingMode

int MPG4K_SetEncodingMode (*nCard*, *nChannel*, *nMode)*

int *nCard*; /* */
int *nChannel*; /* */
int *nMode*; /* */

The **MPG4K_SetEncodingMode** function sets the encoding mode for the MPEG4000.

Parameter	***Description***
nCard	Specifies the card to set the encoding mode for
nChannel	Specifies the channel to set the encoding mode for
nMode	Specifies the encoding mode. This can be one of the following values.

Value	***Meaning***
ID_VBR	Variable Bit Rate. The bit rate will change but the quality will stay the same
ID_CBR	Constant Bit Rate. The bit rate will stay constant and the quality will change.
ID_HBR	Hybrid Bit Rate. This is a hybrid of VBR and CBR. A minimum and maximum bit rate is specified and the encoder will try to use up to the maximum only when necessary.

Returns

The return value is one of the following values.

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel was specified
ID_ERR_INVALID_MODE	An invalid mode was specified.

Comments

MPG4K_SetQuality

int MPG4K_SetQuality (*nCard*, *nChannel*, *nQuality)*

int *nCard*; /* */
int *nChannel*; /* */
int *nQuality*; /* */

The **MPG4K_SetQuality** function sets the quality factor for use in VBR mode.

Parameter	*Description*
nCard	Specifies the card to set the quality for
nChannel	Specifies the channel to set the quality for
nQuality	Specifies the quality. This can be in the range 1 to 31 inclusive where 1 is highest quality and 31 is the lowest.

Returns

The return value is one following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_INVALID_MODE	An invalid quality factor was specified.

Comments

The default quality factor is 5.

Calling this function when in CBR or HBR mode will reset the current encoding quality quantisation. CBR/HBR encoding constantly changes the quantisation to vary the quality and keep the bitrate constant.

MPG4K_SetBitrates

int MPG4K_SetBitrates (*nCard*, *nChannel*, *ulBitrate*, *ulMinBitrate)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulBitrate*; /* */
unsigned long *ulMinBitrate*; /* */

The **MPG4K_SetBitrates** function sets the bit rate for use in CBR or HBR encoding mode.

Parameter	*Description*
nCard	Specifies the card to set the bit rate for
nChannel	Specifies the channel to set the bit rate for
ulBitrate	Specifies the bit rate. In the case of ID_HBR mode, this specifies the maximum bit rate. This value can be in the range 20000 up to 7500000.
ulMinBitrate	Specifies the Minimum bit rate. This is only used for ID_HBR mode. This value can be in the range 20000 up to 7500000

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_INVALID_BITRATE	An invalid bit rate was specified.

Comments

If the bit rate is too low then the encoder will not obey the bit rate limit and will provide data at a higher bit rate.

Typically, the minimum bit rate should be at least 350000 for ID_QUAD_MODE and 1000000 for ID_SINGLE_MODE and ID_QUAD_MODE_SINGLE.

For video with low motion or lower frame rates, lower values can be used.

When using CBR, frame skipping can be enabled using the **MPG4KX_SetFrameSkip** (page **43**) function. This will allow the encoder to insert small, dummy frames if the bit rate gets much above the requested value.

MPG4K_SetFrameRate

int MPG4K_SetFrameRate (*nCard*, *nFrameRate)*

int *nCard*; /* */
int *nFrameRate*; /* */

The **MPG4K_SetFrameRate** function sets the frame rate for the specified card.

Parameter	***Description***
nCard	Specifies the card to set the frame rate for
nFrameRate	Specifies the frame rate. This can be one of the following values:

Value	***Meaning for PAL***	***Meaning for NTSC***
0	25	30
1	12.5	15
2	6.25	7.5
3	3.125	3.75
4	1.562	1.865
5	0.781	0.938

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_FRAMERATE	An invalid frame rate was specified.

Comments

When using ID_MUXD1_MODE, the frame rates given above should be divided by the number of channels being encoded.
This function can only be called after **MPG4K_SetInputStandard** (page **20**)

MPG4KX_SetFrameRate

int MPG4KX_SetFrameRate (*nCard*, *nChannel, nFrameRate)*

int *nCard*; /* */
int *nChannel*; /* */
int *nFrameRate*; /* */

The **MPG4KX_SetFrameRate** function sets the frame rate for the specified card.

Parameter	*Description*
nCard	Specifies the card to set the frame rate for
nChannel	Specifies the channel to set the frame rate for
nFrameRate	Specifies the frame rate. This is in the form of an integer divider of the input frame rate. Full frame rate is 65536.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_FRAMERATE	An invalid frame rate was specified.

Comments

The frame rate can be calculated from the nFrameRate using

$$OutputFPS = \frac{65536 * InputFPS}{nFrameRate}$$

Alternatively, the nFrameRate value can be calculated for a specified output frame rate using

$$nFrameRate = \frac{InputFPS * 65536}{OutputFPS}$$

When using ID_MUXD1_MODE, the InputFPS used above should be modified to be the MaxFPS given by the equation in **MPG4K_SetMode** (page **22**)

This function can only be called after **MPG4K_SetInputStandard** (page **20**)

MPG4K_SetIInterval

int MPG4K_SetIInterval (*nCard*, *nInterval)*

int *nCard*; /* */
int *nInterval*; /* */

The **MPG4K_SetIInterval** function sets the interval of I-frames.

Parameter	***Description***
nCard	Specifies which MPEG4000 card to set the I-frame interval for
nInterval	Specifies the I-frame interval. This can be one of the following value

Value	***Meaning***
1	Every frame will be a key frame
2	Every other frame will be a key frame
4	Every fourth frame will be a key frame
8	Every eighth frame will be a key frame
16	Every sixteenth frame will be a key frame
32	Every 32nd frame will be a key frame
64	Every 64th frame will be a key frame
128	Every 128th frame will be a key frame
256	Every 256th frame will be a key frame

Returns

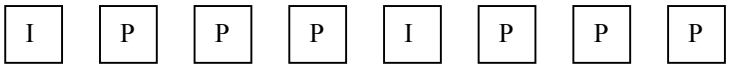
The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_INTERVAL	An invalid interval number was specified

Comments

In general, key frames have a larger size than non key frames.
When using VBR encoding, specifying a lower interval number will increase the bandwidth used.
When using CBR encoding, specifying a lower interval number will decrease the quality of the video.
Specifying a large interval number can cause quantization build up errors to be more noticeable.
The I-frame interval will only change after the next I-frame in the sequence is received.

For an I-frame interval of 4, the frame pattern would be



MPG4KX_SetIInterval

int MPG4KX_SetIInterval (*nCard*, *nChannel, nInterval)*

int *nCard*; /* */
int *nChannel*; /* */
int *nInterval*; /* */

The **MPG4KX_SetIInterval** function sets the interval of I-frames.

Parameter	*Description*
nCard	Specifies which MPEG4000 card to set the I-frame interval for
nChannel	Specifies the channel to set the I-frame interval for
nInterval	Specifies the I-frame interval. This is specified as the number of P-frame between I-frames.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_INTERVAL	An invalid interval number was specified

Comments

See **MPG4K_SetIInterval** (page **34**)

MPG4K_SetAudioFormat

int MPG4K_SetAudioFormat (*nCard*, *nChannel, nInterval)*

int *nCard*; /* */
int *nChannel*; /* */
int *nInterval*; /* */

The **MPG4K_SetAudioFormat** function sets the format of the audio.

Parameter	***Description***
nCard	Specifies on which MPEG4000 card to set the audio format
nChannel	Specifies the channel on which to set the audio format
nInterval	Specifies the audio format. This can be one of the following values:

Value	***Meaning***
ID_MULAW	8000Hz 64Kbit/s uLaw encoded
ID_ADPCM	8000Hz 32Kbit/s ADPCM encoded (default)

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_FORMAT	An invalid audio format was specified

Comments

Changing the audio format while the encoder is running and saving to disk will result in a corrupted audio stream.

MPG4K_SetOutputType

int MPG4K_SetOutputType (*nCard*, *nChannel*, *bOutput)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned char *bOutput*; /* */

The **MPG4K_SetOutputType** function sets the output type for the specified channel.

Parameter	***Description***
nCard	Specifies the card to set the output type for
nChannel	Specifies the channel to set the output type for
bOutput	Specifies the output type. This is a bit mask of the following values

Value	***Meaning***
ID_AVI_BIT	Output to AVI file
ID_MP4_BIT	Output to MP4 file

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_ENCODING	The encoder is running

Comments

If outputting to MP4 file, the audio will not be saved in the resultant MP4 file. This is because MP4 files do not support ADPCM or mu-Law audio.

This function can only be called when the encoder is in a stopped state.

To allow file type selection while the encoder is running, enable all file types using this function but do set a NULL filename. Setting a filename will then cause the file to be created.

MPG4KX_SetDigitalPath

int MPG4KX_SetDigitalPath (*nCard*, *nPath*, *nInput)*

int *nCard*; /* */
int *nPath*; /* */
int *nInput*; /* */

The **MPG4KX_SetDigitalPath** function routes the digital video to different paths.

Parameter	*Description*	
nCard	Specifies the card to set the output type for	
nPath	Specifies the input of the encoder engine to receive the digital video specified by nInput. Bit mask of the following values.	
	Value	***Meaning***
	ID_PREVIEW	Preview input
	ID_CAPTURE	Capture input
nInput	Specifies the output of the video decoder to route to the specified encoder input	
	Value	***Meaning***
	ID_PREVIEW	Preview output
	ID_CAPTURE	Preview output

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified

Comments

Due to differences in the capabilities of the different video decoder paths, this function can be used to change which video decoder output is used for preview and capture by the encoder engine.

This function changes the digital path for the encoder and analogue output.

To change only the analogue output path use **MPG4KX_SetAnalogPath** (page **137**)

MPG4K_SetPIPPosition

int MPG4K_SetPIPPosition (*nCard*, *nX*, *nY)*

int *nCard*; /* */
int *nX*; /* */
int *nY*; /* */

The **MPG4K_SetPIPPosition** function specifies the position of the PIP window.

Parameter	*Description*
nCard	Specifies the card to set the output type for
nX	Specifies the horizontal location of the PIP window
nY	Specifies the vertical location of the PIP window

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified

Comments

The position of the PIP window can only be set on the preview path.

MPG4KX_SetPIPSize

int MPG4KX_SetPIPSize (*nCard*, *nW*, *nH)*

int *nCard*; /* */
int nW; /* */
int *nH*; /* */

The **MPG4KX_SetPIPSize** function specifies the size of the PIP window.

Parameter	*Description*
nCard	Specifies the card to set the output type for
nW	Specifies the horizontal size of the PIP window
nH	Specifies the vertical size of the PIP window

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified

Comments

The size of the PIP window can only be set on the preview path.

MPG4K_GetFrameCount

int MPG4K_GetFrameCount (*nCard)*

int *nCard*; /* */

The **MPG4K_GetFrameCount** function returns the number of frames encoded by the MPEG4000 card specified.

Parameter	*Description*
nCard	Specifies the card to return the frame count for

Returns

The return value is the number of frames encoded by the specified MPEG4000XLP or MPEG4CPCI.

Comments

MPG4K_WaitForFirstFrame

int MPG4K_WaitForFirstFrame (*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_WaitForFirstFrame** function waits for the first frame to be captured from the specified channel on the specified card.

Parameter	*Description*
nCard	Specifies which card to wait for the first frame for
nChannel	Specifies which channel to wait for the first frame

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_TIMEOUT	The wait timed out.

Comments

The wait will time out after 10seconds if the first frame has not been received.

MPG4KX_SetFrameSkip

int MPG4KX_SetFrameSkip (*nCard*, *nChannel, nFrameSkip)*

int *nCard*; /* */
int *nChannel*; /* */
int *nFrameSkip*; /* */

The **MPG4KX_SetFrameSkip** function enables or disables and sets the skip factor.

Parameter	*Description*
nCard	Specifies on which card to set the skip factor
nChannel	Specifies for which channel to set the skip factor
nFrameSkip	Specifies the skip factor. This determines how often frames may be skipped. Valid values are between 0 and 31 inclusive. 0 disables frame skipping. Lower values result in more frame skipping.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Frame skipping can be used in CBR mode to help keep the bit rate within the requested bit rate.

If the bit rate goes above the requested bit rate and frame skipping is enabled then small dummy frames are used instead of the actual frame.

Please note that the dummy frames represent no motion and so the playback may appear to judder or jump.

MPG4KX_SetInputScale

int MPG4KX_SetInputScale (*nCard*, *nWidth, nHeight)*

int *nCard*; /* */
int *nWidth*; /* */
int *nHeight*; /* */

The **MPG4KX_SetInputScale** function sets the width and height the input video is scaled to before encoding.

Parameter	*Description*
nCard	Specifies on which card to set the input scale
nWidth	Specifies the width (in pixels) that the video input should be scaled to
nHeight	Specifies the height (in lines) that the video input should be scaled to

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_MODE	Scaling is not allowed when in quad mode
ID_ERR_INVALID_LOCATION	An invalid width or height was specified
ID_ERR_ENCODING	The encoder is running

Comments

This function can not be used when the encoding mode is ID_QUAD_MODE.

Calling **MPG4K_SetMode** (page **22**) will override the scaling specified by this function.

This function can only be used when the encoder is in a stopped state.

MPG4KX_SetChannelScale

int MPG4KX_SetChannelScale (*nCard*, *nChannel, nWidth, nHeight)*

int *nCard*; /* */
int *nChannel*; /* */
int *nWidth*; /* */
int *nHeight*; /* */

The **MPG4KX_SetChannelScale** function sets the width and height the specified channels input video is scaled to before encoding.

Parameter	*Description*
nCard	Specifies on which card to set the input scale
nChannel	Specifies which input channel to scale
nWidth	Specifies the width (in pixels) that the video input should be scaled to
nHeight	Specifies the height (in lines) that the video input should be scaled to

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_MODE	Scaling is not allowed when in quad mode
ID_ERR_INVALID_LOCATION	An invalid width or height was specified
ID_ERR_ENCODING	The encoder is running

Comments

This function can only be used when the encoding mode is ID_MUXD1_MODE.

Calling **MPG4K_SetMode** (page **22**) after this function will override the scaling specified by this function.

This function can be used in combination with **MPG4KX_SetEncoderLocation** (page **46**) to encode different channels at different resolutions.

This function can only be called when the encoder is in a stopped state.

MPG4KX_SetEncoderLocation

int MPG4KX_SetEncoderLocation (*nCard*, *nChannel, lpRect)*

int *nCard*; /* */
int *nChannel*; /* */
RECT **lpRect*; /* */

The **MPG4KX_SetEncoderLocation** function sets location of the encoding window within the input video.

Parameter	*Description*
nCard	Specifies on which card to set the input scale
nChannel	Specifies for which channel to set the encoder window
lpRect	Pointer to a RECT structure that contains the location

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified.
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The lpRect pointer was invalid
ID_ERR_INVALID_MODE	Scaling is not allowed when in quad mode
ID_ERR_INVALID_LOCATION	An invalid location was specified

Comments

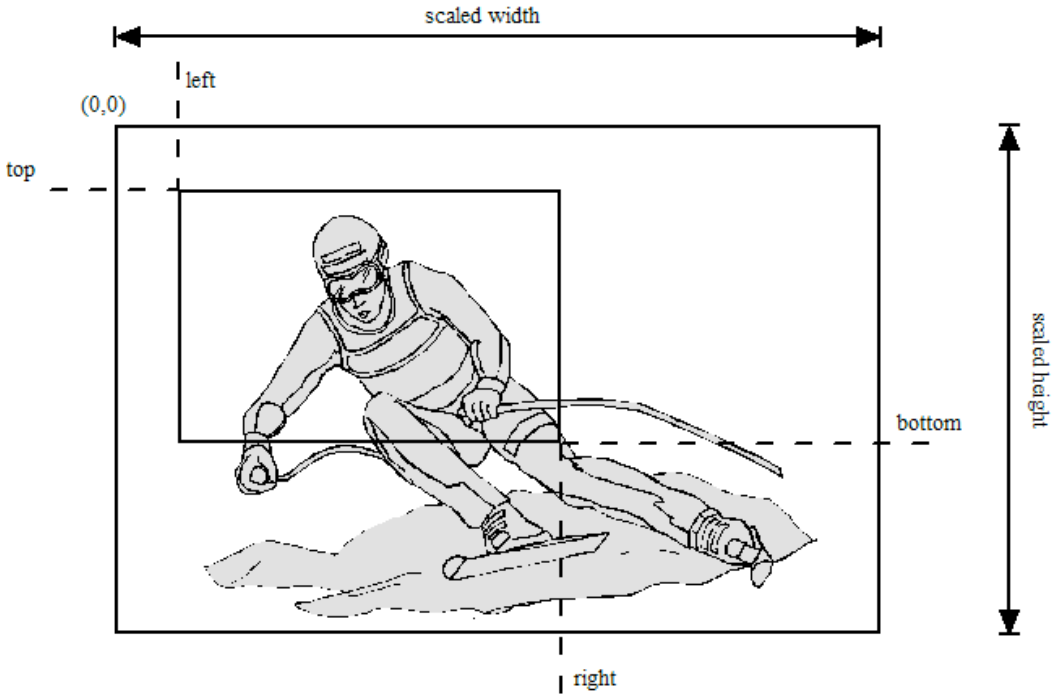
This function can not be used when the encoding mode is ID_QUAD_MODE.

If the encoding mode is not ID_MUXD1_MODE, the location of the encoding window can be set for channel 0 only.

The width and height of the encoding window must be whole multiple of 16.

Calling **MPG4K_SetMode** (page **22**) will override the location specified by this function.

The diagram shows how the encoding window can select a sub-window of the scaled input.



The scaled width and height are set using **MPG4KX_SetInputScale** (page **44**). The output from the encoder will include only the video within the window given by left, right, top and bottom.

The size of encoder window must be less than or equal to the size scaled input window.

The encoder window must fit within the scaled input window.

The scaled window size should be set be calling this function.

MPG4K_SetAudioPath

int MPG4K_SetAudioPath (*nCard*, *nChannel*, *nAudioInput)*

int *nCard*; /* */
int *nChannel*; /* */
int *nAudioInput*; /* */

The **MPG4K_SetAudioPath** function sets the routing for the audio inputs.

Parameter	*Description*
nCard	Specifies which card to set the input routing for.
nChannel	Specifies the destination channel.
nAudioInput	Specifies the input to route to the specified channel

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified

Comments

This function is only available on the MPEG4000-XLP.

Input 0 corresponds to input Audio_In A

Input 1 corresponds to input Audio_In B

Input 2 corresponds to input Audio_In C

Input 3 corresponds to input Audio_In D

It is possible to route the same input to multiple channels.

The input routing can be modified without stopping the encoding.

In ID_SINGLE_MODE only channel 0 is encoded.

MPG4KX_SetNumChannels

int MPG4KX_SetNumChannels(*nCard*, *nChannels)*

int *nCard*; /* */
int *nChannels*; /* */

The **MPG4KX_SetNumChannels** function sets the number of channels that ID_MUXD1_MODE should use.

Parameter	*Description*
nCard	Specifies which card to control
nChannels	Specifies the number of channels to encode. Valid values are between 1 and 4 inclusive

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid number of channels was specified
ID_ERR_ENCODING	The encoder is running

Comments

This function is only available on the MPEG4000-XLP.
This function must be called before MPG4K_SetMode is called to set ID_MUXD1_MODE.
This function only has an effect on ID_MUXD1_MODE.
The channels encoded start at channel 0 and increase up to the specified number. For example, nChannels=3 would give channels 0, 1 and 2.
Re-routing the video inputs using **MPG4K_SetInputPath** (page **28**) cannot be done in ID_MUXD1_MODE.
This function can only be called when the encoder is in a stopped state.

MPG4KX_EnableQTCompatibility

int MPG4KX_EnableQTCompatibility(*nCard*, *nChannel, nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4KX_EnableQTCompatibility** function enables MPEG4 compatibility with Apple QuickTime.

Parameter	***Description***
nCard	Specifies which card to control
nChannel	Specifies for which channel to enable QuickTime compatibility
nEnable	Specifies whether to enable or disable QuickTime compatibility. This can be one of the following value

Value	***Meaning***
ID_ENABLE	Enable QuickTime compatibility
ID_DISABLE	Disable QuickTime compatibility

Returns

The return value is one of the following values.

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid number of channels was specified

Comments

MPG4KX_SetMuxInputSequence

int MPG4KX_SetMuxInputSequence(*nCard*, *pSequence, nNumEntries)*

int *nCard*; /* */
int **pSequence*; /* */
int *nNumEntries*; /* */

The **MPG4KX_SetMuxInputSequence** function sets the channel sequence for MUXD1 mode.

Parameter	*Description*
nCard	Specifies which card to control
pSequence	Array of int, each element specifies the next channel to display
nNumEntries	Specifies the number of entries in the pSequence array. Must be less than or equal to 127

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	An invalid pSequence pointer was specified

Comments

This function is only available on the MPEG4000-XLP.
This function must be called after MPG4K_SetMode is called to set ID_MUXD1_MODE.
This function only has an effect on ID_MUXD1_MODE.
The input multiplexor will cycle through the inputs as specified by the pSequence array, returning to the start when nNumEntries has been reached.

The default behaviour of ID_MUXD1_MODE is to show all inputs equally.

The input frame rate to each encoder channel will be determined by the proportion of each channels entries in the pSequence array.

Examples

Value	*Meaning*
int Sequence[]={0,1,2,3}	Default for 4 channel MUXD1. Each channel is 1/4 frame rate
int Sequence[]={0,1,0,2,0,3}	Channel 0 is 1/2 (3/6) frame rate, channels 1,2 and 3 are 1/6 frame rate
int Sequence[]={0,0,0,0,1}	Channel 0 is 4/5 frame rate, channel 1 is 1/5 frame rate.
int Sequence[]= {0,0,1,0,0,2,0,0,1,0,0,1,0,0,2}	Channel 0 is 2/3 (10/15) frame rate, channel 1 is 1/5 (3/15) frame rate and channel 2 is 2/15 frame rate

MPG4KX_ShowPlayback

int MPG4KX_ShowPlayback(*nCard*, *nEnable)*

int *nCard*; /* */
int *nEnable*; /* */

The **MPG4KX_ShowPlayback** function configures the video output source.

Parameter	***Description***
nCard	Specifies which card to control
nEnable	Specifies whether to enable or disable playback. Can be one of the following values

Value	***Meaning***
ID_ENABLE	Output the decoder output
ID_DISABLE	Output the encoder input

Returns

The return value is one of the following values.

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The card specified does not support this function

Comments

This function is available on all MPEG4000-XLP boards attached to the system or the first channel of the MPEG4-CPCI board.

By default, the MPEG4000XLP will output the decode output when decoding.
This function should be used with the MPEG4-CPCI to enable decode output.

MPG4KX_SetFrameMode

int MPG4KX_SetFramemode(*nCard, nChannel*, *nFrameMode)*

int *nCard*; /* */
int *nChannel*; /* */
int *nFrameMode*; /* */

The **MPG4KX_SetFrameMode** function configures the encoder frame mode.

Parameter	***Description***
nCard	Specifies which card to control
nChannel	Specifies for which channel to set the frame mode
nFrameMode	Specifies the frame mode. Can be one of the following values

Value	***Meaning***
ID_FRAME_MODE	Interleave the interlaced video
ID_FIELD_MODE	Keep the fields separate.

Returns

The return value is one of the following values.

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The card specified does not support this function

Comments

When ID_FIELD_MODE is used, the two fields of the input video will be not be interleaved but will appear with the top field at the top and the bottom field at the bottom.
This can be used to encode a single field by selecting sub-region using the **MPG4KX_SetEncoderLocation** (page **46**)

MPG4K_GetFIFOLevel

int MPG4K_GetFIFOLevel(*nCard)*

int *nCard*; /* */

The **MPG4K_GetFIFOLevel** function returns the fill level of the kernel driver data FIFO.

Parameter	*Description*
nCard	Specifies from which card to get the FIFO level

Returns

The return value is the fill level of the FIFO, in bytes.

Comments

MPG4KX_GetFrameQueueCount

int MPG4KX_GetFrameQueueCount(*nCard*, *nChannel, nQueue)*

int *nCard*; /* */
int *nChannel*; /* */
int *nQueue*; /* */

The **MPG4KX_GetFrameQueueCount** function returns the number frames of the specified type that are queued for the callback chain.

Parameter	***Description***
nCard	Specifies which card to control
nChannel	Specifies for which channel to get the frame count
nQueue	Specifies the frame type. This can be one of the following values

Value	***Meaning***
ID_VIDEO	Return the number of video frames
ID_AUDIO	Return the number of audio frames
-1	Return the number of all frames

Returns

The return value is one of the following values.

Value	***Meaning***
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NOT_INITIALISED	The callback chain has not been initialised
All other positive values	The number of frames

Comments

MPG4KX_EnableExtraFrameInfo

int MPG4KX_EnableExtraFrameInfo(*nCard*, *nChannel, nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4KX_EnableExtraFrameInfo** function enables or disables the extra frame information appended to each MPEG4 video frame.

Parameter	***Description***	
nCard	Specifies which card to control	
nChannel	Specifies for which channel to enable the extra information	
nEnable	Specifies whether to enable or disable. This can be one of the following values	
	Value	***Meaning***
	ID_ENABLE	Enable the extra frame information
	ID_DISABLE	Disable the extra frame information

Returns

The return value is one of the following values.

Value	***Meaning***
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified

Comments

AVI functions

MPG4K_SetAVIFilename

int MPG4K_SetAVIFilename (*nCard*, *nChannel*, *szFilename)*

int *nCard*; /* */
int *nChannel*; /* */
char **szFilename*; /* */

The **MPG4K_SetAVIFilename** function sets the filename for the AVI output.

Parameter	*Description*
nCard	Specifies the card to set the filename for
nChannel	Specifies the channel to set the filename for
**szFilename*	NULL terminated string specifying the filename

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOTFOUND	This function was called to close a file that was not open
ID_ERR_TIMEOUT	Timed out waiting to close the file

Comments

If the encoder is running when this function is called, the old file will be closed from the next I-frame and all subsequent data will be saved to the new filename. If an error occurs when trying to open the new file, an error will be sent to the asynchronous error handler installed by the **MPG4K_SetErrorCallback** (page **86**) function

To close the file without stopping the encoder or starting a new file, call this function with szFilename=NULL.

When closing the previous file, there is a 500ms timeout while waiting for control of the file. If the frame write blocks for more than 500ms (for example if the write to disk blocks) then the ID_ERR_TIMEOUT error will be signalled and the file will remain open.

The actual closure of the current AVI file is asynchronous to this call. To find out if the previous file has been fully closed call the **MPG4KX_AVIFileOpen** (page **67**)

MPG4K_SetAVIBufferSize

int MPG4K_SetAVIBufferSize (*nCard*, *nChannel*, *ulBufferSize)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulBufferSize*; /* */

The **MPG4K_SetAVIBufferSize** function sets the buffer size for the AVI file in memory.

Parameter	*Description*
nCard	Specifies the card
nChannel	Specifies which channel
ulBufferSize	Specifies the buffer size to use, in bytes.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	The specified MPEG4000 could not be opened or was not detected.

Comments

The AVI file is created in memory before being flushed to disk. This function specifies how much memory to use before flushing to disk.

MPG4K_FlushAVIBuffer

int MPG4K_FlushAVIBuffer (*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_FlushAVIBuffer** function flushes the AVI buffer to disk.

Parameter	*Description*
nCard	Specifies the card
nChannel	Specifies the channel to flush the AVI buffer for

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

This function can be used to prematurely flush the AVI buffer to disk.

MPG4K_AddInfoChunk

int MPG4K_AddInfoChunk (*nCard*, *nChannel*, *ulID, szString)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulID*; /* */
char * *szString*; /* */

The **MPG4K_AddInfoChunk** function adds the specified ID and string to the INFO chunk in the saved AVI file.

Parameter	*Description*
nCard	Specifies the card
nChannel	Specifies which channel
ulID	Specifies the FOURCC ID of the new info chunk
szString	NULL terminated string to save in the info chunk

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	The specified MPEG4000 could not be opened or was not detected.

Comments

To remove a previously defined INFO chunk ID, pass szString=NULL

MPG4K_DisableCreationDate

int MPG4K_DisableCreationDate (*nCard*, *nChannel*, *nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4K_DisableCreationDate** function disables the addition of file creation timestamp to the AVI INFO chunk.

Parameter	***Description***	
nCard	Specifies the card	
nChannel	Specifies which channel	
nEnable	.Specifies whether to enable or disable the creation date.	
	Value	***Meaning***
	ID_ENABLE	Enable creation date
	ID_DISABLE	Disable creation date

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	The specified MPEG4000 could not be opened or was not detected.

Comments

MPG4K_SetVideoFOURCC

int MPG4K_SetVideoFOURCC (*nCard*, *nChannel*, *ulFOURCC)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulFOURCC*; /* */

The **MPG4K_SetVideoFOURCC** function sets the FOURCC in the AVI for the specified channel.

Parameter	*Description*
nCard	Specifies which card
nChannel	Specifies which channel to set the FOURCC for
ulFOURCC	Specifies the FOURCC to use in the AVI file.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The FOURCC must be set before the file is created by calling **MPG4K_Start** (page **17** or before changing the name with **MPG4K_SetAVIFilename**(page) **57**

The value for the ulFOURCC parameter can be obtained by using the mmioFOURCC function. For example, the value for DIVX would be the return value from

```
mmioFOURCC('d','i','v','x')
```

mmioFOURCC is defined as:

```
#define mmioFOURCC( ch0, ch1, ch2, ch3 ) \
( (unsigned long)(unsigned char)(ch0) |\
( (unsigned long)(unsigned char)(ch1)<< 8 ) | \
( (unsigned long)(unsigned char)(ch2) << 16 ) | \
( (unsigned long)(unsigned char)(ch3) << 24 ) )
```

MPG4K_EnablePrivateData

int MPG4K_EnablePrivateData (*nCard*, *nChannel*, *nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4K_EnablePrivateData** function enables or disables storage of the private data side band in the AVI file.

Parameter	*Description*
nCard	Specifies the card
nChannel	Specifies the channel to enable or disable the private data
nEnable	Specifies whether to enable or disable the private data side band storage. This can be one of the following values

Value	*Meaning*
ID_ENABLE	Enable storage of the private data
ID_DISABLE	Disable storage of the private data

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

This function must be called before **MPG4K_SetAVIFilename** (page **57**)

Calling this function after MPG4K_SetAVIFilename has no effect until the next call to these functions.

The default is for private side band data to not be stored in the AVI file.

MPG4K_RegisterPrivate

int MPG4K_RegisterPrivate (*nCard*, *ulFOURCC)*

int *nCard*; /* */
unsigned long *ulFOURCC*; /* */

The **MPG4K_RegisterPrivate** function registers a FOURCC for use with the private side band data.

Parameter	*Description*
nCard	Specifies the card
ulFOURCC	Specifies a FOURCC code to register for use with the private side band data.

Returns

The return value is one of the following values

Value	*Meaning*
All other values	ID number of the FOURCC
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The FOURCC is stored with the private side band data in the AVI file to identify the type of data.
Each type of private data stored should use a unique FOURCC in order to identify it.
The ID number returned by this function should be used when calling **MPG4K_WritePrivateData** (page **65**)

MPG4K_WritePrivateData

int MPG4K_WritePrivateData (*nCard*, *nChannel*, *nDataID*, **bpData, nSize)*

int *nCard*; /* */
int *nChannel*; /* */
int *nDataID*; /* */
unsigned char *bpData; /* */
int nSize; /* */

The **MPG4K_WritePrivateData** function writes the data specified by bpData to the AVI file as part of the side band data.

Parameter	*Description*
nCard	Specifies which card to write the private data for
nChannel	Specifies which channel write the private data for
nDataID	Specifies the ID of the private data.
bpData	Pointer to the data to write to the private data side band in the AVI
nSize	Specifies the size of the data in bytes

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOTFOUND	The ID number specified by nDataID was not found in the private data FOURCC list

Comments

nDataID must be the ID returned by the call to **MPG4K_RegisterPrivate** (page **64**) to register the FOURCC for the type of this data

MPG4K_SetPrivateRate

int MPG4K_SetPrivateRate (*nCard*, *nChannel*, *ulRate*, *ulScale)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulRate*; /* */
unsigned long *ulScale*; /* */

The **MPG4K_SetPrivateRate** function sets the rate that the private data will be saved to AVI file.

Parameter	*Description*
nCard	Specifies which card to set the private data rate for.
nChannel	Specifies which channel to set the private data rate for
ulRate	Specifies the rate.
ulScale	Specifies the scale

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The Rate and Scale is saved to the AVI file to tell the decoder how fast to play the private data.
The number of private data packets per second is calculated by dividing ulRate by ulScale.
If the actual rate of private data packets is less than sample rate specified by this function then dummy packets are saved to the AVI file.
If the actual rate of private data packets is greater than the sample rate specified by this function then the private data will play back too slowly.
The dummy private packets have a total overhead of 52bytes per packet.

MPG4KX_AVIFileOpen

int MPG4KX_AVIFileOpen(*nCard, nChannel, szFilename)*

```
int nCard;              /*  */
int nChannel;           /*  */
char *szFilename;       /*  */
```

The **MPG4KX_AVIFileOpen** function returns whether the specified AVI file has been fully closed.

Parameter	*Description*
nCard	Specifies which card to control
nChannel	Specifies for which channel to get the AVI file status from
szFilename	NULL terminated string specifying which file to get the status for.

Returns

The return value is one of the following values

Value	*Meaning*
0	File is not open
1	File is still open
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4KX_SetAVICloseFrameCount

int MPG4KX_SetAVICloseFrameCount (*nCard*, *nChannel*, *ulCount)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulCount*; /* */

The **MPG4KX_SetAVICloseFrameCount** function sets how often the AVI file closure should sleep .

Parameter	*Description*
nCard	Specifies which card to set the frame count for
nChannel	Specifies which channel to set the frame count for
ulCount	Specifies the number of frames between each sleep. Default is 250

Returns

The return value is 0

Comments

The AVI close is asynchronous to the **MPG4K_SetAVIFilename** (page **57**) call.
In order to not consume too much CPU, the library sleeps for 10ms after a specific number of index entries has been written. Without doing this, lower performance systems will have high CPU usage which can cause unwanted side effects.
Pass ulCount=0 to disable sleeping

MP4 functions

MPG4K_SetMP4Filename

int MPG4K_SetMP4Filename (*nCard*, *nChannel*, *szFilename)*

int *nCard*; /* */
int *nChannel*; /* */
char **szFilename*; /* */

The **MPG4K_SetMP4Filename** function sets the filename for the MP4 output.

Parameter	*Description*
nCard	Specifies the card to set the filename for
nChannel	Specifies the channel to set the filename for
**szFilename*	NULL terminated string specifying the filename

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOTFOUND	This function was called to close a file that was not open
ID_ERR_TIMEOUT	Timed out waiting to close the file

Comments

If the encoder is running when this function is called, the old file will be closed from the next I-frame and all subsequent data will be saved to the new filename. If an error occurs when trying to open the new file, an error will be sent to the asynchronous error handler installed by the **MPG4K_SetErrorCallback** (page **86**) function

To close the file without stopping the encoder or starting a new file, call this function with szFilename=NULL.

When closing the previous file, there is a 500ms timeout while waiting for control of the file. If the frame write blocks for more than 500ms (for example if the write to disk blocks) then the ID_ERR_TIMEOUT error will be signalled and the file will remain open.

Audio will not be saved to the file since the MP4 file format does not support ADPCM or mu-Law.

If Apple's QuickTime is to be the video player, it is necessary to enable compatibility using the **MPG4KX_EnableQTCompatibility** (page **50**) function.

MPG4KX_MP4FileOpen

int MPG4KX_MP4FileOpen(*nCard, nChannel, szFilename)*

```
int nCard;              /*  */
int nChannel;           /*  */
char *szFilename;       /*  */
```

The **MPG4KX_MP4FileOpen** function returns whether the specified MP4 file has been fully closed.

Parameter	*Description*
nCard	Specifies which card to control
nChannel	Specifies for which channel to get the MP4 file status from
szFilename	NULL terminated string specifying which file to get the status for.

Returns

The return value is one of the following values

Value	*Meaning*
0	File is not open
1	File is still open
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Pre and post trigger buffering functions

MPG4K_EnablePreTrigger

int MPG4K_EnablePreTrigger (*nCard*, *nChannel*, *nEnable, nPreBuffer, nPostBuffer)*

int *nCard*; /* */
int *nChannel*; /* */
int nE*nable*; /* */
int *nPreBuffer*; /* */
int *nPostBuffer*; /* */

The **MPG4K_EnablePreTrigger** function enables or disables pre-trigger and post-trigger buffering.

Parameter	***Description***
nCard	Specifies which card
nChannel	Specifies for which channel to enable the pre-trigger buffering
nEnable	Specifies whether to enable or disable the pre-trigger and post trigger buffering. This can be one of the following values:

Value	***Meaning***
ID_ENABLE	Enable the pre-trigger buffering
ID_DISABLE	Disable the pre-trigger buffering

nPreBuffer	Specifies the number of frames to pre-buffer. Passing 0 disables pre-buffering
nPostBuffer	Specifies the number of frames to post-buffer. Passing 0 disables post-buffering

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Enabling pre-trigger and post-trigger buffering causes the video and audio to be buffered before and after the trigger event is signalled.
The number of frames for pre-buffering will be rounded up to the next I-frame and will be the maximum number of frames buffered.
The number of frames for post-buffering will not be rounded up to the next I-frame
The trigger is set/cleared by calling MPG4K_TriggerPreBuffer
A filename change during the post-buffering phase will not occur until the first I-frame after the end of the phase unless triggering is reasserted during this period. In that case, the filename will be changed at the next I-frame

MPG4K_TriggerPreBuffer

int MPG4K_TriggerPreBuffer (*nCard*, *nChannel*, *nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int nE*nable*; /* */

The **MPG4K_EnablePreTrigger** function enables or disables pre-trigger and post-trigger buffering.

Parameter	***Description***	
nCard	Specifies which card	
nChannel	Specifies for which channel to set the trigger state	
nEnable	Specifies whether to trigger buffering. This can one of the following values:	
	Value	***Meaning***
	ID_ENABLE	Assert the trigger.
	ID_DISABLE	De-assert the trigger

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Pre-trigger buffering needs to be enabled using the MPG4K_EnablePreTrigger function
If buffering is enabled, the MPEG4000 will record to file as long as the trigger is set, including the specified number of frames encoded before and after the trigger was set.
Asserting the trigger takes precedence over the post triggering phase
Asserting the trigger causes the pre-trigger buffer to be emptied.

MPG4K_PostTriggering

int MPG4K_PostTriggering (*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_PostTriggering** function returns whether the post trigger buffering is still ongoing.

Parameter	*Description*
nCard	Specifies which card
nChannel	Specifies from which channel to get the post trigger buffer state

Returns

The return value is one of the following values

Value	*Meaning*
0	Not post trigger buffering
1	Post trigger buffering is still ongoing
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

This function can be used to poll if post triggering is still ongoing. This is useful since filename changes do not take effect until the first I-frame after the end of post triggering
The exception to this is if the trigger is reasserted during the post trigger period, in which case the filename will be changed at the next I-frame

MPG4KX_SetAlwaysOpenFile

int MPG4KX_SetAlwaysOpenFile (*nCard*, *nChannel*, ul*Files)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned long *ulFiles*; /* */

The **MPG4KX_SetAlwaysOpenFile** function sets whether the specified files types should always be opened when a new filename is requested.

Parameter	***Description***
nCard	Specifies which card to control
nChannel	Specifies for which channel to set file open flags
ulFiles	Specifies the file types on which files should be opened. This is a bit mask of the following values:

Value	***Meaning***
ID_AVI_BIT	AVI files should always be created

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

This function can be used to defer file creation until data is ready to be written when using pre-trigger buffering.
If file creation is not set to always then the file will only be created once the trigger is asserted.
If file creation is set to always then the file will be created upon the call to MPG4K_SetAVIFilename, even if the trigger is not asserted. This could lead to empty AVI files.

MPG4KX_SingleShotTrigger

int MPG4KX_SingleShotTrigger (*nCard*, *nChannel, *szFilename, nEventDuration)*

int *nCard*; /* */
int *nChannel*; /* */
char **szFilename*; /* */
int *nEventDuration*; /* */

The **MPG4KX_SingleShotTrigger** function causes the specified file to be created with the currently pre-buffered data.

Parameter	*Description*
nCard	Specifies which card
nChannel	Specifies for which channel to generate the single shot trigger
**szFilename*	NULL terminated string specifying the filename that should be created for this trigger event.
nEventDuration	Specifies the duration of the event in frames. This can also be 0 and -1

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_ENCODING	The encoder is not currently running
ID_ERR_NOT_INITIALISED	The callback chain is not initialised
ID_ERR_TIMEOUT	Timed out waiting to flush to disk
ID_ERR_AVISTART	Error creating the AVI file
ID_ERR_MP4START	Error creating the MP4 file
All positive values	Trigger ID

Comments

This function can be used to create a file for a single shot trigger. Unlike asserting the main trigger, this function does not change the buffered data.
The pre and post buffer duration are controlled by **MPG4K_EnablePreTrigger** (page **71**).
The nEventDuration specifies how long the event should last for, in frames. This means that the total duration of the created file will be at least nPreBuffer+nEventDuration+ nPostBuffer frames.

Generating a single shot trigger while the trigger is asserted will not save any pre-event data.
Multiple single shot triggers can be saving data simultaneously independently of each other.

If the filename contains .avi then the file created will be an AVI file.
If the filename contains .mp4 then the file created will be a MP4 file.

The trigger ID returned can be used to stop the by calling **MPG4KX_StopSingleShotTrigger**(page **77**)

Specifying an event duration of –1 will cause the data to be saved to disk until stopped by calling **MPG4KX_StopSingleShotTrigger**(page **77**) is called

MPG4KX_StopSingleShotTrigger

int MPG4KX_StopSingleShotTrigger (*nCard*, *nChannel, nID)*

int *nCard*; /* */
int *nChannel*; /* */
int *nID*; /* */

The **MPG4KX_StopSingleShotTrigger** function stops the specified single shot trigger.

Parameter	*Description*
nCard	Specifies which card
nChannel	Specifies for which channel to stop the single shot trigger
nID	Specifies the ID for the single shot trigger to stop

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_FOUND	The specified single shot trigger was not found

Comments

This function will cause the specified single shot trigger event to become de-asserted. The saving to disk will continue for the post trigger buffering duration specified in the call to **MPG4K_EnablePreTrigger** (page **71**).

Callback functions

MPG4K_AddVideoCallback

int MPG4K_AddVideoCallback (*nCard*, *nChannel, *Callback, *lpContext)*

int *nCard*; /* */
int *nChannel*; /* */
void (*Callback)(void *lpContext, unsigned *char *bpData, int nDataSize, int nChannel, int isKeyFrame, int nFrameCnt)*; /* */
void **lpContext*; /* */

The **MPG4K_AddVideoCallback** function adds the supplied callback function to the video callback chain.

Parameter	*Description*
nCard	Specifies which card to add the callback for
nChannel	Specifies which channel to add the callback for
Callback	Specifies the callback function to call for each frame of compressed video data
lpContext	Specifies the context to pass to the callback as the first parameter.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	Callback ID number.

Comments

The callback is called in the same thread context as the data parser so processing in the callback function should not take very long since other callbacks may be waiting to be called. The callback function should not block.
The callback function is called once for each frame of video.
The bpData contains one frame of compressed video. This data complies with ISO 14496-2. Do not modify the data pointed to by this variable.
The isKeyFrame variable passed to the callback equals 1 if the data is an I-frame and 0 if it is a P-frame
The callback can be removed from the callback chain by calling MPG4K_RemoveVideoCallback

MPG4K_AddAudioCallback

int MPG4K_AddAudioCallback (*nCard*, *nChannel, *Callback, *lpContext)*

int *nCard*; /* */
int *nChannel*; /* */
void (*Callback)(void *lpContext, unsigned *char *bpData, int nDataSize ,int nChannel, int isKeyFrame, int nFrameCnt)*; /* */
void **lpContext*; /* */

The **MPG4K_AddAudioCallback** function adds the supplied callback function to the audio callback chain.

Parameter	*Description*
nCard	Specifies which card to add the callback for
nChannel	Specifies which channel to add the callback for
Callback	Specifies the callback function to call for each frame of compressed audio data
lpContext	Specifies the context to pass to the callback as the first parameter.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	Callback ID number.

Comments

The callback is called in the same thread context as the data parser so processing in the callback function should not take very long since other callbacks may be waiting to be called. The callback function should not block.
The bpData contains compressed audio. The format of this data depends on the audio format specified by MPG4K_SetAudioFormat. The default is ADPCM.
Do not modify the data pointed to by this variable.
The isKeyFrame variable passed to the callback equals 1 if the data is an I-frame and 0 if it is a P-frame
The nFrameCnt variable passed to the callback contains the number of audio frames pointed to by bpData
The callback can be removed from the callback chain by calling MPG4K_RemoveAudioCallback

MPG4K_AddMotionCallback

int MPG4K_AddMotionCallback (*nCard*, *nChannel, *Callback, *lpContext)*

int *nCard*; /* */
int *nChannel*; /* */
void (*Callback)(void *lpContext, unsigned *char *bpData, int nDataSize, int nChannel, int isKeyFrame, int nFrameCnt)*; /* */
void **lpContext*; /* */

The **MPG4K_AddMotionCallback** function adds the supplied callback function to the motion detection callback chain.

Parameter	*Description*
nCard	Specifies which card to add the callback for
nChannel	Specifies which channel to add the callback for
Callback	Specifies the callback function to call for each frame of motion detection data
lpContext	Specifies the context to pass to the callback as the first parameter.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	Callback ID number.

Comments

The callback is called in the same thread context as the data parser so processing in the callback function should not take very long since other callbacks may be waiting to be called. The callback function should not block.
The bpData contains one frame of motion detection data.
Do not modify the data pointed to by this variable.
The callback can be removed from the callback chain by calling MPG4K_RemoveMotionCallback
Please see the motion detection section for details of the motion detection data.

MPG4K_RemoveVideoCallback

int MPG4K_RemoveVideoCallback (*nCard*, *nChannel, nID)*

int *nCard*; /* */
int *nChannel*; /* */
int *nID*; /* */

The **MPG4K_RemoveVideoCallback** function removes the supplied callback function from the video callback chain.

Parameter	*Description*
nCard	Specifies which card to remove the callback for
nChannel	Specifies which channel to remove the callback for
nID	Specifies the ID of the callback to remove

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_RemoveAudioCallback

int MPG4K_RemoveAudioCallback (*nCard*, *nChannel, nID)*

int *nCard*; /* */
int *nChannel*; /* */
int *nID*; /* */

The **MPG4K_RemoveAudioCallback** function removes the supplied callback function from the audio callback chain.

Parameter	*Description*
nCard	Specifies which card to remove the callback for
nChannel	Specifies which channel to remove the callback for
nID	Specifies the ID of the callback to remove

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_RemoveMotionCallback

int MPG4K_RemoveMotionCallback (*nCard*, *nChannel, nID)*

int *nCard*; /* */
int *nChannel*; /* */
int *nID*; /* */

The **MPG4K_RemoveMotionCallback** function removes the supplied callback function from the motion detection callback chain.

Parameter	***Description***
nCard	Specifies which card to remove the callback for
nChannel	Specifies which channel to remove the callback for
nID	Specifies the ID of the callback to remove

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_AddPreviewCallback

int MPG4K_AddPreviewCallback (*nCard*, *nChannel, *Callback, *lpContext)*

int *nCard*; /* */
int *nChannel*; /* */
void (*Callback)(void *lpContext, unsigned *char *bpData, int nDataSize, int nChannel, int isKeyFrame, int nFrameCnt)*; /* */
void **lpContext*; /* */

The **MPG4K_AddPreviewCallback** function adds the supplied callback function to the preview callback chain.

Parameter	*Description*
nCard	Specifies which card to add the callback for
nChannel	Specifies which channel to add the callback for
Callback	Specifies the callback function to call for each frame of raw preview data
lpContext	Specifies the context to pass to the callback as the first parameter.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	Callback ID number.

Comments

The callback is called in the same thread context as the onscreen preview so processing in the callback function should not take very long since other callbacks may be waiting to be called. The callback function should not block.
The bpData contains one frame of uncompressed video data.
Do not modify the data pointed to by this variable.
If onscreen preview is not enabled, the format of the preview data can be chosen by calling MPG4K_SetPreviewFOURCC, MPG4K_SetPreviewDepth and MPG4K_SetPreviewPitch
Before using the preview data for the first time, the format should be confirmed by calling MPG4K_GetPreviewFOURCC, MPG4K_GetPreviewDepth and MPG4K_GetPreviewPitch
This function is available in both preview and non-preview versions of the SDK

MPG4K_RemovePreviewCallback

int MPG4K_RemovePreviewCallback (*nCard*, *nChannel, nID)*

int *nCard*; /* */
int *nChannel*; /* */
int *nID*; /* */

The **MPG4K_RemovePreviewCallback** function removes the supplied callback function from the preview callback chain.

Parameter	***Description***
nCard	Specifies which card to remove the callback for
nChannel	Specifies which channel to remove the callback for
nID	Specifies the ID of the callback to remove

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_SetErrorCallback

int MPG4K_SetErrorCallback (*ErrorCallback, lpContext)*

void (*ErrorCallback)(void *lpContext, tErrorCallback *Cause)*; /* */
void **lpContext*; /* */

The **MPG4K_SetErrorCallback** function sets the supplied callback function as the error callback.

Parameter	***Description***
ErrorCallback	Specifies the callback to call with asynchronous errors
lpContext	Specifies the context to pass to the callback as the first parameter

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The tErrorCallback structure will contain card number and channel that the error occurred on. If the error is not channel specific the nChannel element will be -1
The nSource element will be one of the ID_ERRCB_ error numbers.
The nError element will be one of the ID_ERR_ error numbers.
The callback will only be called for errors that don't occur immediately when one of the SDK functions is called, for example changing the AVI filename

Video setting functions

MPG4K_SetBrightness

int MPG4K_SetBrightness (*nCard*, *nChannel*, *bBright)*

int *nCard*; /* */
int *nChannel*; /* */
signed char *bBright*; /* */

The **MPG4K_SetBrightness** function sets the brightness for the specified channel.

Parameter	*Description*
nCard	Specifies the card to set the brightness for
nChannel	Specifies the channel to set the brightness for
bBright	Specifies the new brightness value to use. This value can be between –128 and 127 where positive values are brighter.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_GetBrightness

int MPG4K_GetBrightness (*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_GetBrightness** function returns the current brightness setting.

Parameter	*Description*
nCard	Specifies the card to return the brightness for
nChannel	Specifies the channel to return the brightness for

Returns

The return value is the brightness.

Comments

MPG4K_SetContrast

int MPG4K_SetContrast (*nCard*, *nChannel*, *bContrast)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned char *bContrast*; /* */

The **MPG4K_SetContrast** function sets the contrast for the specified channel.

Parameter	*Description*
nCard	Specifies the card to set the contrast for
nChannel	Specifies the channel to set the contrast for
bContrast	Specifies the contrast.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_GetContrast

int MPG4K_GetContrast(*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_GetContrast** function returns the current contrast setting.

Parameter	*Description*
nCard	Specifies the card to return the contrast for
nChannel	Specifies the channel to return the contrast for

Returns

The return value is the contrast.

Comments

MPG4K_SetHue

int MPG4K_SetHue (*nCard*, *nChannel*, *bHue)*

int *nCard*; /* */
int *nChannel*; /* */
signed char *bHue*; /* */

The **MPG4K_SetHue** function sets the hue for the specified channel.

Parameter	*Description*
nCard	Specifies the card to set the hue for
nChannel	Specifies the channel to set the hue for
bHue	Specifies the hue.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_GetHue

int MPG4K_GetHue(*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_GetHue** function returns the current hue setting.

Parameter	*Description*
nCard	Specifies the card to return the hue for
nChannel	Specifies the channel to return the hue for

Returns

The return value is the hue.

Comments

MPG4K_SetSaturation

int MPG4K_SetSaturation (*nCard*, *nChannel*, *bSaturation)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned char *bSaturation*; /* */

The **MPG4K_SetSaturation** function sets the saturation for the specified channel.

Parameter	*Description*
nCard	Specifies the card to set the hue for
nChannel	Specifies the channel to set the hue for
bSaturation	Specifies the saturation.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

MPG4K_GetSaturation

int MPG4K_GetSaturationU(*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4K_GetSaturation** function returns the current saturation setting for the specified channel.

Parameter	*Description*
nCard	Specifies the card to return the saturation for
nChannel	Specifies the channel to return the saturation for

Returns

The return value is the saturation.

Comments

MPG4K_PowerDecoder

int MPG4K_PowerDecoder (*nCard*, *nChannel*, *nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4K_PowerDecoder** function powers up or down the specified input decoder.

Parameter	*Description*
nCard	Specifies the card to power the decoder
nChannel	Specifies the channel to power up or down
nEnable	Specifies whether to power the specified decoder up or down. This can be one of the following values

Value	*Meaning*
ID_ENABLE	Power up decoder
ID_DISABLE	Power down decoder

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Upon initialisation, only the decoder for channel 0 (input group A) is powered. Selecting modes using **MPG4K_SetMode** (page **22**) or **MPG4K_SetPreviewMode** (page **117**) or changing the routing using **MPG4K_SetInputPath** (page **28**) or **MPG4K_SetPreviewInputPath**(page **118**) will cause the required decoders to be powered up. However, decoders will not be powered down when not used. This function can be used to power down the unused decoders.

MPG4KX_VideoDetected

int MPG4KX_VideoDetected (*nCard*, *nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_VideoDetected** function returns whether a video signal has been detected on the specified channel.

Parameter	*Description*
nCard	Specifies on which card to get the video detected status
nChannel	Specifies which channel to get the video detected status from

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

For the MPEG4000XLP, the return value can also be a logical combination of the following

Value	*Meaning*
ID_NO_HLOCK	The horizontal PLL is not locked.
ID_NO_VIDEO	Video loss has been detected
ID_VIDEO_BLIND	Blind video has been detected. This could signify the camera has been covered

For the MPEG4-CPCI can be a logical combination of the following

Value	*Meaning*
ID_NO_HLOCK	The horizontal PLL is not locked.
ID_NO_VIDEO	Video loss has been detected
ID_NO_SLOCK	The sub-carrier PLL is not locked. The incoming video could be in the wrong format
ID_NO_VLOCK	The vertical lock is not locked. The incoming video could be in the wrong format

Comments

Video Filter functions

The MPEG4000XLP and MPEG4-CPCI has optional video filters built in. These can be used to filter out noise or de-interlace the input video. The filters are within the encoding engine so do not affect the preview.

MPG4KX_EnableFilters

int MPG4KX_EnableFilters (*nCard*, *nChannel*, *nF1, nF2, nF3)*

int *nCard*; /* */
int *nChannel*; /* */
int *nF1*; /* */
int *nF2*; /* */
int *nF3*; /* */

The **MPG4KX_EnableFilters** function enables or disables the filters.

Parameter	*Description*
nCard	Specifies the card to enable the filters on
nChannel	Specifies on which channel to enable the filters for
nF1	Specifies whether to enable Filter 1 This can be one of the following values

Value	*Meaning*
ID_ENABLE	Enable filter
ID_DISABLE	Disable filter

nF2 Specifies whether to enable Filter 2 This can be one of the following values

Value	*Meaning*
ID_ENABLE	Enable filter
ID_DISABLE	Disable filter

nF3 Specifies whether to enable Filter 3 This can be one of the following values

Value	*Meaning*
ID_ENABLE	Enable filter
ID_DISABLE	Disable filter

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Filter 1

Filter 1 is a two-dimensional (horizontal and vertical) de-interlacing filter designed to reduce line flicker and other artefacts due to interlaced video input.

Filter 2

Filter 2 is designed to reduce the thermal noise associated with some camera sensor technologies.

Filter 3

Filter 3 models the input picture and is designed to eliminate the outliner noise that does not conform to this model.

MPG4KX_SetFilterLevel

int MPG4KX_SetFilterLevel(*nCard*, *nChannel*, *nFilter, nLevel)*

int *nCard*; /* */
int *nChannel*; /* */
int *nFilter*; /* */
int *nLevel*; /* */

The **MPG4KX_SetFilterLevel** function sets the level for the specified filter.

Parameter	*Description*
nCard	Specifies the card to set the filter level on
nChannel	Specifies for which channel to set the filter level
nFilter	Specifies which filter to set the level for. This must be 1, 2 or 3
nLevel	Specifies the filter level.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_INVALID_MODE	An invalid filter number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Filter 1

The filter level can be between 0 and 7 (inclusive). Values of 0 to 3 provide two-dimensional filtering. and values 4 to 7 provide filtering in the vertical direction only. A higher number results in stronger filtering.

Filter 2

The filter level is split into two 8-bit values, with bits 0 to 7 of nLevel being the luminance filter level and bits 8 to 15 being the chrominance filter level. The filter level should indicate the expected amount of noise so a higher values provides greater filtering. Valid values for each level are 0 to 31.

Filter 3

The filter level is split into two 3-bit values, with bits 0 to 2 of nLevel being the low frequency filter level and bits 16 to 18 being the high frequency filter. Higher levels for each provide greater filtering. Valid values for each level are 0 to 7.

Motion detection

The MPEG4000XLP supports a basic motion detection method where an object moving out of a 16x16 macro block will be flag that macro block as having motion.
Motion detection data is generated for every video frame that is encoded at the same rate that the encoding takes place.
The motion detection data is packed such that each bit of the data corresponds to one 16x16 macro block, with the bit being set meaning motion has been detected.

MPG4K_EnableMotion

int MPG4K_EnableMotion (*nCard*, *nChannel, nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4K_EnableMotion** function enables or disables the motion detection.

Parameter	***Description***
nCard	Specifies on which card to enable or disable motion detection.
nChannel	Specifies the channel to enable or disable the motion detection on
nEnable	Specifies whether to enable or disable the motion detection This can be one of the following values:

Value	***Meaning***
ID_ENABLE	Enable the motion detection
ID_DISABLE	Disable the motion detection

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The motion detection threshold value controls how sensitive the motion detection is, with 1 being most sensitive and 10 being least sensitive.
Motion detection is performed on all channels independently for ID_QUAD_MODE and ID_MUXD1_MODE.
In ID_SINGLE_MODE, ID_QUAD_MODE_SINGLE and ID_PIP_MODE, motion detection is only performed on channel 0

MPG4KX_SetMotionPreProcessing

int MPG4KX_SetMotionPreProcessing (*nCard*, *nChannel, nEnable)*

int *nCard*; /* */
int *nChannel*; /* */
int *nEnable*; /* */

The **MPG4KX_SetMotionPreProcessing** function sets enables or disables the pre-processing of motion detection data.

Parameter	*Description*
nCard	Specifies on which card to set the motion pre-processing
nChannel	Specifies the channel to set the motion pre-processing for
nEnable	Specifies whether to enable or disable motion detection data pre-processing. This can be one of the following values:.

Value	*Meaning*
ID_ENABLE	Enable the motion detection pre-processing
ID_DISABLE	Disable the motion detection pre-processing

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The motion detection data pre-processing defaults to disabled.
With this enabled, the motion detection callback is called only when the motion detection data changes. Additionally, the pre-processing can also mask out regions of the video to that motion detected in those areas are not passed on. The functions **MPG4KX_SetMotionMask** (page **103**) and **MPG4KX_AddMotionMask** (page **104**) should be used to set up the masks.

MPG4K_SetMotionThreshold

int MPG4K_SetMotionThreshold (*nCard*, *nChannel, bTR)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned char *bTR*; /* */

The **MPG4K_SetMotionThreshold** function sets the threshold value for the motion detection.

Parameter	*Description*
nCard	Specifies on which card to set the motion threshold
nChannel	Specifies the channel to set the motion threshold value for
bTR	Specifies the threshold value for the motion detection. This can be between 1 and 10 inclusive.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The motion detection threshold value controls how sensitive the motion detection is, with 1 being most sensitive and 10 being least sensitive.
Motion detection is performed on all channels independently for ID_QUAD_MODE and ID_MUXD1_MODE.
In ID_SINGLE_MODE, ID_QUAD_MODE_SINGLE and ID_PIP_MODE, motion detection is only performed on channel 0
Please note that motion detection is more sensitive when lower frame rates are used since the changes between frames are more noticeable.

MPG4KX_SetMotionMask

int MPG4KX_SetMotionMask (*nCard*, *nChannel, *bpMask, nLength)*

int *nCard*; /* */
int *nChannel*; /* */
unsigned char **bpMask*; /* */
int nLength; /* */

The **MPG4KX_SetMotionMask** function sets the mask used to mask out areas from the motion detection.

Parameter	*Description*
nCard	Specifies on which card to start the motion detection mask
nChannel	Specifies the channel to set the motion detection mask for
bpMask	Specifies the motion detection mask to use
nLength	Specifies the length in bytes of the motion detection mask pointed to by bpMask.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

The data pointed to by bpMask is in the same form as the motion detection data, each bit corresponds to a 16x16 macro block in the video. The most significant bit of the first byte corresponds to the top left of the video. Having the bit set means that motion detection for that area should be used, having the bit cleared means that motion detection for that area should be masked out.

The motion detection mask set by this function overrides all previous calls to this function or **MPG4KX_AddMotionMask** (page **104**)

MPG4KX_AddMotionMask

int MPG4KX_AddMotionMask (*nCard*, *nChannel, *bpMask, nLength)*

int *nCard*; /* */
int *nChannel*; /* */
RECT *lpRect; /* */

The **MPG4KX_AddMotionMask** function adds the specified area to the motion detection mask.

Parameter	*Description*
nCard	Specifies on which card to start the motion detection mask
nChannel	Specifies the channel to set the motion detection mask for
lpRect	Specifies an area to mask out

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Motion detection pre-processing must be enabled using **MPG4KX_SetMotionPreProcessing** (page **101**) for the motion detection mask to be used
The area specified by this function is masked out so that motion detection in this area is ignored.

Preview functions

MPG4K_StartPreview

int MPG4K_StartPreview (*nCard*, *hWnd)*

int *nCard*; /* */
HWND *hWnd*; /* */

The **MPG4K_StartPreview** function starts the preview engine.

Parameter	*Description*
nCard	Specifies which card to start the preview for.
hWnd	Specifies the window handle of the parent window

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Windows only.

MPG4K_StartPreview

int MPG4K_StartPreview (*nCard)*

int *nCard*; /* */

The **MPG4K_StartPreview** function starts the preview engine.

Parameter	*Description*
nCard	Specifies which card to start the preview for.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

All other operating systems

MPG4K_StopPreview

int MPG4K_StopPreview (*nCard)*

int *nCard*; /* */

The **MPG4K_StopPreview** function stops the preview engine.

Parameter	*Description*
nCard	Specifies which card to stop the preview for

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Under Linux, this function will close the preview window.

MPG4K_EnablePreview

int MPG4K_EnablePreview (*nCard*, *nEnable)*

int *nCard*; /* */
int *nEnable*; /* */

The **MPG4K_EnablePreview** function enables the preview window.

Parameter	*Description*
nCard	Specifies which card to set the private data rate for.
nEnable	Specifies whether to enable or disable the preview window. This can be one of the following values:

Value	*Meaning*
ID_ENABLE	Enable the preview window
ID_DISABLE	Disable the preview window

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

This function should be called after **MPG4K_StartPreview** (page **105**) to show or hide the preview. ID_DISABLE has no effect when used under Linux with the SDL preview engine.

MPG4K_SetPreviewDestination

int MPG4K_SetPreviewDestination (*nCard*, *nDestination)*

int *nCard*; /* */
int *nDestination*; /* */

The **MPG4K_SetPreviewDestination** function specifies the destination of the preview.

Parameter	***Description***
nCard	Specifies which card to set the private data rate for.
nDestination	Specifies the destination of the preview. It is a bit mask of the following values

Value	***Meaning***
ID_PREVIEW_ONSCREEN	Onscreen preview

Returns

The return value is one of the following values

Value	***Meaning***
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Setting nDestination to 0 prevents preview to screen. The data will still be captured and passed to any registered preview callbacks.

MPG4K_PreviewSetColourKey

int MPG4K_SetPreviewColourKey (*nCard, ulColourKey)*

int *nCard*; /* */
unsigned long *ulColourKey*; /* */

The **MPG4K_SetPreviewColourKey** function sets the colour key for the preview window.

Parameter	*Description*
nCard	Specifies which card to set the colour key for
ulColourKey	Specifies the colour key to use

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NO_COLOURKEY	Colour keying not available

Comments

The colour key should be in the correct format for the current display mode.
Colour keying may not be available on all display hardware or in all display modes.
This function does not set the colour when using the SDL preview engine under Linux.
Colour keying is used by default if available on the hardware.

MPG4K_SetPreviewLocation

int MPG4K_SetPreviewLocation (*nCard, lpRect)*

int *nCard*; /* */
RECT **lpRect*; /* */

The **MPG4K_SetPreviewLocation** function sets the location of the preview window.

Parameter	*Description*
nCard	Specifies which card to set the location for
lpRect	Specifies the location of the preview window

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_PREVIEW	Setting the desired location requires a restart of the preview engine.
All other values	DirectX error values

Comments

The location specified in the lpRect is in units of pixels and is relative to the parent windows client area.

This function can be used to position the preview window only after MPG4K_StartPreview has been called.

Under Windows, if **MPG4K_EnablePreview** (page **108**) has not been called, the preview window will not be visible.

This function will scale the preview to be the size specified

This function can also be used to set the preview capture resolution when preview to screen has been disabled.

The width and height of the preview captured may be different from the values passed to this function due to the need for certain memory alignments. **MPG4K_GetPreviewWidth** (page **119**) and **MPG4K_GetPreviewHeight** (page **120**) should be called if the actual width and height are required.

The preview engine will capture up to a maximum of 704x576 for PAL sources and 704x480 for NTSC sources. For windows above this, hardware scaling provided by the display adapter is used where available.

This function does not set the position but does rescale the preview when using the SDL preview engine under Linux.

MPG4K_SetPreviewFOURCC

int MPG4K_SetPreviewFOURCC (*nCard*, *ulFOURCC)*

int *nCard*; /* */
unsigned long *ulFOURCC*; /* */

The **MPG4K_SetPreviewFOURCC** function sets the desired format of the preview data.

Parameter	*Description*
nCard	Specifies which card to set the preview format for
ulFOURCC	Specifies the desired format of the preview data.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_MODE	An invalid FOURCC was passed

Comments

Supported FOURCCs are

Value	*Meaning*
mmioFOURCC('R','G','B','4')	32bits per pixel RGB
mmioFOURCC('R','G','B','3')	24bits per pixel RBG
mmioFOURCC('R','G','B','2')	16bits per pixel RGB
mmioFOURCC('Y','U','Y','2')	YUV 422
mmioFOURCC('U','Y','V','Y')	YUV 422 with different component ordering

If previewing to screen then the format automatically selected takes precedence over the format specified by this function.

24bit per pixel and 32bit per pixel RGB are not supported under QNX.

MPG4K_GetPreviewFOURCC

int MPG4K_GetPreviewFOURCC (*nCard)*

int *nCard*; /* */

The **MPG4K_SetPreviewFOURCC** function returns the current format of the preview data.

Parameter	*Description*
nCard	Specifies from which card to get the preview format

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other values	FOURCC

Comments

The preview data format should be determined using this function before using it.
All RGB formats will return the same FOURCC, which will be mmioFOURCC(' ','R','G','B'). To get the depth of the RGB, call **MPG4K_GetPreviewDepth** (page **116**)

MPG4K_SetPreviewPitch

int MPG4K_SetPreviewPitch (*nCard*, *ulFOURCC)*

int *nCard*; /* */
unsigned long *ulPitch*; /* */

The **MPG4K_SetPreviewPitch** function sets the desired pitch of the preview data.

Parameter	*Description*
nCard	Specifies on which card to set the preview pitch
ulPitch	Specifies the desired pitch of the preview data. This is the number of bytes between the start of consecutive lines in memory.

Returns

The return value is one of the following values

Value	*Meaning*
ID_OK	Success
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised

Comments

Pitch is also sometimes called stride.
Setting the pitch to a value greater than the preview width multiplied by the number of bytes per pixel is useful when saving the data to file formats that pad the end of each video line. BMP is one such format that pads to the next DWORD.

If previewing to screen then the pitch automatically selected takes precedence over the format specified by this function.

MPG4K_GetPreviewPitch

int MPG4K_GetPreviewPitch (*nCard)*

int *nCard*; /* */

The **MPG4K_GetPreviewPitch** function gets the pitch of the preview data.

Parameter	*Description*
nCard	Specifies from which card to get the preview pitch

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	Pitch.

Comments

MPG4K_GetPreviewDepth

int MPG4K_GetPreviewDepth (*nCard)*

int *nCard*; /* */

The **MPG4K_GetPreviewDepth** function returns the current depth the preview data.

Parameter	*Description*
nCard	Specifies from which card to get the preview depth

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	Number of bits per pixel.

Comments

MPG4K_SetPreviewMode

int MPG4K_SetPreviewMode (*nCard*, *nMode)*

int *nCard*; /* */
int *nMode*; /* */

The **MPG4K_SetPreviewMode** function sets the mode for the preview path.

Parameter	*Description*
nCard	Specifies the card to set the mode for.
nMode	Specifies the mode to use. This can be one of the following values.

Value	*Meaning*
ID_QUAD_MODE	Preview the four channels as four separate quadrants, each at CIF resolution
ID_QUAD_MODE_SINGLE	As ID_QUAD_MODE
ID_SINGLE_MODE	Preview only one channel at D1 resolution
ID_MUXD1_MODE	Preview the four channels at D1 resolution cycling between each
ID_PIP_MODE	Preview one channel at D1 resolution and one channel at reduced resolution
ID_DUAL_MODE_SINGLE	Encode the first two channels arranged in two halves as a single file at D1 resolution

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_INVALID_MODE	An invalid mode was specified

Comments

In ID_PIP_MODE, the smaller window is QCIF.

This function sets the mode of the preview path only. To set the capture mode as well, use **MPG4K_SetMode** (page **22**).

MPG4K_SetPreviewInputPath

int MPG4K_SetPreviewInputPath (*nCard*, *nChannel*, *nInput)*

int *nCard*; /* */
int *nChannel*; /* */
int *nInput*; /* */

The **MPG4K_SetPreviewInputPath** function sets the routing for the video inputs in the preview path.

Parameter	*Description*
nCard	Specifies which card to set the input routing for.
nChannel	Specifies the destination channel.
nInput	Specifies the input to route to the specified channel

Returns

The return value is one of the following values.

Value	*Meaning*
ID_OK	Success
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified

Comments

This function is only available on the MPEG4000-XLP.

Input 0 corresponds to input group A

Input 1 corresponds to input group B

Input 2 corresponds to input group C

Input 3 corresponds to input group D

It is possible to route the same input to multiple channels.

This functions sets the routing only for the preview path.

MPG4K_GetPreviewWidth

int MPG4K_GetPreviewWidth (*nCard)*

int *nCard*; /* */

The **MPG4K_GetPreviewWidth** function returns the width (in pixels) the preview engine is capturing the preview data at.

Parameter	*Description*
nCard	Specifies which card to get the preview width for

Returns

The return value is one of the following values.

Value	*Meaning*
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
Other positive values	Width of preview capture in pixels

Comments

This function should be used to retrieve the width of the preview since **MPG4K_SetPreviewLocation** (page **111**) may adjust the actual width to prevent memory alignment issues.

MPG4K_GetPreviewHeight

int MPG4K_GetPreviewHeight (*nCard)*

int *nCard*; /* */

The **MPG4K_GetPreviewHeight** function returns the height (in lines) the preview engine is capturing the preview data at.

Parameter	*Description*
nCard	Specifies which card to get the preview width for

Returns

The return value is one of the following values.

Value	*Meaning*
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
Other positive values	Height of preview capture in pixels

Comments

This function should be used to retrieve the height of the preview since **MPG4K_SetPreviewLocation** (page **111**) may adjust the actual height to prevent memory alignment issues.

MPG4KX_SetNumPreviewBuffers

int MPG4KX_SetNumPreviewBuffers (*nCard, nNumBuffers)*

int *nCard*; /* */
int *nNumBuffers*; /* */

The **MPG4KX_SetNumPreviewBuffers** function sets the number of preview buffers that should be used. Each buffer corresponds to one frame and is queued for use by the preview capture engine

Parameter	*Description*
nCard	Specifies for which card to set the number of preview buffer
nNumBuffers	Specifies the number of buffers to use. Default is 8

Returns

The return value is one of the following values.

Value	*Meaning*
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_ENCODING	The preview engine is currently running
ID_ERR_NOT_SUPPORTED	An invalid number of buffers was specified
ID_OK	Success

Comments

This function cannot be called while the preview engine is running.
If called, this function must be called before **MPG4K_StartPreview** (page **105**).

If using a reduced preview frame rate, the number of buffers must be set to 1.

MPG4KX_SetPreviewFrameRate

int MPG4KX_SetPreviewFrameRate (*nCard, nFrameRate)*

int *nCard*; /* */
int *nFrameRate*; /* */

The **MPG4KX_SetPreviewRate** function sets the desired frame rate for the preview.

Parameter	***Description***
nCard	Specifies for which card to set the preview frame rate
nFrameRate	Specifies the desired frame rate for the preview. This is a maximum and is not guaranteed. This can be one of the following values:

Value	***Meaning for PAL***	***Meaning for NTSC***
0	25	30
1	12.5	15
2	6.25	7.5
3	3.125	3.75
4	1.562	1.865
5	0.781	0.938

Returns

The return value is one of the following values.

Value	***Meaning***
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_CHANNEL	An invalid card number was specified
ID_ERR_NOT_SUPPORTED	An invalid frame rate was specified or the number of preview buffers is not 1
ID_OK	Success

Comments

If using a reduced preview frame rate, the number of buffers must be set to 1 first using **MPG4KX_SetNumPreviewBuffers** (page **121**).

OSD Functions

The MPEG4000-XLP allows text and basic graphics to be overlaid on top of the captured or previewed video. The MPEG4000-XLP supports up to 16 different software loadable fonts. The SDK includes an application to convert a bitmap into the required format.
The path specified in the ulFlags parameter to the OSD functions specify which digital video output from the video decoder they should operate on and not the input to the MPEG4 video encoder. For more details on the digital video paths see the **Hardware block diagram** (page **9**).
Up to 16 rectangles can be drawn onto the OSD using one of 16 colours (12 predefined and 4 custom colours)
The font size is fixed with each glyph being 32 lines high and 16 pixel wide. This gives an available character grid of 45x18 in PAL and 45x15 in NTSC modes.

MPG4KX_LoadFont

int MPG4KX_LoadFont (*nCard, szFont, nPage)*

```
int nCard;          /*  */
char * szFont;      /*  */
int nPage;          /*  */
```

The **MPG4KX_LoadFont** function loads the specified font file for use by the OSD functions.

Parameter	*Description*
nCard	Specifies which card to load the font into
szFont	NULL terminated string specifying the name of the font file to load
nPage	Specifies which font page to load the font into.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The specified filename is invalid
ID_ERR_NOTFOUND	The specified file could not be open
ID_ERR_NOT_SUPPORTED	The specified card does not support font uploading
ID_OK	Success.

Comments

MPG4KX_PrintOSD

int MPG4KX_PrintOSD (*nCard, nX, nY, szString, ulFlags, bColour, bFont)*

int *nCard*; /* */
int *nX*; /* */
int *nY*; /* */
char * *szString*; /* */
unsigned long *ulFlags*; /* */
unsigned char *bColour*; /* */
unsigned char *bFont*; /* */

The **MPG4KX_PrintOSD** function prints the specified string to the OSD.

Parameter	***Description***
nCard	Specifies on which card to print the OSD
nX	Specifies the horizontal location of the first character of the string in character units
nY	Specifies the vertical location of the string in lines
szString	NULL terminated string specifying the string to print to the OSD
ulFlags	Bit mask of the following values:

Value	***Meaning***
ID_PREVIEW	String will show on the preview
ID_CAPTURE	String will show on the capture
ID_OSD_ALPHA	Alpha blend the string
ID_OSD_BLINK	Blink the string
ID_OSD_ODDFIELD	Use the odd field from the font for both fields of the OSD
ID_OSD_EVENFIELD	Use the even field from the font for both fields of the OSD
ID_OSD_BOTHFIELDS	Use both odd and even fields from the font

bColour	Specifies the palette entry to use to print the string. This can be between 0 and 3 inclusive.
bFont	Specifies which font to use.

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The specified string is invalid
ID_OK	Success.

Comments

The font specified must have previously been loaded **using MPG4KX_LoadFont** (page **123**) or forced enabled using **MPG4KX_ForceFont** (page **130**.)
Although the MPEG4000-XLP supports up to 16 different fonts, each line on the display must use the same font.
The OSD consists of 45 characters horizontally and 18 lines in PAL and 15 lines in NTSC.
The colour of the palette entry that bColour corresponds to is set using the **MPG4KX_SetOSDTextColour** (page **132**) function.
The field selection is on a line by line basis and so any subsequent writes to the same line will override the setting.

MPG4KX_BlitOSD

int MPG4KX_BlitOSD (*nCard, nX, nY, bpData, nLen, ulFlags, bColour, bFont)*

```
int nCard;                  /*  */
int nX;                     /*  */
int nY;                     /*  */
unsigned char * bpData;     /*  */
int nLen;                   /*  */
unsigned long ulFlags;      /*  */
unsigned char bColour;      /*  */
unsigned char bFont;        /*  */
```

The **MPG4KX_BlitOSD** function blits the specified buffer to the OSD.

Parameter	*Description*
nCard	Specifies on which card to blit the OSD
nX	Specifies the horizontal location of the first character of the string in character units
nY	Specifies the vertical location of the string in lines
bpData	Pointer to the data to blit to the OSD
nLen	Length, in bytes, of the data pointed to by bpData
ulFlags	Bit mask of the following values:

Value	*Meaning*
ID_PREVIEW	String will show on the preview
ID_CAPTURE	String will show on the capture
ID_OSD_ALPHA	Alpha blend the string
ID_OSD_BLINK	Blink the string

bColour	Specifies the palette entry to use to print the string. This can be between 0 and 3 inclusive.
bFont	Specifies which font to use.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The specified data pointer is invalid
ID_OK	Success.

Comments

The font specified must have previously been loaded using MPG4KX_LoadFont (page **123**) or forced enabled using **MPG4KX_ForceFont** (page **130**.)
Although the MPEG4000-XLP supports up to 16 different fonts, each line on the display must use the same font.
The OSD consists of 45 characters horizontally and 18 lines in PAL and 15 lines in NTSC.

This function can be used to print basic bitmaps to the OSD. The bitmap must have been converted to a font using the supplied tool
The colour of the palette entry that bColour corresponds to is set using the **MPG4KX_SetOSDTextColour** (page **132**) function.

MPG4K_ClearOSD

int MPG4K_ClearOSD (*nCard)*

int *nCard*; /* */

The **MPG4KX_ClearOSD** function clears the OSD.

Parameter	*Description*
nCard	Specifies on which card to clear the OSD

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_OK	Success.

Comments

MPG4KX_ClearOSD

int MPG4KX_ClearOSD (*nCard, ulFlag)*

int *nCard*; /* */
unsigned long *ulFlags*; /* */

The **MPG4KX_ClearOSD** function clears the OSD.

Parameter	***Description***	
nCard	Specifies on which card to clear the OSD	
ulFlags	Bit mask of the following values.	
	Value	***Meaning***
	ID_PREVIEW	Clear the OSD on the preview
	ID_CAPTURE	Clear the OSD on the capture

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The specified filename was invalid
ID_ERR_NOTFOUND	The specified file could not be open
ID_ERR_NOT_SUPPORTED	The specified card does not support font uploading
ID_OK	Success.

Comments

MPG4KX_ForceFont

int MPG4KX_ForceFont (*nCard, nPage, nEnable)*

int *nCard*; /* */
int *nPage*; /* */
int *nEnable*; /* */

The **MPG4KX_ForceFont** function forces the specified font page to be enabled or disabled.

Parameter	*Description*	
nCard	Specifies on which card to set the force font setting	
nPage	Specifies which font page to force enabled or disabled	
nEnable	Specifies whether to enable or disable	
	Value	***Meaning***
	ID_ENABLE	Forces the font to be enabled
	ID_DISABLE	Forces the font to be disabled

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The specified card does not support font forcing
ID_OK	Success.

Comments

Since uploading fonts can be time consuming, this function can be used on subsequent initialisations of the SDK to force a font to be enabled. This can significantly reduce the start up time. It is up to the application to decide whether the font must be uploaded or if it can be forced.
This function must only be used to force font pages that have previously had fonts uploaded to them. Forcing font pages that that have not been uploaded to will result in undefined behaviour.
The OSD functions will not use font pages that have not been enabled.
Uploading a font using the **MPG4KX_LoadFont** (page **123**) function automatically enables the page it uploads to.

MPG4KX_SetOSDCustomColour

int MPG4KX_SetOSDCustomColour (*nCard, nColour, bR. bG, bB)*

int *nCard*; /* */
int *nIndex*; /* */
unsigned char *bR*; /* */
unsigned char *bG*; /* */
unsigned char *bB*; /* */

The **MPG4KX_SetOSDCustomColour** function sets the specified custom colour palette entry to the specified colour.

Parameter	*Description*
nCard	Specifies on which card to set the custom colour
nIndex	Specifies which entry in the custom colour palette to set. This can be between 0 and 3 inclusive
bR	Red component of the desired colour. This value can be in the range 0-255
bG	Green component of the desired colour. This value can be in the range 0-255
bB	Blue component of the desired colour. This value can be in the range 0-255

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The specified card does not support custom colours
ID_OK	Success.

Comments

The custom colours set using this function can be used when drawing rectangles using the **MPG4K_Rectangle** (page **133**) function or displaying text using the **MPG4KX_PrintOSD** (page **124**) or **MPG4KX_BlitOSD** (page **126**) functions.

MPG4KX_SetOSDTextColour

int MPG4KX_SetOSDTextColour (*nCard, bIndex, bColour)*

int *nCard*; /* */
unsigned char b*Index*; /* */
unsigned char *bColour*; /* */

The **MPG4KX_SetOSDTextColour** function sets the specified text colour palette entry to the specified colour.

Parameter	***Description***
nCard	Specifies on which card to set the text colour
bIndex	Specifies which entry in the text colour palette to set. This can be between 0 and 3 inclusive
bColour	Specifies which colour to use. This can be one of the following value

Value	***Meaning***
ID_OSD_WHITE75	White 75% amplitude, 100% saturation
ID_OSD_YELLOW	Yellow 75% amplitude, 100% saturation
ID_OSD_CYAN	Cyan 75% amplitude, 100% saturation
ID_OSD_GREEN	Green 75% amplitude, 100% saturation
ID_OSD_MAGENTA	Magenta 75% amplitude, 100% saturation
ID_OSD_RED	Red 75% amplitude, 100% saturation
ID_OSD_BLUE	Blue 75% amplitude, 100% saturation
ID_OSD_BLACK	0% black
ID_OSD_WHITE100	100% white
ID_OSD_GREY50	50%grey
ID_OSD_GREY25	25% grey
ID_OSD_BLUE75	Blue 75% amplitude, 75% saturation
ID_OSD_CUSTOM0	Custom colour index 0
ID_OSD_CUSTOM1	Custom colour index 1
ID_OSD_CUSTOM2	Custom colour index 2
ID_OSD_CUSTOM3	Custom colour index 3

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The specified card does not support modifying the text colour
ID_OK	Success.

Comments

MPG4K_Rectangle

int MPG4K_Rectangle (*nCard, lpRect, ulFlags)*

int *nCard*; /* */
LPRECT *lpRect*; /* */
unsigned long *ulFlags*; /* */

The **MPG4K_Rectangle** function draws a rectangle on the OSD at the specified location.

Parameter	*Description*	
nCard	Specifies on which card to draw the rectangle	
lpRect	Specifies the location of the corner rectangle in pixels/lines.	
ulFlags	Specifies the flags. This is a combination of the following values	
	Value	***Meaning***
	ID_OSD_RECTANGLEX	Set the colour for rectangle X, where X is 0-15
	ID_OSD_INBORDER	Draw a border around the inside of the rectangle
	ID_OSD_OUTBORDER	Draw a border around the outside of the rectangle
	ID_OSD_PLANE	Fill the rectangle
	ID_OSD_ALPHA	Alpha blend the rectangle with the video
	ID_CAPTURE	Draw the rectangle on the capture
	ID_PREVIEW	Draw the rectangle on the preview

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The specified card does not support rectangle drawing
ID_OK	Success.

Comments

The MPEG4000XLP can display up to 16 rectangles on the OSD.
The colour of each rectangle is specified using the **MPG4K_SetRectangleColour** (page **134**) function.
The position of the top left corner must be a multiple of 2.
The width and height of the rectangle must be a multiple of 4.

MPG4K_SetRectangleColour

int MPG4K_SetRectangleColour (*nCard, nColour, ulFlags)*

int *nCard*; /* */
int *nColour*; /* */
unsigned long *ulFlags*; /* */

The **MPG4K_SetRectangleColour** function sets the colour for the specified rectangle.

Parameter	***Description***
nCard	Specifies on which card to draw the rectangle
nColour	Specifies the colour of the rectangle. This can be one of the following values

Value	***Meaning***
ID_OSD_WHITE75	White 75% amplitude, 100% saturation
ID_OSD_YELLOW	Yellow 75% amplitude, 100% saturation
ID_OSD_CYAN	Cyan 75% amplitude, 100% saturation
ID_OSD_GREEN	Green 75% amplitude, 100% saturation
ID_OSD_MAGENTA	Magenta 75% amplitude, 100% saturation
ID_OSD_RED	Red 75% amplitude, 100% saturation
ID_OSD_BLUE	Blue 75% amplitude, 100% saturation
ID_OSD_BLACK	0% black
ID_OSD_WHITE100	100% white
ID_OSD_GREY50	50%grey
ID_OSD_GREY25	25% grey
ID_OSD_BLUE75	Blue 75% amplitude, 75% saturation
ID_OSD_CUSTOM0	Custom colour index 0
ID_OSD_CUSTOM1	Custom colour index 1
ID_OSD_CUSTOM2	Custom colour index 2
ID_OSD_CUSTOM3	Custom colour index 3

ulFlags Specifies the flags. This is a combination of the following values

Value	***Meaning***
ID_OSD_RECTANGLEX	Set the colour for rectangle X, where X is 0-15
ID_OSD_INBORDER	Set the colour for the inner border
ID_OSD_OUTBORDER	Set the colour for the outer border
ID_OSD_PLANE	Set the colour for the rectangle

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified

ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOT_SUPPORTED	The specified card does not custom colours
ID_OK	Success.

Comments

The border colours are the same across all rectangles.
The colour of the custom colours is set using the **MPG4KX_SetOSDCustomColour** (page **131**) function.

Analogue Output functions

The MPEG4000-XLP has two analogue outputs, video out1 and video out2.

MPG4KX_EnableAnalogueOutput

int MPG4KX_EnableAnalogueOutput (*nCard, nOutput, nEnable)*

int *nCard*; /* */
int *nOutput*; /* */
int *nEnable*; /* */

The **MPG4KX_EnableAnalogueOutput** function enables or disables the specified outputs.

Parameter	*Description*	
nCard	Specifies on which card to enable the outputs	
nOutput	Bit mask of the following values.	
	Value	***Meaning***
	ID_PREVIEW	Specifies the preview output.
	ID_CAPTURE	Specifies the capture output
nEnable	Specifies whether to enable or disable the specified output. Can be a combination of the following	
	Value	***Meaning***
	ID_ENABLE	Enable the specified output
	ID_DISABLE	Disable the specified output

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_OK	Success.

Comments

By default the capture output corresponds to video out1 and the preview output corresponds to video out2. To change these routings, use **MPG4KX_SetAnaloguePath**(page **137**)

MPG4KX_SetAnaloguePath

int MPG4KX_SetAnalogueSet (*nCard, nPath, nInput)*

int *nCard*; /* */
int *nOutput*; /* */
int *nPath*; /* */

The **MPG4KX_SetAnaloguePath** function is

Parameter	***Description***	
nCard	Specifies the card to set the output type for	
nOutput	Specifies the output receive the digital video specified by nPath. Bit mask of the following values.	
	Value	***Meaning***
	ID_PREVIEW	Video out 2
	ID_CAPTURE	Video out 1
nPath	Specifies the video path to route to the specified analogue output	
	Value	***Meaning***
	ID_PREVIEW	Preview path
	ID_CAPTURE	Preview path

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_OK	Success.

Comments

.

Decoding control functions

The MPEG4000-XLP can also act as a MPEG4 decoder.
The current version of the API supports decoding only a single channel at a time.
The decoded video is output on Video Out2 and can also be displayed on VGA using the Preview functions
Decoding is not supported under QNX.

MPG4KX_StartDecode

int MPG4KX_StartDecode (*nCard, nChannel, szSource)*

int *nCard*; /* */
int *nChannel*; /* */
char **szSource*; /* */

The **MPG4KX_StartDecode** function starts decoding the specified source.

Parameter	*Description*
nCard	Specifies on which card to start decoding
nChannel	Specifies the decoder channel. to use
szSource	NULL terminated string specifying the source to decode.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The specified source was invalid
ID_ERR_NOTFOUND	The specified file could not be opened or is in an incompatible format
All other non-negative numbers	ID number of file.

Comments

Currently only AVI files are supported as sources.
Once decoding has finished the analogue output will be switched back to preview output.
Multiple AVI files can be queued for consecutive playback with no gaps between them by calling this function multiple times or by calling **MPG4KX_QueueDecode** (page **140**) function.
The name of the file currently playing can be retrieved using the **MPG4KX_GetDecodeSource** (page **149**) function.
The ID number of the file current playing can be retrieved using the **MPG4KX_GetDecodeSourceID** (page **150**) function.
References to all queue files are stored until playback of the last file finishes. This enables skipping back to previously played files. To remove a file from the queue use the MPG4KX_RemoveDecodeSource function
szSource can be NULL is sources have previously been queue using **MPG4KX_QueueDecode** (page **140**) function.

MPG4KX_StopDecode

int MPG4KX_StopDecode (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_StopDecode** function stops the decoding.

Parameter	*Description*
nCard	Specifies on which card to stop decoding
nChannel	Specifies the decoder channel. to use

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_OK	Success.

Comments

This function will not return until the decoding has stopped

MPG4KX_QueueDecode

int MPG4KX_QueueDecode (*nCard, nChannel, szSource)*

int *nCard*; /* */
int *nChannel*; /* */
char **szSource*; /* */

The **MPG4KX_QueueDecode** function adds the specified source to the decode queue.

Parameter	*Description*
nCard	Specifies on which card to start decoding
nChannel	Specifies the decoder channel. to use
szSource	NULL terminated string specifying the source to decode.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_POINTER	The specified source was invalid
ID_ERR_NOTFOUND	The specified file could not be opened or is in an incompatible format
All other non-negative numbers	ID number of file.

Comments

Currently only AVI files are supported as sources.

MPG4KX_DecoderRunning

int MPG4KX_DecoderRunning (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_DecoderRunning** function returns whether the decoder is currently running.

Parameter	***Description***
nCard	Specifies on which card to return the decoder status
nChannel	Specifies the decoder channel. to use

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
0	The decoder is not running
1	The decoder is running.

Comments

MPG4KX_GetDecodeDuration

__int64 MPG4KX_GetDecodeDuration (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_GetDecodeDuration** function returns the duration of the current source in microseconds.

Parameter	***Description***
nCard	Specifies from which card to get the decode duration
nChannel	Specifies the decoder channel. to use

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
-1	The decode source is live or has no duration information
All other positive values	The duration of the source in microseconds

Comments

Under Linux the return type is long long

MPG4KX_GetDecodeLocation

__int64 MPG4KX_GetDecodeLocation (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_GetDecodeLocation** function returns the current decoding position of the current source in microseconds.

Parameter	*Description*
nCard	Specifies from which card to get the decode location
nChannel	Specifies the decoder channel. to use

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
-1	The decode source is live or has no duration information
All other positive values	The current decode position of the source in microseconds

Comments

Under Linux the return type is long long

MPG4KX_SetDecodeFrameRate

int MPG4KX_SetDecodeFrameRate (*nCard, nChannel, nFrameRate)*

int *nCard*; /* */
int *nChannel*; /* */
int *nFrameRate*; /* */

The **MPG4KX_SetDecodeFrameRate** function sets the playback rate of the decode.

Parameter	*Description*
nCard	Specifies on which card to set the decode frame rate
nChannel	Specifies the decoder channel. to use
nFrameRate	Specifies the frame rate to decode at. This is in the form of a integer divider of the sources input standard frame rate. Full frame rate is 65536.

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_INVALID_FRAMERATE	The specified frame rate is invalid
ID_OK	Success

Comments

Changing the decode frame rate can be used to slow down or speed up video playback.
Audio and video will not be in sync unless playback is at the rate it was recorded.
The maximum frame rate is 25fps for PAL sources and 29.97 for NTSC sources.
The rate passed to this function will be used for all subsequent decode sessions
Calling with a rate of 0 or 1 while decoding is ongoing will cause the rate to be set back to the source rate
Calling with a rate of 1 before decoding causes the source rate to be used.

The frame rate can be calculated from the nFrameRate using

$$OutputFPS = \frac{nFrameRate * InputFPS}{65536}$$

Alternatively, the nFrameRate value can be calculated for a specified output frame rate using

$$nFrameRate = \frac{OutputFPS * 65536}{InputFPS}$$

MPG4KX_GetDecodeFrameRate

int MPG4KX_GetDecodeFrameRate (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_GetDecodeFrameRate** function returns the current playback rate of the decode.

Parameter	***Description***
nCard	Specifies from which card to get the decode frame rate
nChannel	Specifies the decoder channel. to use

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other positive values	The current decode playback rate

Comments

MPG4KX_DecoderCommand

int MPG4KX_DecoderCommand (*nCard, nChannel, ulCommand)*

int *nCard*; /* */
int *nChannel*; /* */
int *ulCommand*; /* */

The **MPG4KX_DecoderCommand** function sends a command to the decode engine.

Parameter	***Description***
nCard	Specifies which card to send the decode command to
nChannel	Specifies the decoder channel. to use
ulCommand	Specifies the command to send. The command is made up of two parts: command and parameter. The command is the least significant 8bits. The parameter is the remaining 24 bits.

Command	***Parameter***	***Meaning***
ID_DECODE_PLAY	none	Resume normal decode
ID_DECODE_PAUSE	none	Pause the decode
ID_DECODE_STEP	none	Single frame step
ID_DECODE_FF	rate	Fast forward
ID_DECODE_RW	rate	Rewind

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_OK	Success

Comments

The commands are queued so multiple calls will queue commands.
Fast forward and rewind both take a rate parameter. In fast forward and rewind mode, only the I-frames within the MPEG4 stream are played. The rate parameter specifies the number of I-frames to skip between played frames. A rate of 1 plays all I-frames, 2 plays every 2nd I-frame, etc. The normal rate is 0.
Single step mode will pause the decode and increment one frame for every single step command.

MPG4KX_DecoderCommandEx

int MPG4KX_DecoderCommandEx (*nCard, nChannel, Command)*

int *nCard*; /* */
int *nChannel*; /* */
tDecodeCommand *Command*; /* */

The **MPG4KX_DecoderCommandEx** function sends a command to the decode engine.

Parameter	***Description***
nCard	Specifies which card to send the decode command to
nChannel	Specifies the decoder channel. to use
Command	Specifies the command to send..

Command	***Meaning***	***Parameters***
ID_DECODE_PLAY	Resume normal decode	None
ID_DECODE_PAUSE	Pause the decode	None
ID_DECODE_STEP	Single frame step	None
ID_DECODE_FF	Fast forward	Rate
ID_DECODE_RW	Rewind	Rate
ID_DECODE_SEEK	Seek	Location

Returns

The return value is one of the following values

Value	***Meaning***
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_OK	Success

Comments

The commands are queued so multiple calls will queue commands.
For commands that take parameters, the Command.ulFlags and Command.Param elements must be filled in correctly.

ID_DECODE_FW and ID_DECODE_RW

The commands take a parameter to specify the rate.
A rate of 1 plays all I-frames, 2 plays every 2nd I-frame, etc. The normal rate is 0.
This can be passed using the bParam, usParam, ulParam or nParam entries in the Param element.
A negative rate reverses the direction.

ID_DECODE_STEP

Single step mode will pause the decode and increment one frame for every single step command.
Single step can be combined with ID_DECODE_FW or ID_DECODE_RW to step to each I-frame.

ID_DECODE_SEEK

The seek will only happen within the current decoded file. Seeking past the start or end of a file will have no effect.
To seek based on the number of frames, the Command.ulFlags must have ID_DECODE_SEEK_FRAME set. The parameter must be an integer and contain the number of frames to seek.
To seek based on time the Command.ulFlags must have ID_DECODE_SEEK_TIME set. The parameter must be an int64 and specifies the time in microseconds to seek.

The seeking can be performed relative to the current location or as an absolute location within the file. The default is absolute. The specify relative seeking the Command.ulFlags should have ID_DECODE_SEEK_RELATIVE set.
If doing relative seeking, negative parameters will seek backwards from the current location.
All seeking is done to the nearest I-frame. Backward seeks go to the nearest I-frame before the requested location. Forward seeks go the nearest I-frame after the requested location.

MPG4KX_GetDecodeSource

char *MPG4KX_GetDecodeSource (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_GetDecodeSource** function returns the name of the file currently being decoded.

Parameter	*Description*
nCard	Specifies from which card to get the file currently being decoded
nChannel	Specifies the decoder channel to use

Returns

The return value is one of the following values

Value	*Meaning*
NULL	No file is currently being decoded
Other values	NULL terminated string specifying the file currently being decoded

Comments

MPG4KX_GetDecodeSourceID

int MPG4KX_GetDecodeSourceID (*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_GetDecodeSourceID** function returns the ID number of the file currently being decoded.

Parameter	*Description*
nCard	Specifies from which card to get the file currently being decoded
nChannel	Specifies the decoder channel to use

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
All other non-negative numbers	ID number

Comments

MPG4KX_RemoveDecodeSource

int MPG4KX_DecodeSource (*nCard, nChannel, nID)*

int *nCard*; /* */
int *nChannel*; /* */
int *nID*; /* */

The **MPG4KX_RemoveDecodeSource** function removes the specified file from the decode queue.

Parameter	*Description*
nCard	Specifies from which card to remove the file from the decode queue
nChannel	Specifies the decoder channel to use
nID	Specifies the ID of the file to remove from the queue

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOTFOUND	The specified file could not be found
ID_OK	Success

Comments

MPG4KX_ClearDecodeQueue

int MPG4KX_ClearDecodeQueue(*nCard, nChannel)*

int *nCard*; /* */
int *nChannel*; /* */

The **MPG4KX_ClearDecodeQueue** function empties the decode queue.

Parameter	*Description*
nCard	Specifies from which card to remove the file from the decode queue
nChannel	Specifies the decoder channel to use

Returns

The return value is one of the following values

Value	*Meaning*
ID_ERR_INVALID_CHANNEL	An invalid card or channel number was specified
ID_ERR_NODEVICE	No hardware was detected or the library not initialised
ID_ERR_NOTFOUND	The specified file could not be found
ID_OK	Success

Comments

Technical Support

The AMP office can be reached in a number of ways. The preferred medium for support issues is via electronic mail using the address given at the end of this section.

Our address is:

Advanced Micro Peripherals Ltd.
Unit 1, Harrier House,
Sedgeway Business Park
Witchford,
Ely,
Cambridgeshire,
CB6 2HY

Fax: +44 (0)1353 659600

The Advanced Micro Peripherals web site can be found at http://www.ampltd.com

E-mail enquiries of a support manner can be sent to support@ampltd.com

Function Index

M